FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Engineering and Management of heterogenous Smart City Application Ecosystems

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der technischen Wissenschaften

eingereicht von

Johannes Michael Schleicher
Martikelnummer 0125876

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.Prof. Schahram Dustdar

Diese Dissertation haben begutachtet:

(Univ.Prof. Schahram Dustdar)	(Univ.Prof. Andreas Voigt)	(Univ.Prof. Uwe Zdun)

Wien, 24.01.2017

(Johannes Michael Schleicher)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Engineering and Management of heterogenous Smart City Application Ecosystems

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der technischen Wissenschaften

by

Johannes Michael Schleicher
Registration Number 0125876

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.Prof. Schahram Dustdar

The dissertation has been reviewed by:

(Univ.Prof. Schahram Dustdar)	(Univ.Prof. Andreas Voigt)	(Univ.Prof. Uwe Zdun)

Wien, 24.01.2017

(Johannes Michael Schleicher)

Erklärung zur Verfassung der Arbeit

Johannes Michael Schleicher
Keutschacherstrasse 116, 9220 Velden

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

The research in this thesis was supported by the *Wiener Stadtwerke Group* through the interdisciplinary doctoral college URBEM[1] as well as by the *Bundesministerium für Verkehr, Innovation und Technologie - BMVIT* through the research project SIMULTAN.

[1] http://urbem.tuwien.ac.at

Danksagung

Als Erstes gilt mein Dank meinem Betreuer und Doktorvater, Prof. Schahram Dustdar, nicht nur für die Möglichkeit meine Dissertation an der Distributed Systems Group (DSG) zu schreiben, sondern auch für den Freiraum meine Forschungsideen großteils eigenständig verfolgen und umsetzen zu dürfen. Vor allem aber für wichtige Reflexion und wesentliche Impulse in vielen Bereichen, die es mir erlaubt haben diese Arbeit erfolgreich abzuschliessen.

Ein grosses Dankeschön geht auch an Herrn Prof. Andreas Voigt für seine Unterstützung während URBEM und dafür, dass er sich dazu bereit erklärt hat, diese Arbeit zu begutachten. In diesem Sinne möchte ich mich auch recht herzlich bei Herrn Prof. Uwe Zdun bedanken, der sich ebenfalls als Gutachter zur Verfügung gestellt hat.

Ein ganz besonderes Dankeschön gilt meinen Freunden und Kollegen, allen voran Christian Inzinger, Michael Vögler und Andreas Katzian. Christian und Michael danke ich nicht nur für die Beteiligung an allen Forschungsarbeiten und den damit verbundenen interessanten Diskussionen, sondern vor allem für die wichtige Motivation in den entscheidenden Momenten. Andreas danke ich für den notwendigen Freiraum und die Unterstützung in allen anderen Belangen, diese Arbeit wurde erst durch euch möglich.

Ich möchte mich bei allen Beteiligten an der Kooperation zwischen TU Wien und Wiener Stadtwerken nicht nur für die Finanzierung der vorliegenden Arbeit, sondern vor allem für den Mut und die Möglichkeit zur interdisziplinären Forschung, sowie den größtenteils reibungslosen Ablauf bedanken. Ein besonderes Dankeschön geht in diesem Rahmen an Herrn Mag. Christian Schlemmer für die Unterstützung von Seiten der Wien IT, sowie an Herrn Prof. Bednar und Frau Dr. Weinwurm für die Organisation und Abwicklung des Doktoratkollegs. Natürlich geht ein Dankeschön an meine URBEM-Kollegen und Kolleginnen, Sara Fritz, Nikolaus Rab, Nadine Haufe, Manuel Ziegler, Dominik Bothe, Thomas Kaufmann, Peter Eder-Neuhauser, Christina Winkler, Paul Pfaffenbichler und Julia Forster, für die gute Zusammenarbeit und alles darüber hinaus.

Zu guter Letzt gilt mein ganz besonderer Dank meiner Familie, allen voran meiner grossartigen Mutter und meiner wunderbaren Frau. Ihr wart mir Halt, Fundament, Inspiration und Motivation auf mehr Ebenen als ich es jemals hier zum Ausdruck bringen könnte.

Abstract

The recent advent and rapid success of the smart city paradigm has led to its widespread adoption in cities and their supporting ecosystems around the globe. Spear headed by multiple international research initiatives more and more vital aspects of cities are becoming smart. This opens up a vast array of new application possibilities, but also brings along several novel challenges. Various areas like infrastructure, industry, government and most importantly citizens, of a smart city generate large amounts of data and create sophisticated tangled interactions leading to ever increasing complexity. Stakeholders in the smart city domain not only face the challenges of managing these complex systems themselves, they also need to be able to make informed decisions based on the massive amounts of data smart cities generate. In order for stakeholders to stay on top of this emerging complexity, while still seeing the big picture in this dynamic environment, it is essential to provide a holistic interdisciplinary view on the city. To enable stakeholders to build applications that enable such a view, they need to be able to focus on their respective area of expertise without the burden of dealing with underlying complexities that arise from the large scale nature of smart cities. This calls for sensible abstractions that hide the complexities of operating, managing, and running complex smart city applications in a similar way as today's mobile application ecosystems do.

In this thesis we present novel approaches for infrastructure, operations, and data management for enabling such smart city applications ecosystems. First, we present an approach for infrastructure-agnostic artifact deployment that allows easy integration of heterogenous smart city infrastructures as well as the independent evolution of smart city applications and infrastructures. This enables a broader integration of, as well as an easy migration between, infrastructures for smart city applications. To address the key challenge of data management, we present an approach for modeling and management of data sources in the smart city domain. Our approach allows for efficient, distributed data access for applications and introduces a simple technology-agnostic description of data sources for stakeholders. This mechanism enables the exposure of relevant data sources not only for other stakeholders in the same city, but also in other smart cities around the globe significantly extending the application spectrum of smart city applications. In the context of operation management, we present solutions to enable and improve the operation as well as evolution of smart city applications, while respecting the complex security and compliance constraints. We introduce a service mobility approach that can enable the execution, as well as significantly improve the results of, distributed analytical environments. Additionally, we present a method for continuous evolution of container application deployments, capable of integrating security and compliance constraints. By doing so, we further enable the use of software containers for building smart city applications, leading to a substantial increase in flexibility for

developers and operations teams alike. We integrate these approaches into a comprehensive middleware toolkit, the Smart City Operating System (SCOS) that serves as the central element for a Smart City Application Ecosystem (SCALE). We discuss the URBEM Smart City Application as an example of such an smart city application utilizing SCOS to realize a holistic interdisciplinary decision support system for smart cities. Finally, we evaluate our approach in the context of multiple scenarios and show that the contributions of this thesis significantly improve infrastructure, operations and, data management in smart city application ecosystems.

Kurzfassung

Das Konzept intelligenter Städte und dessen rasanter Erfolg haben dazu geführt, dass immer mehr Städte weltweit dieses Konzept aufnehmen und in ihre Ökosysteme integrieren. Unter dem Mantel internationaler Forschungsinitiativen werden immer mehr bedeutende Aspekte dieser Städte intelligenter. Dies eröffnet einerseits eine Vielzahl neuer Anwendungsmöglichkeiten, bringt aber andererseits auch zahlreiche neue Herausforderungen mit sich. Immer mehr Bereiche, wie Infrastruktur, Industrie, Regierung und vor allem die Bürger, einer intelligenten Stadt erzeugen große Datenmengen und schaffen anspruchsvolle komplizierte Interaktionen, die zu einem immer komplexer werdenden System führen. Akteure im Umfeld von intelligenten Städten stehen nicht nur vor der Herausforderung, diese komplexen Systeme selbst zu verwalten, sondern müssen auch fundierte Entscheidungen treffen können, die auf den massiven Datenmengen basieren, die intelligente Städte erzeugen. Damit sie sich in diesem dynamischen Umfeld den komplexen Herausforderungen stellen können und dabei immer noch den Überblick behalten, ist es unerlässlich ihnen eine ganzheitliche interdisziplinäre Sicht auf die Stadt zu ermöglichen. Um Akteuren die Möglichkeit zu geben Anwendungen zu erstellen die eine solche Sicht ermöglichen, müssen sie sich auf ihr jeweiliges Fachgebiet konzentrieren können, ohne dabei auf zugrunde liegende Details der komplexen Smart City Struktur achten zu müssen. Dies erfordert sinnvolle Abstraktionen, die die Komplexität des Betriebs, der Verwaltung und der Ausführung komplexer Smart City Anwendungen in ähnlicher Weise verbergen wie heutige mobile Anwendungsökosysteme.

In dieser Arbeit stellen wir eine Reihe von neuen Ansätzen für Infrastruktur, Betriebs- und Datenmanagement vor, um so die Schaffung von Ökosystemen für Smart City Anwendungen zu ermöglichen. Zunächst präsentieren wir einen Ansatz zur infrastruktur-agnostischen Bereitstellung von Artefakten, der eine einfache Integration heterogener physischer Smart City Infrastrukturen, sowie die unabhängige Weiterentwicklung von Smart City Anwendungen und solcher physischer Infrastrukturen ermöglicht. Dies erlaubt eine breitere Integration von Infrastrukturen, sowie die einfache Migration von Smart City Anwendungen zwischen unterschiedlichen physischen Infrastrukturen. Um die zentrale Herausforderung des Datenmanagements zu lösen, stellen wir einen Ansatz zur Modellierung und Verwaltung von Datenquellen in intelligenten Städten vor. Unser Ansatz ermöglicht effiziente, verteilte Datenzugriffe für Smart City Anwendungen und führt eine einfache technologie-agnostische Beschreibung von Datenquellen für Akteure ein. Durch diesen Mechanismus ermöglichen wir die einfache Bereitstellung von relevanten Datenquellen nicht nur für andere Akteure in der gleichen Stadt, sondern auch in anderen intelligenten Städten rund um den Globus, wodurch sich das Anwendungsspektrum von Smart City Anwendungen deutlich erweitert. Weiters stellen wir Lösungen vor, die den Betrieb und die Entwicklung von Smart City Anwendungen, unter Beachtung der komplexen Sicherheitseinschränkungen sowie in Überein-

stimmung mit generellen Richtlinien, ermöglichen und verbessern. Wir stellen einen Ansatz zur Servicemobiliät vor, der die Ausführung von sogenannten verteilten analytischen Umgebungen ermöglicht und deren Ergebnisse deutlich verbessern kann. Darüber hinaus präsentieren wir ein Verfahren zur kontinuierlichen Weiterentwicklung von Containeranwendungen unter Berücksichtigung von Sicherheitsstandards und allgemeiner Richtlinien. Dadurch ermöglichen wir den Einsatz von Software-Containern für die Entwicklung von Smart City Anwendungen, was zu einer deutlichen Steigerung der Flexibilität für Entwickler und Betriebsteams führt. Wir integrieren diese Ansätze in einem umfassenden Middleware-Toolkit, dem "Smart City Operating System" (SCOS), das als zentrales Element für ein "Smart City Application Ecosystem" (SCALE) dient. Wir diskutieren die URBEM Smart City Anwendung als Beispiel für eine Smart City Anwendung die auf dem SCOS basiert, um ein ganzheitliches interdisziplinäres System zur Entscheidungsunterstützung für intelligente Städte zu realisieren. Schließlich evaluieren wir unsere Ansätze im Rahmen mehrerer Szenarien und zeigen, dass die Beiträge dieser Arbeit die Infrastruktur, den Betrieb und das Datenmanagement in Ökosystemen für Smart City Anwendungen deutlich verbessern.

Contents

List of Tables

List of Figures

List of Publications

The work presented in this thesis is based on research that has been published in the following conference papers, journal articles, and technical reports. For a full publication list of the author please refer to the website at http://dsg.tuwien.ac.at/staff/jschleicher.

- Johannes M Schleicher, Michael Vögler, Christian Inzinger, and Schahram Dustdar. Towards the internet of cities: A research roadmap for next-generation smart cities. In *Proceedings of the ACM First International Workshop on Understanding the City with Urban Informatics*, pages 3–6. ACM, 2015

- Johannes M Schleicher, Michael Vögler, Schahram Dustdar, and Christian Inzinger. Enabling a smart city application ecosystem: requirements and architectural aspects. *IEEE Internet Computing*, 20(2):58–65, 2016

- Michael Vögler, Johannes M Schleicher, Christian Inzinger, Schahram Dustdar, and Rajiv Ranjan. Migrating smart city applications to the cloud. *IEEE Cloud Computing*, 3(2): 72–79, 2016

- Johannes M Schleicher, Michael Vögler, Christian Inzinger, and Schahram Dustdar. Smart fabric-an infrastructure-agnostic artifact topology deployment framework. In *2015 IEEE International Conference on Mobile Services*, pages 320–327. IEEE, 2015

- Johannes M Schleicher, Michael Vögler, Christian Inzinger, Waldemar Hummer, and Schahram Dustdar. Nomads-enabling distributed analytical service environments for the smart city domain. In *Web Services (ICWS), 2015 IEEE International Conference on*, pages 679–685. IEEE, 2015

- Johannes M Schleicher, Michael Vögler, Christian Inzinger, and Schahram Dustdar. Modeling and management of usage-aware distributed datasets for global smart city application ecosystems. *PeerJ Computer Science, under review*, 2016

- Johannes M Schleicher, Michael Vögler, Christian Inzinger, and Schahram Dustdar. Smart brix—a continuous evolution framework for container application deployments. *PeerJ Computer Science*, 2:e66, 2016

- Johannes M Schleicher, Michael Vögler, Christian Inzinger, Sara Fritz, Manuel Ziegler, Thomas Kaufmann, Dominik Bothe, Julia Forster, and Schahram Dustdar. A holistic,

interdisciplinary decision support system for sustainable smart city design. In *International Conference on Smart Cities*, pages 1–10. Springer International Publishing, 2016

- J. M. Schleicher, M. Vögler, S. Dustdar, and C. Inzinger. Application architecture for the internet of cities: Blueprints for future smart city applications. *IEEE Internet Computing*, 20(6):68–75, Nov 2016. doi:10.1109/MIC.2016.130

CHAPTER 1

Introduction

Todays cities are home to more than half of the world's population, making them the primary habitat of modern society. With urbanization still at rise, this number will continue to grow, putting another staggering 2.5 billion people [27] into metropolis around the globe by the end of 2050. This massive strain on cities' infrastructure and resources, is making it essential to become smarter and more efficient in dealing with them. To tackle these challenges metropolises have turned to information and communications technology (ICT), which under the umbrella term smart city is transforming them to ever-evolving, complex cyber physical systems of systems. While the first concept of a smart city focused on cities utilizing communication technologies to deliver services to their citizens, it quickly evolved to using information technology to be smarter and more efficient about the utilization of all of their resources. Initially, resources were limited to fields that were tangible, mainly energy and mobility systems. In recent years, however, not only what can be done with information technology has changed significantly, but also the resources and areas addressable by a smart city have broadened significantly. They now cover areas like smart buildings, smart traffic systems and roads, autonomous driving, energy hubs, electric car utilization, water/waste and pollution management as well as concepts like urban farming. This not only demonstrates the diversity of relevant, previously untapped areas, but also illustrates the dynamics of this field and the importance to be able to incorporate new directions as they emerge. Additionally, sophisticated tangled interdisciplinary interactions, combined with huge heaps of data generated by various areas like infrastructure, industry, government and of course the citizens, lead to an ever increasing complexity inherent to this domain. Stakeholders in the smart city domain not only face the challenges of managing these complex systems themselves, they also need to be able to make informed decisions based on the massive amounts of data smart cities generate. In order for them to stay on top of this emerging complexity, while still seeing the big picture in this dynamic environment, it is essential to provide a holistic interdisciplinary view on the city.

In this thesis we draw a map to enable such a holistic interdisciplinary view overcoming not only cognitive compartmentalization, but also paving the path for an Internet of Cities [89]. We

draw this map based on recent research experience, as well as lessons learned during URBEM[1], a smart city initiative focusing on the city of Vienna. URBEM focuses on enabling the path to a sustainable, supply-secure, affordable and livable smart city by providing holistic interdisciplinary views on the city based on real world scenarios in close cooperation with domain experts and industry stakeholders. Based on this foundation we envisioned the Smart City Application Ecosystem (SCALE) [91] a system for building holistic applications for the smart city domain shown in Figure 1.1.

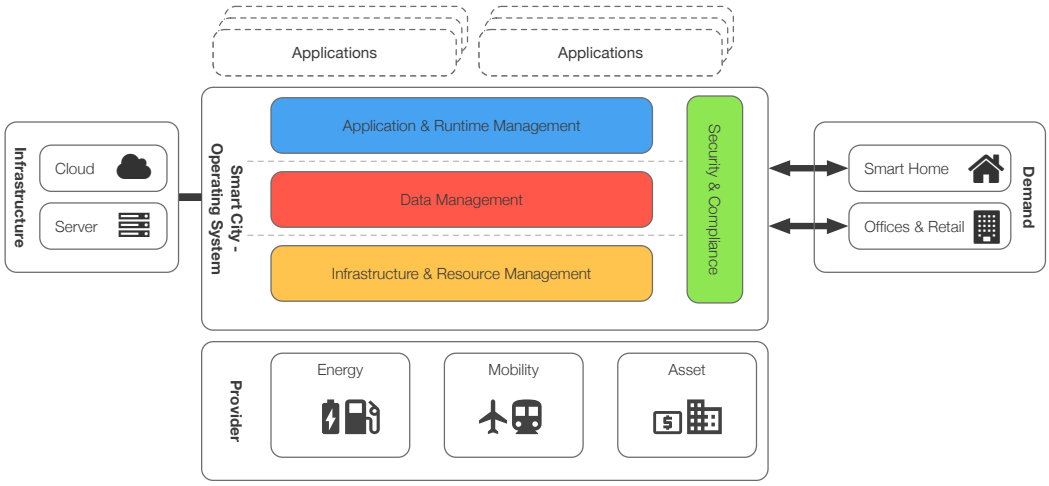


Figure 1.1: Smart City Application Ecosystem Overview

SCALE enables practitioners and stakeholders in a smart city domain to focus on the development of novel smart city applications without the necessity to concern themselves with the specifics of infrastructure, data or application management, while still respecting vital aspects like security and compliance. In this thesis, we present novel approaches for infrastructure, operations and data management of smart city applications to enable SCALE.

1.1 Problem Statement

In order for stakeholders to build and operate smart city applications that enable holistic management and planning for smart cities, they have to respect the characteristics of this domain. The city itself emits massive amounts of data from a plethora of sources including Internet of Things (IoT) networks, documents and citizens. This data is created by large scale infrastructures and data providers, which need to be operated and managed. Additionally, the generated data itself needs to be managed and made accessible so it can be used for analytics and planning purposes. To understand this complex multifaceted data, stakeholders rely on domain experts, who provide analytics and models about various aspects of the city which in turn are fueled by this data. The analytics and models of these domain experts utilize complex tool chains and rely on dynamic

[1] http://urbem.tuwien.ac.at/

emergent interactions to deliver their results. These results are the fundament for decision support relevant for industry and governance stakeholders as well as citizens to manage and plan for the city. To enable stakeholders to address all these aspects when building applications they need to be able to focus on their respective area of expertise without the burden of dealing with underlying complexities. This calls for sensible abstractions that hide the complexities of operating, managing and running complex smart city applications in a similar way as today's mobile application ecosystems do. To enable such a smart city application ecosystem it is therefore necessary to provide layers of abstraction for infrastructure, data, and application management by addressing the following challenges.

First, such an application ecosystems needs to provide the ability to manage and operate a large number of applications as well as data providers in a smart city. This not only calls for means to stage, deploy, organize and evolve applications, but also requires abilities to adapt to the diverse infrastructure landscape common in smart cities. Additionally, the enormous heterogeneity and large number of providers with different infrastructures along with the rapid pace of infrastructure evolution, call for means to move applications and their respective deployment topologies seamlessly between providers (e.g., from a dedicated server setup to a hosted cloud and vice versa).

Second, in order to enable the distributed analytical models that provide the essential baseline for informed planning in the smart city domain, it is vital to ensure that these models can access and process the necessary data. This diverse and complex data, distributed over different providers, combined with the dynamic, emergent interactions between these models, calls for smart management mechanisms. Among other things, these mechanisms need to provide the ability to easily expose heterogenous data as well as to enable such distributed analytical environments, while still respecting the security and compliance constraints of this domain.

Finally, in addition to security concerns, the smart city domain requires mechanisms that ease the enforcement of complex ownership and compliance constraints. Especially in the context of this domain, there are unique security and compliance constraints. These constraints apply on many different levels ranging from company regulations to governmental restrictions and from city to state level. To enable smart city applications that can evolve along these changing requirements it is therefore essential to provide mechanisms that are able to respect them.

1.2 Research Questions

The problems identified in Section 1.1 serve as motivation for the research conducted throughout this thesis. Specifically, this work addresses the following research questions.

Research Question I:
How can smart city application topologies be deployed while specifically respecting the heterogenous, evolving infrastructure landscape in smart city environments?

As discussed in Section 1.1, smart city environments consist of heterogeneous large scale infrastructure landscapes that are a natural consequence of multi-provider environments. In order for smart city applications to be operated and deployed in this context, they need to be able to

respect the diverse infrastructure requirement that arise from these environments. Additionally, they have to deal with the fact that these infrastructures evolve over time. On top of that, smart city applications themselves are complex and dynamic artifacts with strict Quality of Service (QoS) requirements, that change over time. Finally, the dependency on specific infrastructures (e.g., specific cloud providers) can be problematic due to volatility in offered services, availability and pricing. While approaches like DevOps [44] and Infrastructure as Code [43] simplify application provisioning by integrating deployment directives into the development process, infrastructure evolution is currently not considered. Additionally, current approaches for software engineering and lifecycle management do not sufficiently support the independent evolution of infrastructure alongside an application. In order to enable broad deployment of smart city applications, we need approaches that address these challenges by enabling independent evolution of infrastructure and applications.

Research Question II:
How can complex, heterogenous smart city data sources be efficiently exposed in distributed smart city environments?

The ability to access and process data is a crucial element to enable smart city applications. In smart city environments data sources are complex, heterogenous, and scattered between a magnitude of different stakeholders. This fact, combined with the inherently distributed nature of smart city applications, alongside their strict QoS requirements, and emergent nature of data access, poses several challenges. On the one hand smart city applications need a way to incorporate new data sources as they evolve, to either enhance the quality of their results or to be applied to new cities or novel domains. They need to be able to do so, while still ensuring that QoS requirements are met without major reengineering efforts. On the other hand the large number of different stakeholders is mostly bound by the complex constraints of their specific environment. This leads to situations where stakeholders are unable to expose the data because of their inability to ensure constraint enforcement, especially in light of emergent access by smart city applications. To overcome these problems, we need mechanisms that enable ubiquitous data exposure allowing efficient and transparent distributed access, while still respecting essential security and compliance constraints.

Research Question III:
How can smart city applications be operated and evolved while respecting the complex and changing compliance requirements in smart city environments?

The smart city domain consists of large number of different stakeholders leading to many complex ownership and compliance constraints. In order to successfully operate and evolve smart city applications it is essential to respect these constraints. The dynamic and complex nature of smart city applications combined with these constraints however, leads to several challenges in the context of operation and evolution management. The emergent interactions, common in smart city applications, make it difficult to ensure constraint enforcement, potentially hindering essential operation. Additionally, the heterogenous, large-scale nature of smart city applications has made

operating system virtualization based on containers a natural fit to address this complexity. The rapid adoption of this paradigm in combination with the introduction of new applications, new application components, and updates to existing ones have led to a huge number of containers. This has created fragmented large-scale container environments, which in combination with the complex compliance constraints makes it challenging to adhere to all relevant security, and regulatory requirements. These challenges call for novel approaches that ensure the operation as well as evolution of smart city applications while ensuring the ability to respect and enforce compliance constraints.

1.3 Scientific Contributions

The work conducted during the course of this thesis, guided by the research questions introduced in Section 1.2, has led to the following contributions.

Contribution I:
An approach for infrastructure-agnostic artifact topology deployment of smart city applications

To deploy smart city applications in smart city environments they need to be able to handle the heterogenous infrastructure landscapes of this domain. These diverse infrastructures as well as the applications deployed on them evolve over time, which leads to changing, potentially conflicting requirements that make it necessary to separate applications and infrastructures from each other. This calls for approaches that allow the seamless migration of application topologies between deployment targets as well as the ability to enable independent, parallel evolution of both, applications and underlying infrastructures. We present a methodology and toolset to enable infrastructure-agnostic deployment of artifact topologies based on a declarative, constraint-based specification of the required deployment infrastructure. Details are presented in Chapter 4. Contribution I was originally presented in [88].

Contribution II:
An approach for modeling and management of usage-aware distributed datasets for global smart city application ecosystems

Existing smart city application models assume that produced data is managed by and bound to its original application. Such data silos have emerged, in part due to the complex security and compliance constraints governing the potentially sensitive information produced by current smart city applications. While it is essential to enforce security and privacy constraints, we have observed that smart city data sources can often provide aggregated or anonymized data that can be released for use by other stakeholders or third parties. This is especially promising, as such data sources are not only relevant for other stakeholders in the same city, but also other smart cities around the globe. We present a methodology and framework to enable transparent and efficient distributed data access for data sources in smart city environments. Details are presented in Chapter 5. Contribution II was originally presented in [92].

Contribution III:
An approach for enabling distributed analytical service environments for the smart city domain

Distributed Analytical Environments (DAEs) represent a core element when enabling holistic smart city applications. To successfully operate, these DAEs rely on dynamic service composition, based on specific questions from stakeholders to deliver the desired results. The complex compliance constraints in the smart city domain however, lead to situations where this compositions cannot be satisfied, severely impacting the quality of the DAE's result. We deliver a comprehensive problem formalization identifying service mobility as the core issue and present a framework to significantly improve satisfiability and therefore result quality of DAEs. Details are presented in Chapter 6. Contribution III was originally presented in [90].

Contribution IV:
An approach for continuous evolution of container application deployments in smart city application ecosystems

The numerous benefits of container-based solutions for building smart city applications have led to a rapid adoption of this paradigm in this domain. The ability to package application components into self-contained artifacts has brought substantial flexibility to developers and operation teams alike. However, to enable this flexibility, practitioners need to respect numerous dynamic security and compliance constraints, as well as manage the rapidly growing number of container images that arise from introducing new smart city applications. We present a framework enabling continuous evolution of container application deployments, supporting traditional continuous integration processes as well as custom, domain-relevant processes, to implement security and compliance checks. Details are presented in Chapter 7. Contribution IV was originally presented in [93]

1.4 Organization of this Thesis

The remainder of this thesis is structured as follows.

- Chapter 2 provides background information on basic concepts used in this thesis, specifically the smart city and cloud computing concepts.
- Chapter 3 introduces the URBEM scenario that will be used throughout this thesis to motivate and evaluate our contributions.
- Chapter 4 discusses an approach for infrastructure-agnostic deployment of artifact topologies based on a declarative, constraint-based specification of the required deployment infrastructure.
- Chapter 5 introduces an approach to enable transparent and efficient distributed data access for data sources in smart city environments.

- Chapter 6 introduces a framework that allows service mobility in constrained dynamic composition contexts to enable distributed analytical environments.
- Chapter 7 presents an approach for continuous monitoring and evolution of container based smart city applications.
- Chapter 8 details the smart city application ecosystem architecture along with its major components.
- Chapter 9 introduces the URBEM smart city application for holistic decision support in smart cities.
- Chapter 10 presents architectural blueprints for building future smart city applications.
- Chapter 11 concludes this thesis, discusses the presented contributions in light of the identified research questions, and offers an outlook for ongoing and future research.

CHAPTER 2

Background

In this chapter, we introduce several basic concepts that are used in the remainder of this thesis. We start with illustrating the fundamental properties of the smart city concept, followed by an introduction to cloud computing along with noteworthy facets of these concepts that are relevant in the context of this thesis.

2.1 Smart City

The idea of a smart city started with integrating information and communication technologies (ICT) [75] to deliver services to its citizens. While the initial concept was limited to areas like energy and mobility, it soon gained momentum not only covering more and more vital areas of a city, but also focusing on becoming smarter and more efficient about the general usage of a cities' resources. This led to a rapid rise of the smart city paradigm [38] around the globe, both in industry and governance and brought along numerous smart city research initiatives like the IBM Smarter Planet Initiative[1], the MIT City Science program[2] or Trinity's Smart & Sustainable Cities initiative[3]. Figure 2.1 shows a conceptual overview of a smart city along with abstract layers that are integral in the evolution of modern cities towards smart cities. The rapid advancement of ICT has brought along a plethora of enabling technologies, ranging from smart sensors and smart meters to social media platforms and wearables to name only a few. These enabling technologies are key elements in making more and more areas of a city smart. These areas are referred to as verticals or vertical solutions and cover industry & manufacturing, building & traffic management, energy management, and even urban mining & farming. Finally, in order to enable all these novel vertical solutions, they need to be deployed and operated on heterogenous infrastructure stacks ranging from legacy servers to novel cloud solutions.

The main goal of a smart city, however, remains the same as for a city in general: To increase the quality of life of its citizens by providing them with all the services they need, while utilizing

[1] http://www.ibm.com/smarterplanet/us/en/

[2] http://cities.media.mit.edu/about/initiative

[3] https://www.tcd.ie/research/themes/smart-sustainable-cities/

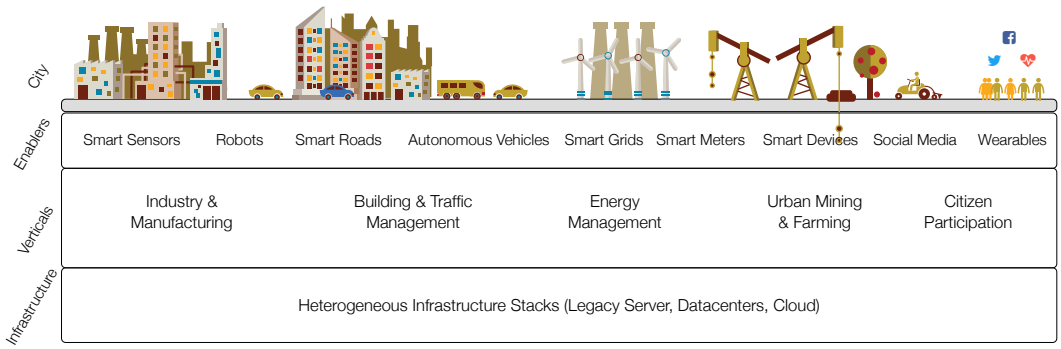


Figure 2.1: Smart City Overview (adapted from [106])

the available resources to achieve this in the most efficient and effective way. The key to enable this, is to understand the citizens' needs and to manage, decide and plan accordingly in a reactive way. With the pervasive adoption of the smart city paradigm in conjunction with the rapid evolution of ICT, the ability to do so has significantly improved through novel possibilities in terms of available information and potential actions. Smart cities provide way more detailed and accurate information about nearly every facet of a city ranging from smart buildings, smart traffic systems and roads, energy management to water/waste and pollution management. Utilizing this information in the right way enables a holistic management and planning approach, which ensures a higher quality of life for a city's citizens. Additionally, ICT enables a close interaction between city officials and citizens allowing for user participation in the evolution of the city. In the context of smart city this reactive system can be illustrated along the Smart City Loop [89] shown in Figure 2.2.

The city itself emits massive amounts of data from a plethora of sources including Internet of Things (IoT) networks, systems, documents, and citizens. This diverse static and dynamic data needs to be managed and made accessible to provide the foundation for planning and decision processes. The complexity of the city in turn makes it necessary to rely on the interaction of domain experts to analyze and model different aspects of the city, based on this data in oder to enable a holistic understanding. This in turn is the foundation of integrated planning and decision processes for industry and governance stakeholders as well as citizen, which influences and impacts the evolution of the city. To enable such a loop, smart cities face several challenges [76], some of them have been outlined in Section 1.1. In addition to these challenges, it is vital to respect the evolutionary nature of a city itself. It makes a smart city a large cluster of many different stakeholders with different technologies, ICT infrastructures and requirements, leading to significant fragmentation that makes the establishment of holistic approaches a challenging task.

2.2 Cloud Computing

In recent years the cloud computing paradigm [63, 82] has become an established and successful concept, both in industry and research. According to the US National Institute of Standards and

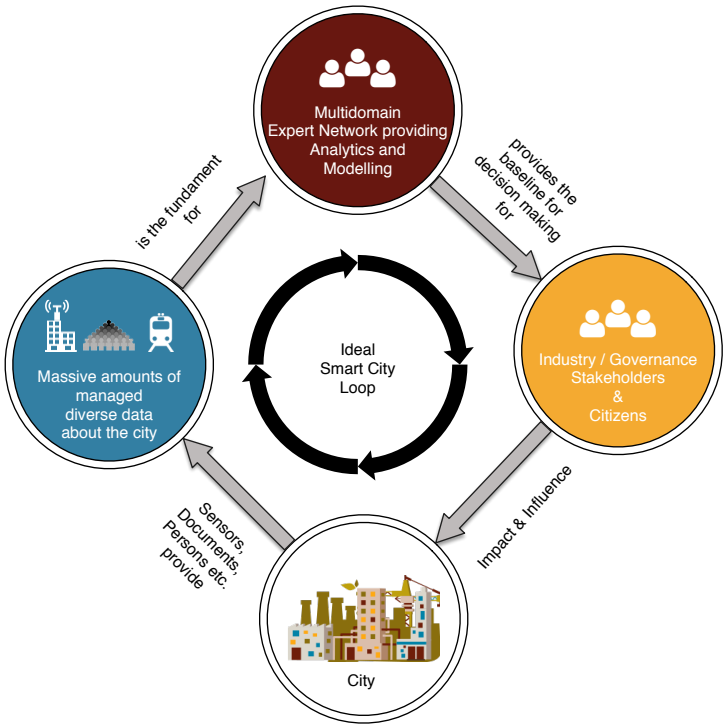


Figure 2.2: Smart City Loop

Technology (NIST) cloud computing is defined as “a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [69]. The cloud computing model has the following essential characteristics, which distinguish it from traditionally provisioned data center infrastructures:

- *On-demand self-service*, allowing consumers to quickly and easily request an arbitrary, seemingly unbound number of computing resources.
- *Broad network access*, which means that capabilities are provided over a network in order to access them via standard mechanisms, promoting heterogenous use.
- *Resource pooling*, allowing to pool resources to serve multiple consumers using a multi-tenant model, based on customer demand.
- *Rapid elasticity*, allowing the elastic provisioning and release of capabilities to enable rapid outward and inward scaling based on demand.
- *Measured Service*, enabling transparency for providers and consumers through monitoring, controlling, and reporting of resource usage.

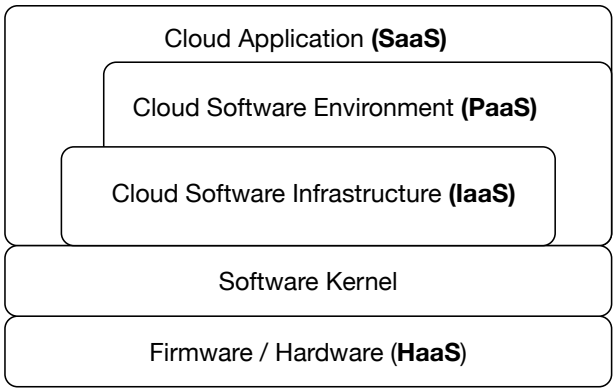


Figure 2.3: Cloud Computing stack (adapted from [119])

Based on these characteristics, it is possible to build elastic applications that are able to dynamically adjust resources based on current demand. These services are offered by providers in different granularities, ranging from low level infrastructure offerings, such as Virtual Machines (VMs), to platforms that are partially managed by providers, up to services that are fully-managed by providers and only consumed by customers. According to NIST these different levels of granularity can be categorized into three different service models and are illustrated in Figure 2.3.

Infrastructure as a Service (IaaS) allows customers to provision computing resources than can be used to deploy and run arbitrary software, such as operating systems and applications. In this model the customer has full control over the instances and is responsible for managing operating system and deployed applications, while the provider only controls the underlying cloud infrastructure. This model brings a high degree of flexibility for the customer but also requires them to have the necessary know-how, as well as human resources to actually manage the instances. Popular examples of IaaS offerings are Amazon Elastic Compute Cloud EC2[4], Google Compute Engine[5] or Microsoft Windows Azure Compute[6]. The *Platform as a Service (PaaS)* offerings were created to reduce the administration overhead of managing instances by providing pre-configured software environments for customers to develop and deploy applications. In this model the provider manages the underlying infrastructure as well as the installed software, while customers control the deployed applications and possibly configuration settings for the application-hosting environment. With PaaS models customers sacrifice some of the flexibility of IaaS for the simplicity and convenience of relying on provider-managed infrastructures. Notable examples of PaaS offerings are Heroku[7], Google App Engine[8] or Amazon Elastic Beanstalk[9]. Finally, the *Software as a Service (SaaS)* model allows customers to use applications, which are

[4]https://aws.amazon.com/ec2/
[5]https://cloud.google.com/compute/
[6]https://www.windowsazure.com/en-us/services/virtual-machines
[7]https://www.heroku.com
[8]https://cloud.google.com/appengine
[9]https://aws.amazon.com/de/elasticbeanstalk/

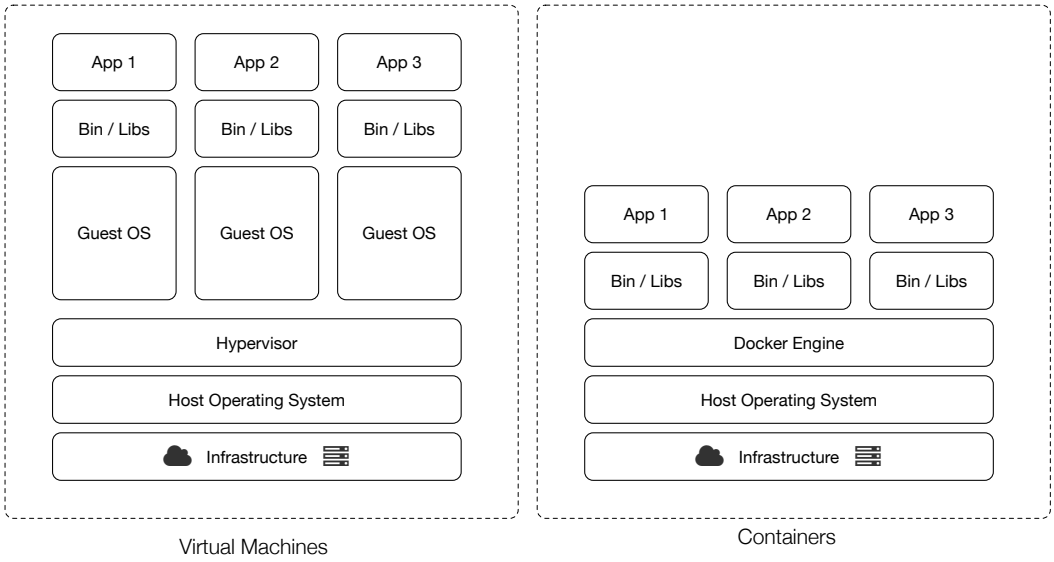


Figure 2.4: Virtual Machines vs Containers based on Docker (adapted from [28])

offered by providers and running on their cloud infrastructure. In this model the provider manages and controls the underlying infrastructure, software and applications, while the customer only uses the application. In contrast to Iaas and PaaS, SaaS offerings are targeted at end users as opposed to application developers or operators, and usually created to meet certain business needs in specific domains. Examples of such SaaS offerings are Dropbox[10], Google Drive[11] or services like Basecamp[12]. Additionally, three major deployment models can be distinguished in the context of cloud computing, which are relevant in the smart city domain. *Private Clouds*, where the cloud infrastructure is exclusively used by a single organization, managed by said organization or any third party and either exists on or off premise. *Public Clouds* where the cloud infrastructure is provisioned for open use by the general public and exists on the premises of the cloud provider. Finally, so called *Hybrid Clouds* where the cloud infrastructure is a composition of two or more distinct cloud infrastructures (e.g., public and private cloud).

In the context of smart city, due to the previously mentioned fragmentation, all of the previously mentioned service and deployment models can be found and need to be respected.

Virtualization plays a central role in providing cloud offerings. Traditionally this has been based on the use of VMs. In recent years however, operating system virtualization based on containers [102] as a mechanism to deploy and manage complex, large-scale software systems has gained significant momentum. A comparison of these two approaches is illustrated in Figure 2.4.

In contrast to VMs, which include the application, binaries and libraries as well as the entire guest operating system, the container approach is more light weight. Containers only include the

[10] https://www.dropbox.com/
[11] https://www.google.com/drive/
[12] https://basecamp.com/

application along with its dependencies, but share the kernel with other containers, running as isolated processes in user space on the host operating system. This leads to several benefits: containers use less ram, start instantly and are constructed from layered filesystems sharing common files, which makes disk usage and image downloads more efficient. Additionally, they provide several advantages for developers. Using containers, they can create self-contained images of application components along with all dependencies that are then executed in isolation on top of a container runtime (e.g., Docker[13], rkt[14], or Triton[15]). This packaging of application components into self-contained artifacts ensures that the application runs everywhere the container runtime is present. In the context of fragmented smart city environments, container based solutions are an important valuable addition for the development and operation of smart city applications.

[13] https://www.docker.com/
[14] https://github.com/coreos/rkt
[15] https://www.joyent.com/

CHAPTER 3

Motivating Scenario

In this chapter, we present the scenario as well as high-level requirements that we will use throughout this thesis to motivate and evaluate the contributions of this work.

3.1 Urban Energy and Mobility System

The "Urban Energy and Mobility System" (URBEM[1]) is a doctoral program that was established as a joint smart city research initiative between the Wiener Stadtwerke Holding AG (Vienna Public Utilities Company) and the Vienna University of Technology. Its goal is to enable the path to a sustainable, supply-secure, affordable, and liveable smart city based on the example of the city of Vienna. It aims at creating an interactive analytical environment that enables the understanding of the multidimensional aspects of the city and their interactions in the context of diverse and complex scenarios. To achieve this goal the college follows an interdisciplinary approach evolving around the integration of experts in different domains including mobility, energy, building physics, sociology, economics, spacial planning, and information technology. Specifically, certain aspects of the city are being analyzed via models and simulations, which fueled by current data and visualized in an interactive 4D environment, should enable a holistic insight. By doings so it aims to provide novel insights and approaches in the following areas:

- Analysis of energy consumption and mobility behavior.
- Optimizing choice of transport in urban areas, incorporating multi modal transportation models.
- Developing sustainable methods for the renovation of existing building stock, as well as for the construction of new buildings.
- Optimizing thermal, natural and electrical energy systems throughout the whole city.

[1] http://urbem.tuwien.ac.at

- Advancing and planing ICT structures to control urban energy supply networks.
- Analyzing business and economic factors as well as managing the risk of urban energy and mobility systems.
- Involvement and strong participation of various stakeholders in planning and decision processes through virtual environments.

An ICT oriented view on this topic is shown in Figure 3.1. In order to provide the aforementioned holistic view of the city the models of domain experts need to interact in a highly dynamic fashion, based on specific questions of the stakeholders, enabled by current data about the city, available in a plethora of different forms, formats and originating from different sources. The problem space in URBEM can therefore be considered a Distributed interactive Analytical Environment (DAE). Additionally, URBEM also covers the main elements in the previously mentioned Smart City Loop (shown in Figure 2.2), which make it a representative example for challenges in this context. URBEM specifically as well as DAE's in an abstract sense pose a plethora of new challenges and problems.

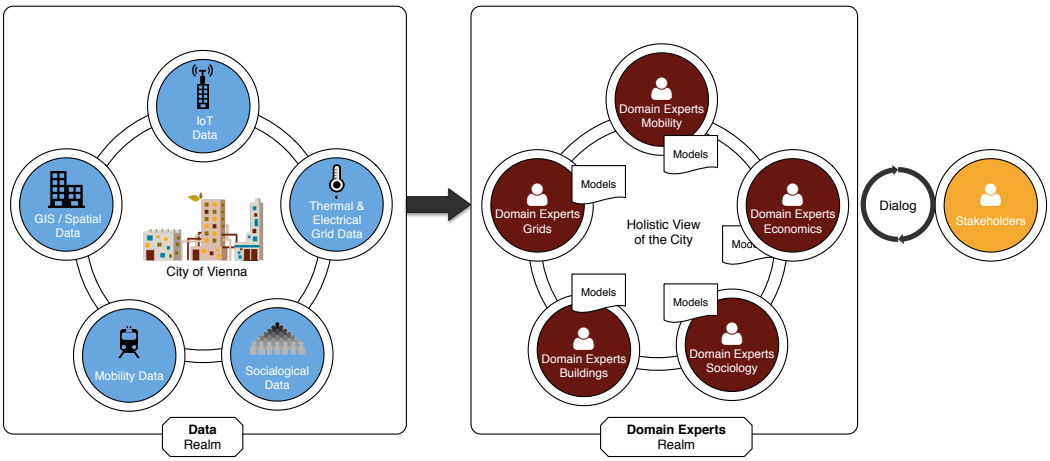


Figure 3.1: ICT oriented URBEM Overview

This starts with the tasks of integrating and handling diverse, very large distributed datasets. In order to build their models and simulations, domain experts rely on current and accurate data, which they are able to process in the formats and forms common to their domain. This data can either be low frequency static data like GIS and grid data or high frequency very dynamic IoT sensor data. It is being emitted from a wide array of different devices, locations, sources and entities including persons. This data needs to be accessed, managed, integrated, linked and transformed on demand in order to provide a coherent basis for utilization in subsequent models and simulations. This poses several challenges in the areas of efficient storage of heterogeneous data and calls for novel unified approaches for data management. Additionally, the multi stakeholder environment of the city itself leads to a fragmentation of relevant data between many different stakeholders.

Furthermore, data is not always available due the volatility of sources as well as security and privacy concerns of participating stakeholders. Such cases require the ability to utilize data from alternative sources (e.g., other cities) in order to ensure reliability of model building and execution. Additionally, data sources from other cities can be valuable input for planning and decision support when incorporated into models and analytics. Therefore, it is necessary to investigate ways to efficiently use such data surrogates when other sources fail to deliver or simply drop out. The massive amounts and diversity of data also makes it necessary to introduce novel and easily graspable (if possible automatic) mechanisms for discovery of data as well as format transformations in order to use them in models and simulations.

In the domain experts realm there needs to be support for dynamic interactions based on variable requests. This makes it necessary to coordinate temporal diverse complex models with a high variability of possible input and output formats to present coherent consumable results. These models and simulations itself rely on different infrastructures and software stacks. In order to support these dynamic elastic processes and infrastructures, on demand staging and scaling mechanisms need to be enabled. Existing cloud computing and cloud analytics environments allow for novel ways of efficient execution and management. Due to the highly dynamic nature, the diverse software stacks for modeling, as well as the different runtime requirements call for novel mechanisms to formulate and address these concerns on the infrastructure level.

Additionally, the DAE itself needs to be designed for evolution, as it is expected to remain in service over longer periods of time. Stakeholder requirements, models and simulations, software stacks or infrastructure constraints can change over time. This requires fundamental changes to the application implementation, its architecture, and deployment topology. Application management needs to account for these issues and assist stakeholders in this domain in creating reliable, fault tolerant applications that are able to dynamically adapt to changes in their runtime environment, as well as support them in enabling evolutionary changes in the application structure and environment.

Finally, such a system has to deal with several non functional aspects beyond QoS. Specifically these are concerns of compliance, provenance and security. The data itself, albeit necessary to run a model or simulation, might underly certain constraints and might not be able to be accessed. This calls for abilities to enable data access respecting these constraints as well as for means to transfer capabilities (e.g. models, services) on an infrastructure level so they can run with the full data they need where they are allowed to. Another relevant aspect is the factor of costs, since these models and simulations can be computationally very expensive. Therefore, the allocation should try to minimize costs and optimize the utilization of resources.

CHAPTER 4

An Infrastructure-Agnostic Artifact Topology Deployment Framework

In this chapter, we present Smart Fabric, a methodology and accompanying toolset for infrastructure-agnostic deployment of application artifact topologies based on a constraint-based, declarative specification of the required deployment infrastructure. Our framework allows for seamless migration of application topologies between deployment targets and enables independent, parallel evolution of both, applications and underlying infrastructure. We discuss the feasibility of the proposed methodology and prototype implementation using representative applications from the Internet of Things and smart city domains.

4.1 Introduction

The recent emergence of the cloud computing paradigm [11] allows stakeholders to leverage a utility-oriented, on-demand approach to create applications that elastically respond to changes in request load, do not depend on dedicated operations teams on site, and can be managed and evolved without upfront infrastructure investments. Despite the apparent benefits, companies and government agencies are still hesitant to migrate business-critical applications to the cloud [49], mainly due to concerns related to vendor lock-in [51, 99] and service availability. While cloud services are designed to provide abstractions that shield stakeholders from the underlying physical infrastructure, applications must nevertheless be specifically tailored to concrete cloud providers to make use of the offered services.

This strong dependence on specific cloud providers is problematic for several reasons. While cloud providers now usually offer service level agreements[1] (SLAs) that provide certain guar-

[1] e.g., https://cloud.google.com/compute/sla, http://aws.amazon.com/ec2/sla/, http://azure.microsoft.com/en-us/support/legal/sla/

antees for service availability to customers, the terms governing customers' use of the offered services can still be unilaterally changed by providers. Changes in offered services or pricing can occur at any time (e.g., retiring offerings, changing pricing structures, introducing new offerings that better suit customers' requirements) and customers may need to migrate their applications to different services or providers as a result. Additionally, in the context of fragmented smart city environments, smart city applications need the ability to run on, as well as to move between, heterogenous infrastructures in order to ensure pervasive deployment and utility. Current approaches for software engineering and lifecycle management do not sufficiently support the independent evolution of infrastructure alongside an application. While approaches like DevOps [44] and Infrastructure as Code [43] simplify application provisioning by integrating deployment directives into the development process, infrastructure evolution is not currently considered.

We argue that deployment infrastructure and application development must be clearly separated to allow for seamless, independent evolution of application components as well as the underlying infrastructure. In this chapter, we present Smart Fabric, a methodology and toolset for infrastructure-agnostic deployment of artifact topologies based on a declarative, constraint-based specification of the required deployment infrastructure. Smart Fabric extends the MADCAT [46] methodology with an abstraction layer that cleanly separates application artifacts from the concrete deployment infrastructure, along with mechanisms to seamlessly migrate application topologies between deployment targets. We illustrate the feasibility of the proposed framework and prototype using representative applications from the Smart City and Internet of Things (IoT) domain.

The remainder of this chapter is structured as follows. In Section 4.2 we discuss the foundational system model underlying our approach. Section 4.3 presents the Smart Fabric framework for infrastructure-agnostic deployment of artifact topologies, followed by a detailed discussion and validation of the framework properties in Section 4.4. Related research is discussed in Section 4.5 and we conclude the chapter with a comprehensive summary in Section 4.6.

4.2 System Model

In order to enable an infrastructure independent artifact deployment framework we introduce an abstraction to model and describe the relevant entities in our domain. As foundation for the abstraction we use the MADCAT [46] methodology, as it introduces several abstract concepts suitable for describing application topologies independent of their deployment targets. Specifically, these are *Technical Units (TU)* to describe applications and their components, as well as *Deployment Units (DU)* to describe how to deploy them on cloud infrastructure. This however is not sufficient for complex, large-scale applications, such as in the Smart City domain, since we need to cover a wider array of different infrastructures (e.g., legacy systems on premises, edge devices, etc.) and deployment types (e.g., scaling across infrastructure boundaries).

To address this limitation we extend the existing concepts with *Infrastructure Specifications (IS)* to capture infrastructure heterogeneity and *Deployment Instances (DI)* to represent the current state of an application deployment topology during runtime. Additionally, we provide an implementation of the abstract concepts along with the proposed extensions of the MADCAT

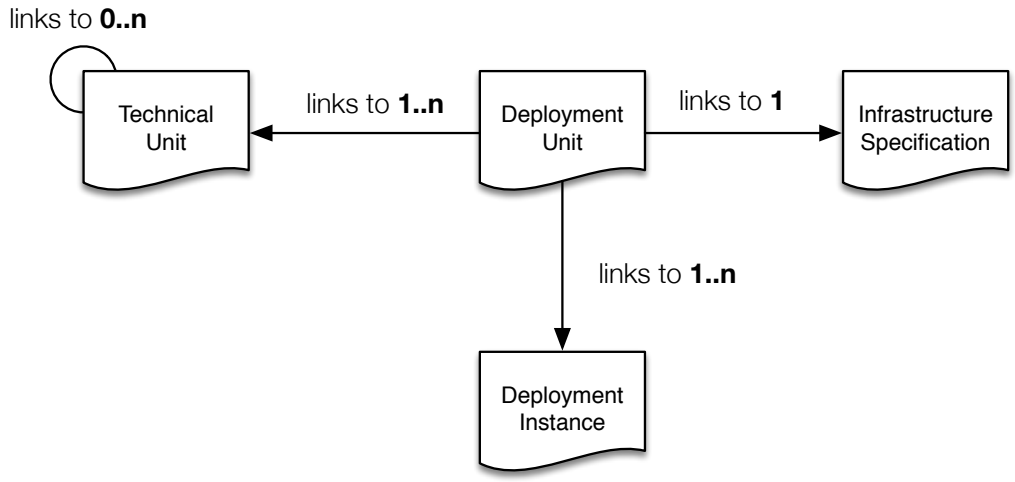


Figure 4.1: Relations between TU, DU, IS and DI

methodology for the Smart Fabric framework. We choose JSON-LD[2] as data format for our concept description as a simple, both human and machine readable, pragmatic, and extensible representation that also allows us to interlink the relevant concepts with each other. Figure 4.1 outlines the relations of the newly introduced *IS* and *DI* to MADCAT concepts *TU* and *DU*. In the following, we discuss the introduced concepts in more detail.

4.2.1 Technical Unit

TUs describe applications as well as application components and focus on the technical aspects of these artifacts. Listing 4.1 shows an example of such a *TU* for a citizen information system based on the Ruby on Rails framework. As JSON-LD document, a *TU* can start with a `context` to set up common namespaces. This is followed by a `type` and `name` to identify the unit. The `artifact-uri` points to the necessary resource to build and execute the artifact, which in turn is described in the `build` and `execution` sections. Both sections are composed of step descriptions as a flexible and extensible way of describing build and execution processes that do not depend on any specific technology. Each `step` is numbered to provide ordering, specifies a `tool` mandatory to perform the step (later used in dependency resolution), as well as a command (`cmd`) to be executed. The `verification` section follows the same step format and serves to verify a successful build, deployment and execution of an artifact. By default, verification steps follow the UNIX philosophy, considering commands that exit with a result code of 0 to be successful, whereas other result codes are interpreted as errors. It further augments the previously introduced step elements with an `expected-results` element that allows each step to specify an expected result in order to verify the successful execution. This allows the

[2]http://json-ld.org/

verification to be as flexible as possible, covering traditional integration tests, custom scripts, or service invocations. Additionally, *TUs* provide a `configuration` element with an extensible key-value format that allows to provide additional configuration information. Furthermore, a *TU* contains a `dependency` section to specify dependent artifacts (such as other application components or required data stores), as well as a `metainformation` section to describe additional relevant aspects like used `framework`, required `runtime`, as well as basic system requirements, such as minimum amount of `memory` or desired `cpu` capacity. Additionally, *TUs* as well as all other elements of the system model allow the use of variables that are evaluated by the Smart Fabric framework. These variables are designated with the '`@`' prefix.

Listing 4.1: Technical Unit - Structure

```
{
  "@context": "http://smartfabric.dsg.tuwien.ac.at",
  "@type": "TechnicalUnit",
  "name": "CitizenInformationSystem",
  "artifact-uri": "...",
  "language": "ruby",
  "build": {
    "assembly": "/citizeninformationsystem",
    "steps":[{"step":1,"tool":"bundler","cmd":"bundle install"},{"step":2,"
        tool":"rake","cmd":"rake db:migrate"}, {"step":3,"tool":"rake","cmd":
        "rake db:seed"}]
  },
  "execute":[{"step":1,"tool":"rails",
    "cmd":"rails s"}],
  "verification": {
    "steps":[{"step":1,"tool":"curl","cmd":
      "curl -i @destination_url/status",
      "expected-result":"HTTP 200 OK"}]
  }
  "metainformation": {
    "type": "standalone",
    "framework": "Rails 4.0",
    "runtime":"ruby 2.0, rails 4.0"
  },
  "dependencies": {
    "datastore": {
      "type":"relational",
      "interface": "sql"
    }
  }
}
```

4.2.2 *Infrastructure Specification*

ISs are used to describe the capabilities of different infrastructure resources. This ranges from legacy systems that are running on premises on cloud infrastructures including Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) providers. The *IS* contains a `name` and `version` identifying specific infrastructures with the support for different versions of infras-

tructure stacks as well as a `kind` to discriminate between infrastructure concepts like bare metal systems, IaaS, and PaaS. This is followed by a `server` entry, which is used to describe the computing capabilities of an infrastructure. For instance, in the case of a classical server system this section contains the bare metal specifications of the machine itself, in the case of Amazon the available compute instances and their respective properties. Processing capabilities are specified using the `compute_units` array, where each entry specifies a `name` to identify the entity as well as a `capacity` element. Similarly, other available services, such as `storage`, `network`, or `databases` can be specified with each of them including `name`, `type`, `version`, `interface`, as well as the previously introduced step notation to describe how to use or access them. Listing 4.2 shows an example IS for an excerpt of the Amazon AWS stack.

Listing 4.2: Infrastructure Specification - Structure

```
{
  "@context": "http://smartfabric.dsg.tuwien.ac.at",
  "@type": "InfrastructureSpecification",
  "name": "Amazon AWS",
  "version": "1.0",
  "kind":"cloudprovider",
  "server": {
      "metainformation":{
        "cpu":"Intel Xeon E5"
      },
      "compute_units":[{"name":"t2.micro","capacity":{"cpu":"1","memory":"1
          GB","storage":"@EBS"}, ..., {"name":"m3.2xlarge","capacity":{"cpu":
          "8","memory":"15 GB","storage":"80 GB"}}]
  },
  "storage": {...},
  "network": {...},
  "databases":[{"name": "RDS db.m3.medium","type": "relational","interface":
      "sql/mysql"},...],
  "services":[...]
}
```

4.2.3 Deployment Unit

DUs provide a mean to describe how to deploy a *TU* on a specific *IS*. They link one or more *TUs* to exactly one specific *IS* by referencing their names. Additionally they allow to define `constraints` that need to be met in terms of hardware and software. Hardware constraints, for example, cover minimal machine requirements in terms of CPU and Memory, or, in the case of IaaS or PaaS providers, the minimal provided compute units. Software constraints allow to specify a priori requirements on the *IS* in order to be able to actually deploy the *TU* on it. This includes programming languages, as well as runtime environments or framework requirements. The last element of the *DU* is the previously introduced flexible step definition that outlines the steps that are necessary to deploy the set of *TUs* to the specified *IS*.

Listing 4.4 shows an example of such a *DU* for the deployment of the *TU* Citizen Information System on the *IS* `dedicatedserver`.

Listing 4.3: Deployment Unit - Structure

```
{
  "@context": "http://smartfabric.dsg.tuwien.ac.at",
  "@type": "DeploymentUnit",
  "name": "CitizenInformationSystem/DedicatedServer",

  "technicalUnits":[{
    "name":"citizeninformationsystem",
    "id":"citizeninformationsystem.tu.json"}],
  "infrastructureSpecification":{
    "name":"dedicatedServer"
  },

  "constraints": [{"hardware":[{"memory": "4GB"}]}],
  "steps": [{"step": 1,"tool": "git","cmd": "git clone @destination/
      @citizeninformationsystem.artifcat-uri"},
             {"step": 2,"tool": "bash","cmd": "cd @destination"},
             {"step": 3,"cmd": "@citizeninformationsystem.@build"},{"step": 4,
                "cmd":"@citizeninformationsystem.@execute"}
            ]
}
```

4.2.4 *Deployment Instance*

A *DI* represents a specific deployment of a *TU* to an *IS* based on a *DU* taking into account all defined hardware and software constraints. There can be multiple *DIs* for a *DU* representing different specific deployments for example covering development, staging, or production deployments of an application on specific infrastructure. A *DI* specifies `context`, `type`, and `name`, as well as a `deploymentUnit` to reference the corresponding `DU`. This is followed by a `machine` element that stores data about the specific machine or machines that are currently used to execute the deployment. The `application` element that contains runtime information about the *TU* stores `name` and `version` of the application component, as well as an `environment` element that contains relevant environment information resolved by the framework. Finally, the `global` element allows to store additional information about the specific deployment in an open key value format that can later be used by the framework components to support deployment decisions.

Listing 4.4: Deployment Instance - Structure

```
{
  "@context": "http://smartfabric.dsg.tuwien.ac.at",
  "@type": "DeploymentInstance",
  "name": "....",
  "deploymentUnit":"CitizenInformationSystem/DedicatedServer",
  "machine": {"id":, ....},
  "application":{"name":"...","environment":[{"key":"..."}]},
  "global":{...}
}
```

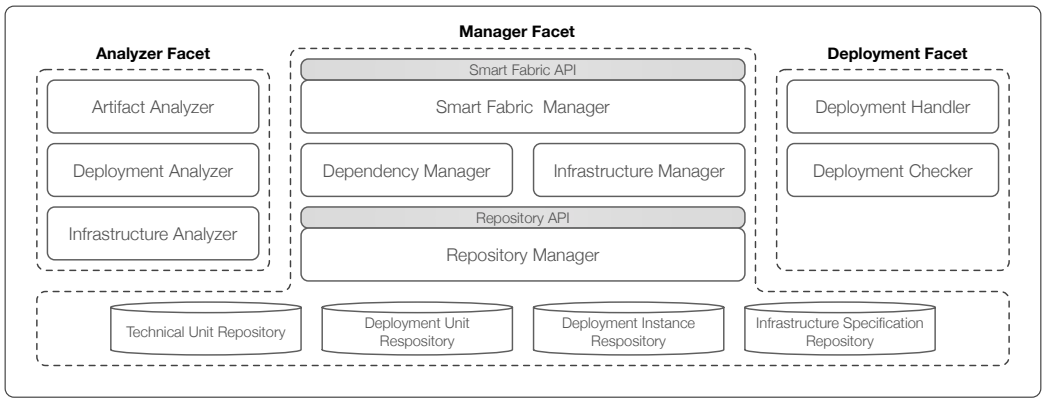


Figure 4.2: Smart Fabric Framework Overview

4.3 The Smart Fabric Framework

In this section we introduce the Smart Fabric framework for infrastructure-agnostic artifact topology deployment that implements the system model presented above. We start with a framework overview, followed by fundamental framework rationales, and conclude with a detailed description of all framework elements.

4.3.1 Framework Rationales

The Smart Fabric framework follows the microservice [78] architecture paradigm. An overview of the main components is shown in Figure 4.2. The framework is logically organized into three main facets which group areas of responsibility. Each of these facets is composed of multiple components where each of these components is a microservice. The components in the *Analyzer* and *Handler Facet* are managed as self-assembling components[3] following a functional approach based on the Command Pattern [33]. In this approach each component consists of multiple *processors*, where each *processor* is able to accept multiple inputs and produces exactly one output, resembling a classic functional approach, as illustrated in Figure 4.3. This allows for a clean separation of concerns into distinct functional steps that can be as specific as necessary in order to decompose complex problems into manageable units. Each of these *processors* in turn announces which inputs it requires, as well as which output it produces. This enables a straightforward auto assembly approach, where connecting previous outputs to desired inputs leads to an automatically assembled complex system consisting of simple manageable *processors*. It also eliminates the necessity of complex composition and organization mechanisms enabling dynamic and elastic composition of desired functionality, where processors can be added on demand and at runtime.

The second foundational framework rationale is that the components follow the principle of *Confidence Elasticity*, which means that each component follows a confidence-based adaptation

[3] http://techblog.netflix.com/2014/06/building-netflix-playback-with-self.html

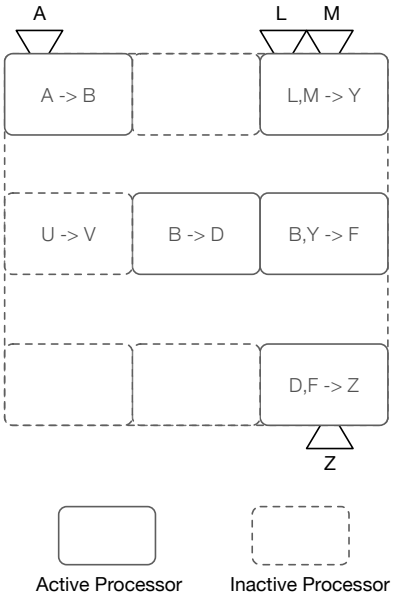


Figure 4.3: Example of auto assembling processors within a component. A, L and M are initial inputs for the component, Z the final output. Each processor is active and produces an output if the expected inputs are available (e.g. L, M produces Y)

model. If a component or processor produces a result, it augments this result with confidence value ($c \in \mathbb{R}, 0 \leq c \leq 1$), with 0 representing no certainty and 1 representing absolute certainty about the produced result. This convention allows the framework to configure certain confidence intervals to augment the auto assembly mechanism. These confidence intervals are provided as configuration elements for the framework. If these confidence thresholds are not met, the framework follows an escalation model to find the next component or processor that is able to provide results with higher confidence until it reaches the point where human interaction is necessary to produce a satisfactory result. Each processor p_i from the set of active processors P_a provides a confidence value c_i. We define the overall confidence value of all active processors c_a as $c_a = \prod_{p_i \in P_a} c_i$. The compensation stops when c_a meets the specified confidence interval of the framework or a processor represents a human interaction which has a confidence value of ($c_i = 1$). This mechanism is outlined in Figure 4.4 illustrating the process by trying to determine if an artifact is a Ruby on Rails application or not. If it is not possible to determine this within the specified confidence interval by utilizing fully automated configuration analyses, the framework escalates until a result is produced, ultimately consulting human experts to determine the nature of the artifact.

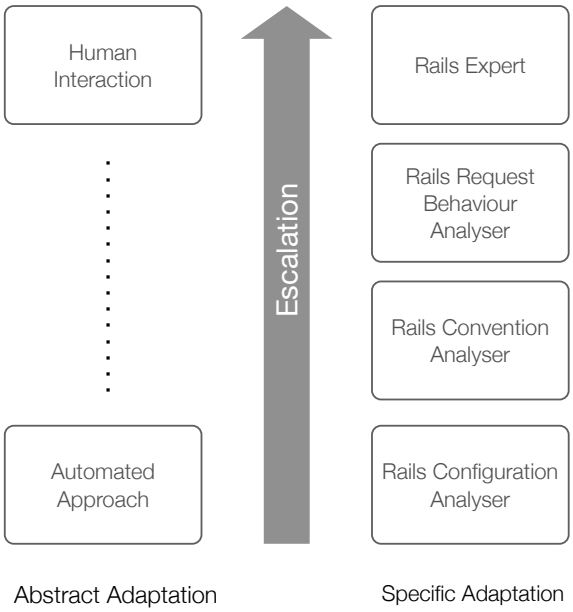


Figure 4.4: Confidence Adaptation Model Escalation

4.3.2 Smart Fabric Manager

In order to initiate a concrete deployment of an artifact on an infrastructure a user invokes the *Smart Fabric API* with the following parameters: (i) names of *TUs* to be deployed, (ii) name of *IS* they should be deployed to, and (iii) any additional artifacts necessary that cannot be accessed by the framework itself (e.g., private repositories or external executables). The *Smart Fabric Manager* first tries to find the specified *TU* and *IS* by querying the *Repository Manager*. If both are found it hands the *TUs* and *IS* over to the *Dependency Manager*, which in turn resolves all dependencies between *TUs* and provides the corresponding *DUs*. All elements of the system model are then forwarded to the *Infrastructure Manager* that verifies if all constraints can be satisfied in order to deploy the *TUs* on the *IS* according to a *DU* via a *DI*. If this is the case the *Infrastructure Manager* produces an augmented *DI*. The *DU* and *DI* are then forwarded to the *Deployment Handler*, which actually deploys the *TU* on the *IS* using the contained deployment directives. After successful deployment it updates the *DI* and returns it to the *Smart Fabric Manager* that in turn persists it using the *Repository Manager*.

If in this process *TUs*, *DUs* or *ISs* cannot be found the *Smart Fabric Manager* utilizes the components in the *Analyzer Facet* to derive or generate them.

4.3.3 Artifact Analyzer

The task of the *Artifact Analyzer* is to generate a *TU* based on a provided artifact. It is invoked by the *Smart Fabric Manager* if no *TU* can be found for a given artifact. To accomplish this

it relies on an open and extendable set of processors which follow the previously discussed framework rationales to analyze provided artifacts. Possible processors in this component are: (i) Configuration Processors that try to select a *TU* based on certain configuration files of the artifacts, (ii) Similarity Processors that try to select a *TU* based on similarities in the *TUs* values by performing actions like collaborative filtering, (iii) Convention Processors that try to derive a *TU* based on certain conventions an artifact follows (e.g., folder or file naming conventions), (iv) Behavior Processors that try to select a *TU* based on the behavior of an artifact (e.g., deriving a *TU* by sending specific requests and analyzing the responses), and (v) Human Provided Processors, which are human experts that manually create a *TU*.

4.3.4 Deployment Analyzer

The *Deployment Analyzer* generates a *DU* based on provided *TU* and *IS* if no suitable *DU* could be found by the *Smart Fabric Manager*. This component, like the *Artifact Analyzer* and the *Infrastructure Analyzer*, relies on an open and extensible set of processors. Possible processors are: (i) Similarity Processors that try to select a *DU* based on similar *DUs* that have been used for similar *TU* and *IS* configurations, (ii) Convention Processors that try to derive a *TU* based on certain conventions an infrastructure follows (e.g., to deploy to a PaaS like Heroku *DUs* for specific *TUs* follow the same conventions), and (iii) Human Provided Processors, which are human experts that manually create a *DU*.

4.3.5 Infrastructure Analyzer

The role of the *Infrastructure Analyzer* is to generate a *IS* if no suitable specification can be found in the repository. Possible processors are: (i) Documentation Processors that try to generate an *IS* based on the accessible documentation of the infrastructure. This can range from querying machine readable linked data (e.g., service registries) about the infrastructure, to analysis of electronic documentation using machine learning techniques, (ii) Behavior Processors that try to derive an *IS* based on externally observable aspects of an infrastructure (e.g., log analytics, system statistics etc.), and (iii) Human Provided Processors, which are human experts that manually create an *IS*.

4.3.6 Dependency Manager

The *Dependency Manager* is responsible for resolving unit dependencies between the modeled entities, as described in the system model above. To achieve this the dependencies between entities in the system model are represented as a tree structure. Based on this tree structure the *Dependency Manager* creates a root node for each *TU*. It then creates a corresponding leaf node for each *TU* that is referenced in the dependency section of the *TU*. After this it checks the related *DUs* and adds them as leaves. The final step is to add the referenced *DIs* to the respective *DUs* as a leaf nodes. The final result is a complete tree structure that is sufficient for the *Deployment Handler* to effectively deploy a *TU* to an *IS*.

4.3.7 Infrastructure Manager

The role of the *Infrastructure Manager* is to handle all concerns regarding the infrastructure, where a specific deployment represented as a *DI* takes place. Its main responsibility is to ensure that all necessary resources described by the *IS* are accessible and available to successfully deploy the *TU*. This also includes all issues of authentication and authorization by ensuring the relevant credentials are provided. Additionally, it ensures that all constraints defined in *TUs* and *DUs* are satisfied. The system model distinguishes between hardware constraints and software constraints. Hardware constraints cover all constraints related to machine-specific aspects, including processors, memory, and disk space, as well as specific machine types in the context of IaaS and PaaS providers. Software constraints cover programming languages, runtime environments, or framework related aspects that need to be satisfied in order to ensure that an artifact can be successfully deployed on a specific *IS*. The *Infrastructure Manager* augments the *DI* accordingly and when finished hands it over to the *Deployment Handler*.

4.3.8 Repository Manager

The *Repository Manager* provides repositories for all units described in the system model and acts as a distributed registry keeping track of deployments and participating entities. It is responsible for system model storage and retrieval. It manages four distinct system model repositories utilizing distributed key-value stores, which store the JSON-LD files that represent *TU, DU, IS and DIs* in a structured way. The *Repository Manager* provides a service interface to store and retrieve these files as well as a search interface to query *TUs*, *DUs*, *ISs*, and *DIs* based on specific elements.

4.3.9 Deployment Handler

The *Deployment Handler* is responsible for effectively executing a deployment. It follows the same processor-based principle as the previously introduced *Analyzers*. The *Deployment Handler* receives the resolved dependency structure as a tree model from the *Dependency Manager*. Each processor in the component is responsible for executing the deployment for specific infrastructure types. Processors can utilize different deployment mechanisms, including (i) Script Processors that utilize simple script-oriented approaches to execute the deployment (e.g., a processor using Bash scripts), (ii) Tool Processors that use more sophisticated tool-based approaches (e.g., Chef[4], LEONORE [108], or Capistrano[5]), and (iii) Human Interaction Processors to allow for deployments that need human interaction (e.g., command line input or user interface based approaches).

In order to ensure all credentials are available and all constraints are met the *Deployment Handler* interacts with the *Infrastructure Manager*. After the *Deployment Handler* has executed the deployment it invokes the *Deployment Checker* to verify that the deployment was successful. In case of a successful deployment the *Deployment Handler* augments the *DI* with all values that have been gathered during the deployment, such as specific service endpoints and IP addresses.

[4] https://www.chef.io/chef/

[5] http://capistranorb.com/

In case of an unsuccessful deployment the *Deployment Handler* escalates to the *Infrastructure Manager*.

4.3.10 Deployment Checker

The *Deployment Checker* is responsible for verifying if a completed deployment was successful or not. To verify this the *Deployment Checker* retrieves the *TU* and *DI* from the *Deployment Handler* and executes the steps in the `verification` section of the *TU*. As mentioned above, verification steps default to following UNIX conventions, considering commands that exit with a result code of 0 to be successful, whereas other result codes are interpreted as errors. Additionally, *Deployment Checker* implementations can choose to use the `expected-result` element of the given *TU* to perform custom validation steps, e.g., based on pattern matching.

4.4 Validation

4.4.1 Implementation

We implemented a preliminary prototype of the Smart Fabric framework in Ruby. To establish the frameworks microservice architecture we rely on REST as message exchange and interface technology. To serve the RESTful interfaces we use Sinatra[6] as web server for each of the implemented components.

The *Repository Manager* utilizes Redis[7] as the key-value store for the repositories in order to provide fast and efficient storage for all elements of the system model. To enable the auto assembly mechanism for each processor within the framework we use RabbitMQ[8] as a message exchange middleware. Each implemented processor of every auto assembled component publishes its output and listens for its desired inputs on dedicated topics. This gives us the opportunity to receive messages based on patterns and allows for finer grained control over processor input values.

To implement the *Dependency Manager* we rely on the RubyTree library[9] to create the described tree structure. To enable system model entity augmentation and generation for the *Analyzer Processors* as well as for all *Deployment Handlers* we use a simple generator mechanism. This mechanism relies on predefined templates that are augmented with ruby code in order to easily build and modify system model entities. To achieve this we rely on the eRuby templating system provided by the Ruby standard library.

4.4.2 Validation Scenario

For validation purposes we use the Smart Fabric framework to deploy and redeploy different sample applications. We demonstrate this based on two typical Smart City and IoT applications. The first one is an exemplary Ruby on Rails based Web Application with a RESTful service

[6] http://www.sinatrarb.com/
[7] http://redis.io/
[8] https://www.rabbitmq.com/
[9] https://github.com/evolve75/RubyTree

interface that represents a *Citizen Information System (CIS)*. The *CIS* is a common application type in the Smart City domain, which provides open data access for various city aspects such as traffic information and municipal energy usage. We chose this application type because on the one hand it is not fragmented into several artifacts and on the other hand because this type of application undergoes a natural infrastructure evolution due to increasing data and demand.

The second application is a sample IoT application based on a case study conducted in our lab in cooperation with a business partner and represents a *Building Management System (BMS)*. The *BMS* is a Java application based on the Spring Boot framework and is built as a microservice architecture with multiple artifact dependencies. We choose the *BMS* since it provides a good contrast to the *CIS* due to its distributed nature with high artifact fragmentation, as well as its inherently large scale. For both applications we created appropriate *TUs*.

We then select four different infrastructures to deploy them to. These infrastructure types are a *Dedicated Server* hosted at our group, *Amazon AWS*[10] as an IaaS, *Heroku*[11] as a PaaS and an *OpenStack*[12] based private cloud hosted at our group. We chose this kind of infrastructure fragmentation because it represents a good coverage of different heterogenous infrastructures that are used to host todays application landscape. They also reflect the trend towards cloud based platforms and provide a good baseline for application evolution challenges [96]. The selected infrastructure types offer different services and capabilities, which leads to several challenges in redeploying them. The *Dedicated Server*, for example, imposes no restriction on the choice of databases (since deployed application packages must be managed by operators), whereas *Heroku* or *Amazon AWS* offer a restricted set of database services that are managed by the provider. This is the case for multiple aspects of these infrastructures and makes them an optimal testing scenario to show the effects of infrastructure evolution and the challenges this brings to infrastructure-agnostic application deployment.

For each of these infrastructures we create corresponding *ISs* as well as *DUs*. We then also provide exemplary *DIs* representing different specific deployments of our two *TUs* to the *ISs* based on the defined *DUs*. After this we test the following scenario. We initialize the Smart Fabric framework and load all previously defined system model entities. In the first step of our test we deploy the *CIS* and the *BMS* to the *Dedicated Server* by initiating two deployment requests to the Smart Fabric framework with the according *TUs* and *IS*. After the successful deployment notification by the framework we test the deployments with the information in the augmented *DIs*. In the case of the *CIS* we query the REST interface of the Traffic Resource and log the results. In the case of the *BMS* we query the REST interface of the Controller and log the results.

We then initiate two service requests to the Smart Fabric framework to transfer the *CIS* as well as the *BMS* to Heroku. We wait for the successful framework deployment notification and perform the previously described test again with the augmented information in the *DIs* and log the results accordingly.

In the final step we execute two additional transfer requests to the Smart Fabric framework, deploying the *CIS* to AWS and the *BMS* to our OpenStack cloud. After the successful framework deployment notification, we again test both application as previously described and log the results.

[10] http://aws.amazon.com/

[11] http://www.heroku.com

[12] https://www.openstack.org/

We then compare the logged results with each other and ensure that deployment was successful on all infrastructures and produced the expected results.

In both sample applications we showed that Smart Fabric was able to successfully redeploy artifacts on heterogenous infrastructures without modifying application artifacts. For instance, Smart Fabric deployed the CIS on Amazon AWS with RDS as database backend whereas it deployed on Heroku using Heroku Postgres.

4.5 Related Work

The migration of application topologies between different deployment targets has been studied at various levels in the literature. With the adoption of the cloud paradigm, for instance, the problem of developing applications for and migrating existing applications to the cloud emerged [51]. To address this problems, Leymann et al. [58] present a meta model and tool that supports splitting an existing application in several parts, such that these parts can be moved to the cloud. In order to develop cloud infrastructure agnostic applications, Ardagna et al. [8] propose MODAClouds that provides a model-driven solution. Kwon et al. [54] present refactoring techniques and automated program transformations that help transitioning an application to use cloud-based services. In contrast to our work, these approaches solely focus on the internal design and execution of singular components of cloud-based applications, rather than providing a solution for overall architecture of a such applications.

Next to the migration of applications to, and designing applications for the cloud, another important area is the migration of an existing cloud application to a different cloud offering [39, 80, 122]. Binz et al. [13] present the Topology and Orchestration Specification for Cloud Applications (TOSCA), which aims for portable and standardized management of cloud services. TOSCA provides means for describing portable application deployment topologies consisting of nodes and their relationships. By specifying possible plans, TOSCA allows for governing the complex workflow of provisioning and deploying an application on cloud infrastructure. Based on TOSCA, Andrikopoulos et al. [6] propose the Generalized Topology Language (GENTL), an application topology modeling language, which provides the foundation for possible optimizations of the distributed deployment of the application.

Additionally, recent work was done to address the problem of vendor lock-in in the cloud [99, 100]. Satzger et al. [86], for instance, propose the meta cloud concept that proposes both design and run-time components to mitigate vendor lock-in by abstracting away technical incompatibilities from existing offerings. This approach helps in finding the right amount of cloud services that are needed for a specific use case, to support the initial deployment and runtime migration of an application. Sellami et al. [95] present a unified description model that represents an application and its requirements independently of the targeted PaaS. In addition, a generic application provisioning and management API is proposed to abstract away PaaS-dependent functionality. These two concepts combined, represent a PaaS-independent solution for the provisioning and management of applications in the cloud to avoid vendor lock-in. The aforementioned approaches have in common that they focus on deployment topologies of cloud applications, and therefore are complementary to the methodology presented in this chapter. However, they do not provide a constraint-based, declarative specification of the required deployment infrastructure in order to

allow for an infrastructure-agnostic deployment. Furthermore, all approaches have in common that they do not provide a mechanism and toolset that also deals with the seamless migration between deployment targets, i.e., across infrastructure boundaries.

4.6 Summary

The emergence of the cloud computing paradigm enables applications to autonomously provision and also release infrastructure resources in order to elastically respond to environmental changes (e.g., increased request load). To fully leverage the potential of the cloud however, applications need to be specifically designed and developed for a concrete cloud provider infrastructure, which leads to a strong dependence on specific offerings provided by the cloud provider. As a result, moving applications to the cloud or migrating an application between cloud providers is a tedious task due to variations of provided services, interfaces and deployment tools among providers. To address these problems, in this chapter we presented Smart Fabric, a methodology and toolset to enable infrastructure-agnostic deployment of artifact topologies based on a declarative, constraint-based specification of the required deployment infrastructure. Current approaches do not sufficiently consider the specific, practical problems of dealing with evolving deployment infrastructure and closely tie application artifacts to their deployment targets. By extending the MADCAT methodology with a dedicated abstraction layer to clearly separate deployment infrastructure and application deployment, Smart Fabric allows for seamless, independent evolution of both, application components, as well as the underlying infrastructure. Moreover, our approach enables transparent application deployment and evolution between deployment targets, i.e., across traditional infrastructure boundaries (e.g., migrating applications between on-premise and PaaS offerings, or between PaaS and IaaS), without changes to application code. Smart Fabric implements a confidence-based decision model that aims to automate application deployment when possible, and will escalate to human operators when necessary. We discussed the feasibility of the introduced methodology and developed a prototype by using representative applications from the Smart City and IoT domains.

CHAPTER 5

Modeling and Management of Usage-Aware Distributed Datasets

In this chapter, we present a methodology and toolset to model smart city data sources and enable efficient, distributed data access in smart city environments. We introduce a modeling abstraction to describe the structure and relevant properties, such as security and compliance constraints, of smart city data sources along with independently accessible subsets in a technology-agnostic way. Based on this abstraction, we present a middleware toolset for efficient and seamless data access through autonomous relocation of relevant subsets of available data sources to improve Quality of Service for smart city applications based on a configurable mechanism. We evaluate our approach using a case study in the context of a distributed city infrastructure decision support system and show that selective relocation of data subsets can significantly reduce application response times.

5.1 Introduction

Sparked by the rapid adoption of the smart city paradigm and fueled by the rise of the Internet of Things, todays metropolises have become data behemoths. With every day that passes more and more areas of cities around the globe start accumulating and producing data. These areas cover building management, traffic and mobility systems, energy grids, water and pollution management, governance, social media, and many more. This plethora of heterogenous data about various aspects of a city represents a vital foundation for decision and planning processes in smart cities. The advent of more and more open data initiatives around the globe, covering cities like London[1], Vienna[2], New York[3], and many more underlines the importance of opening up data to the public to inspire and support novel applications. Even though these initiatives are

[1] https://data.london.gov.uk/

[2] https://open.wien.gv.at/site/open-data/

[3] https://nycopendata.socrata.com/

gaining momentum, they still only cover a fraction of the available data of a city, missing many vital sources, especially when it comes to more sensitive areas like building management, energy grids, or public transport guidance systems. Currently, this data is mostly isolated and restricted to certain application areas, data centers, organizations, or only accessible in a specific city. This isolation creates data silos, which lead to transitive restrictions that apply to the models and applications that build upon them, confining them to their initial application domains. Todays smart cities however, represent heterogeneous, dynamic, and complex environments that rely on emerging interactions in order to operate effectively. These interactions are an essential element of Smart City Applications [91] and not only important in an intracity context, but also a key element to enable the future Internet of Cities [89], an interconnected system of systems that spans multiple cities around the globe. To pave the way for such applications we need to break up the traditional notion of data silos to enable ubiquitous access to the valuable data they contain. In the context of smart cities, an approach is required that respects the complexities of this domain, specifically the need to effectively describe a large variety of heterogenous data sources along with relevant subsets. Additionally, it has to be able to capture important data set characteristics (e.g., size, update frequency, costs), respect essential security and compliance constraints, as well as ensure efficient and seamless data access.

In this chapter, we present Smart Distributed Datasets (SDD), a methodology and framework to enable transparent and efficient distributed data access for data sources in smart city environments. We introduce a system model that provides a simple abstraction for the technology-agnostic description of data sources and their subsets with the ability to express varying data granularities and specific characteristics common in the smart city domain. Based on this abstraction, we present the SDD framework, a middleware toolset that enables efficient and seamless data access for smart city applications by autonomously relocating relevant subsets of available data sources to improve Quality of Service (QoS) based on a configurable mechanism that considers request latency, as well as costs for data transfer, storage, and updates. We provide a proof of concept implementation of the SDD framework and evaluate it using a case study in the context of a the URBEM scenario (Section 3.1). For this case, we show that selective relocation of data subsets using the SDD framework can significantly improve QoS by reducing response times by 66% on average.

The remainder of this chapter is structured as follows. In Section 5.2 we identify the associated key requirements in the context of our URBEM scenario. We introduce the system model underlying SDD in Section 5.3 and present the SDD framework along with a detailed discussion of its components in Section 5.4. In Section 5.5 we evaluate our approach using a case study from the smart city domain. Related work is discussed in Section 5.6, followed by a conclusion in Section 5.7.

5.2 Motivation

In this chapter, we base our discussion on the URBEM scenario, specifically we identify the requirements based on the URBEM Smart City Application (USCA) presented in Chapter 9, a holistic, interdisciplinary decision support system for the city of Vienna and a number of key stakeholders. We argue that such applications will evolve to become composable, interchangeable

abstractions of capabilities similar to the applications known from todays smart phones, but on a much larger scale. This evolution in turn is an essential step towards the so-called Internet of Cities [89], an open and dynamic market place where applications can seamlessly interact and be exchanged between cities around the globe.

To enable these applications as well as this vital open exchange it is essential to provide means to expose and access data in an efficient, secure, and predictable way. Currently, most of the data in a smart city context is confined to certain application areas and stakeholder data centers within a specific city. Open data initiatives around the globe, while crucial, still only expose a certain fraction of the available data, missing out on many important domains, especially when data is stored in legacy systems without openly accessible interfaces or underlies strict security and compliance constraints. This data lies dormant beyond its initial use case even though it could provide essential input for a wide range of smart city applications. The ability to benefit from incorporating new data sources as they evolve, for example to enhance decision support and planning, or to be applied to new cities or novel domains, is hindered by the inability of these applications to access the necessary data. Developers of smart city applications, however, need to be able to utilize and integrate as much relevant data as possible to generate maximum user benefit as well as applicability in as many cities as possible. Stakeholders on the other hand, as willing as they might be to expose this data, are mostly bound by the complex constraints of their specific environment. The dynamic, emergent nature of interactions in and between smart city applications means they are not a priori aware that their data sources might become valuable assets if made accessible. This leads to a problematic stalemate between practitioners and stakeholders in the smart city domain, hindering essential innovation and application.

To overcome this impasse, a mechanism is required that enables flexible, stable, and efficient data access, while providing a simple and tailored way to make data sources available, which still respects security and compliance constraints. Specifically, we identify the following requirements in the context of our domain:

- The ability to describe data sources using an evolvable and technology-agnostic abstraction.
- The ability to describe subsets of these data sources along with relevant characteristics in the context of security, compliance and costs (e.g., effort to generate, store, query, and update particular subsets or the underlying data source as a whole).
- An efficient way to access this data in a transparent way, independent of geographic location while still improving QoS.

5.3 System Model

In order to address the previously outlined requirements, we need an abstraction to model and describe the relevant data entities in our domain. As foundation for said abstraction we use MADCAT [46] and it's extensions, which we introduced in Chapter 4. We presented an infrastructure agnostic deployment model with the following abstract concepts: *Technical Units (TUs)* to describe applications as well as application components, *Infrastructure Specifications (IS)* describe

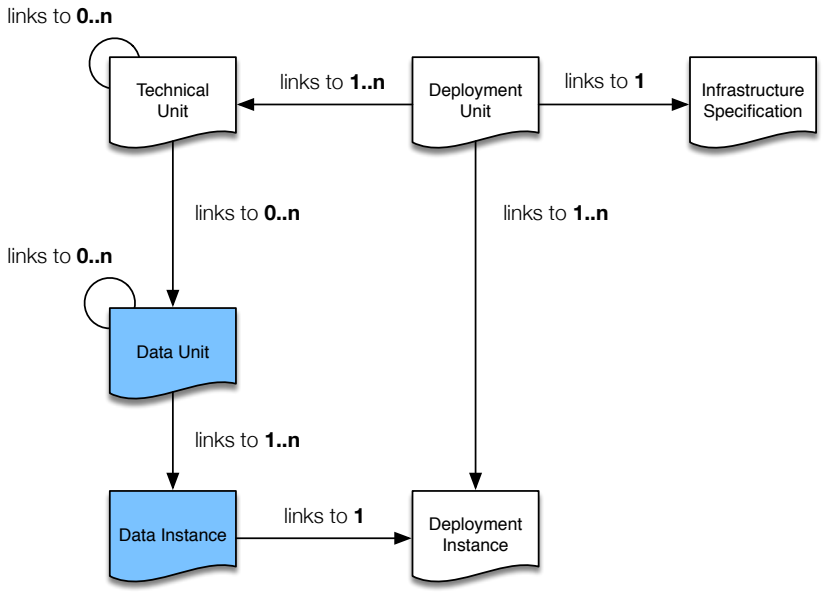


Figure 5.1: Relations between Technical Unit, Deployment Unit, Infrastructure Specification and Deployment Instance including the newly introduced Data Unit and Data Instance

infrastructure resources, *Deployment Units (DUs)* to describe how to deploy and *TU* on an *IS* and an *Deployment Instance (DI)* represented such an actual deployment. In this chapter, we extend this model with the ability to describe and incorporate data entities from the smart city domain. Specifically we introduce the additional concepts of *Data Units (DAUs)* to model data sources as well as *Data Instances (DAIs)* to describe specific deployments of *DAUs* on certain *DIs*, along with the ability to link *TU* to *DAUs*. Additionally, we provide an implementation of the abstract concepts along with the proposed methodology extensions for the SDD framework. We again choose JSON-LD[4] as data format for our concept description as a simple, both human and machine readable, pragmatic, and extensible representation that also allows us to interlink the relevant concepts with each other. Figure 1 shows an overview of all concepts, including the relations of the newly introduced *DAU* and *DAI* (shown in blue) to the previously existing concepts *Deployment Instance (DI)* and *Technical Unit (TU)*. In the following, we discuss the introduced concepts in more detail.

5.3.1 Data Unit

DAUs describe data sources including its subsets. Listing 5.1 shows an example of such a *DAU* for a buildings data source in the URBEM domain. As JSON-LD document, a *DAU* can start with a `context` to set up common namespaces. This is followed by the `type` attribute to identify the corresponding kind of abstraction in our model space. The next attribute is the `name` as *URN*

[4]http://json-ld.org/

to identify the unit, along with a `version` attribute enabling versioning and therefore evolvable *DAUs*. The `creationDate` and the `lastUpdate` define when the unit description was initially created respectively last updated. The next section is `metainformation` about the described data source. It contains a `type` attribute that defines the type of the data source, types can be REST, SOAP, any database, streaming data or file based. In our example listing it is used to describe a document oriented MongoDB[5] based data source. The `schema` attribute can link to a corresponding schema, which based on the `type` and can be anything of the likes of a SQL schema, JSON schema, WADL[6] or WSDL[7]. The next attribute is `securityConstraints` and in the context of this section it is used to define who is allowed to access the unit file. A security constraint can be a link to an OAuth[8] authority, LDAP distinguished name (DN) or any other corresponding authentication and authorization scheme. This is a vital element to ensure the compliance and security constraints of this domain can be met on any level of detail and is again used when describing specific facets of a data source. The last element in the metainformation section is the `dataUnits` attribute, which allows to link a *DAU* to other corresponding *DAUs* enabling the description of linked data sources. The next section `views` allows to express multiple facets of the data source enabling a fined grained level of control about its aspects. Each view has a `name` attribute for identification as well a a `link` to express how to access it. In case of the our example this is a *URL* to the corresponding rest resource. The next section within a view is the *updateFrequency*. It allows to express how often a view is being updated (`period,times`), how long such an update takes (`updateTime`) as well as how much of the resource this update represents (`fraction`). The `size` attribute gives an indication of the expected size of the view. This is followed by a `securityConstraints` attribute that again can be a set of links to a corresponding authentication or authorization scheme supporting the previously mentioned methods. In this section it is used to express security and compliance constraints for specific fractions of a data source, which allows for a fine very grained level of control. The last attribute in a view is the *dataInstances* attribute, which links the view and its corresponding *DAU* to one or more *Data Instance (DAI)* by referencing their names.

Listing 5.1: Data Unit - Structure

```
{ "@context": "http://smartfabric.dsg.tuwien.ac.at",
  "@type": "DataUnit",
  "name": "urn:buildings:vienna",
  "version": "1.0"
  "creationDate": " 2017-01-07 13:04:03 +0100 ",
  "lastUpdate": " 2017-01-07 13:04:03 +0100"

  "metainformation": {
    "type":"nosql:mongodb",
    "schema": "http://sdd.dsg.tuwien.ac.at/urbem/vienna/buildings",
    "securityConstraints":[{
      "type":"ldap"
      "url":"10.2.0.112"
```

[5] https://www.mongodb.com/
[6] https://www.w3.org/Submission/wadl/
[7] https://www.w3.org/TR/wsdl
[8] https://oauth.net/

```
      "dn":"CN=buildings-vienna",
      ...
      }],
    "dataUnits":[...]
  }

  "views":[{
    "name":"urn:buildings:vienna:buildingblocks",
    "link":"/buildingblocks/",
    "updateFrequency":{
      "period":"yearly",
      "times": "1",
      "fraction": "10"
      "updateTime" : "2300"
      },
    "size":"10000303",
    "securityConstraints":[...]
    "dataInstances":[...]

    },
    "name":"urn:buildings:vienna:buildings",
    "link":"/buildings/",
    ...]
}
```

5.3.2 *Data Instance*

A *DAI* represents a specific deployment of a *DAU* on a *DI*. There can be multiple *DAIs* for a *DAU* representing different deployed views, where a specific *DAI* contains a subset of the views specified in the *DAU*, i.e., $DAI \in \mathscr{P}(DAU)$. A *DAI* specifies `context`, `type`, `name` and `version`, as well as a `dataUnit` to reference the corresponding *DAU*. This is followed by `creationDate` and `updateDate` to define when the instance was created as well as last updated. The next attribute is `deploymentInstance`, which contains a *DI* name and is used to link the *DAI* to a corresponding *DI*. Finally, the `metainformation` element allows to store additional information about the specific data instance in an open key value format that can later be used by the framework components to support transfer decisions, examples would be `accessFrequency` of this specific *DAI* or other non functional characteristics.

Listing 5.2: Data Instance - Structure

```
{ "@context": "http://smartfabric.dsg.tuwien.ac.at",
  "@type": "DataInstance",
  "dataUnit":"urn:buildings:vienna"
  "name": "urn:buildings:vienna:buildingblocks",
  "version":"1.0",
  "creationDate":"2017-01-07 13:04:03 +0100"
  "lastUpdate":"2017-01-07 13:04:03 +0100"
  "deploymentInstance":"CitizenInformationSystem/DedicatedServer"
  "metainformation":{...}
}
```

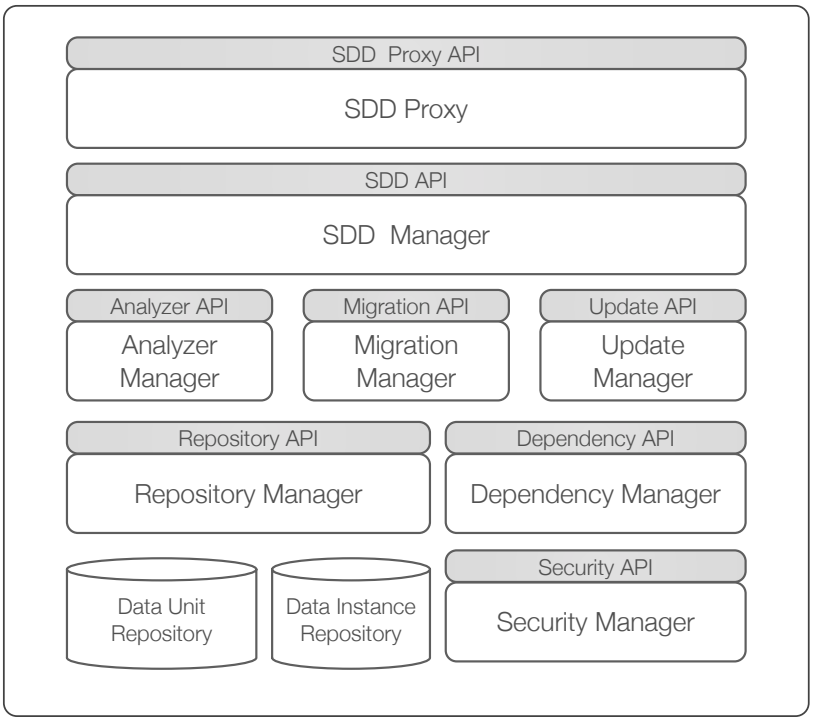


Figure 5.2: SDD Framework Overview

5.4 The SDD Framework

In this section, we introduce the SDD framework for enabling usage-aware distributed datasets, to address the previously introduced requirements. We begin with a framework overview, followed by a detailed description of all framework components and conclude with a comprehensive description of our proof of concept implementation.

5.4.1 Framework Rationales

The Framework with an overview of its main components shown in Figure 5.2 follows the microservice [78] architecture paradigm. It consists of eight main components where each of these components represents a microservice. The components utilize both service based as well as message-oriented communication to exchange information. Specifically we distinguish three different queue types: an *Analyzer Queue*, a *Handler Queue* as well as an *Update Queue*, which will be explained in more detail in the context of the corresponding components.

Additionally, the framework utilizes the principle of *Confidence Elasticity*, a concept we introduced in Section 4.3.1 and successfully applied in our Smart Fabric (Chapter 4) and Smart Brix (Chapter 7) frameworks. In this framework, we use the concept in the *Update Manager* and *Migration Manager* components to select a suitable *Update-* and *Migration Strategy* for a

specific data source. Each strategy is associated with a confidence value ($c \in \mathbb{R}, 0 \leq c \leq 1$), with 0 representing no certainty and 1 representing absolute certainty about the produced result. This convention allows the framework to configure certain confidence intervals to augment the process of choosing an applicable strategy within these two components. These confidence intervals are provided as configuration elements for the framework. If the confidence thresholds are not met, the framework follows an escalation model to find the next strategy that is able to provide results with higher confidence until it reaches the point where human interaction is necessary to produce a satisfactory result. In the context of the *Migration Manager* and *Update Manager* components this means if a migration or update of a data source cannot be performed with a satisfactory confidence, the escalation model will prompt for human interaction to perform said migration or update.

5.4.2 SDD Proxy

The *SDD Proxy* acts as a transparent proxy between clients and data sources (*DAIs*). The proxy itself has two main responsibilities. First, it submits all incoming requests to a *Analyzer Message Queue* before forwarding them to the requested data source. Second, it listens to the *Handler Message Queue* for potential redirections to be taken for a specific request. If the *Handler Message Queue* contains a message for the current request, it is processed by the *SDD Proxy*, the request in question is redirected to the new data source, and the message gets removed from the *Handler Message Queue*. To avoid bottlenecks there can be multiple proxies where each of them is being managed via the *SDD Proxy APIs* by the *SDD Manager*.

5.4.3 SDD Manager

The *SDD Manager* acts as the central management component of the framework and provides the *SDD API* for overall framework control. To activate the framework a user invokes the *SDD API* with the following parameters: *(i)* a set of *Triggers* as well as a *(ii) Confidence Interval* to configure the Confidence Elasticity of the framework. The *SDD Manager* then starts the first *SDD Proxy* and starts monitoring the average request rates as well as the utilization of the proxy via the *SDD Proxy API*. If the *SDD Manager* detects a potential bottleneck it starts another proxy (additional ones if necessary based on the average request rate). The next task of the *SDD Manager* is to submit the provided *Triggers* to the *Analyzer Manager*, which uses them to invoke the corresponding request monitoring. A *Trigger* is used in the analyzer to decide wether a request needs to be handled or not. *Triggers* can for example be time based, size based or follow a customizable cost function and provide a threshold for triggering a handling action. The *Analyzer Manager* uses different pluggable *Analyzer Strategies* in correspondence with these submitted *Triggers* to determine if a request needs to be processed. If this is the case, the *Analyzer Manager* invokes the *Migration Manager* via the *Migration API* and provides the corresponding request including the results of its analysis. The *Migration Manager* is responsible for determining the potential *Migration Strategies* for the data resource in question. To achieve this it first contacts the *Dependency Manager*, which uses a dependency resolution mechanism to determine the corresponding *DAU* for the *DAI* being requested by the current request. The *Dependency Manager* in turn is tightly integrated with the *Security Manager*, which ensures

all security constraints that are defined in the *DAUs* are satisfied before returning the results of the resolution. Once the dependency resolution has provided the *DAU* it is being analyzed by the *Migration Manager*. Specifically it checks the results provided by the *Analyzer Manager* like request time, data size or a respective cost function against the attributes of the specific `view` being requested. It analyzes the update frequency as well as update sizes to determine if a migration should be performed as well as to to determine the fitting *Migration Strategy* and to execute it if applicable. Once the migration is finished the *Migration Manager* executes two tasks. First, it adds the request to the migrated data source to the *Handler Queue* including the new target (*DAI*) after the migration. This in turn triggers the corresponding *SDD proxies* to execute a redirection. Second, it registers the resource at the *Update Manager*, which in turn determines the fitting *Update Strategy* for the migrated data source to ensure that the data stays up to date. Once these steps are successfully finished the *Migration Manager* updates the *DAIs* and *DAUs* to reflect the changes caused by the migration via the *Repository Manager*.

5.4.4 Analyzer Manager

The role of the *Analyzer Manager* is to determine if a request is a potential candidate for a migration. It watches the *Analyzer Queue* for requests that correspond to any of the previously provided *Triggers*. A *Trigger* is a threshold that matches an attribute that is the result of an *Analyzer Strategy*. *Analyzer Strategies* in turn are pluggable mechanisms that analyze a specific request based on the type of the request in question. Basically we distinguish three different types of strategies: *Time Analyzers*, which determine the response time for a specific request, *Size Analyzers*, which determine the size of a request and response as well as *Frequency Analyzers*, which determine the frequency of a request to a certain source. Additionally, there is also the ability to provide *Cost Function Analyzers*, which allow to integrate arbitrary cost functions and enable a much greater analytical flexibility. Once a threshold is met the *Analyzer Manager* submits the request in question including the results of the specific strategy to the *Migration Manager*.

5.4.5 Migration Manager

The *Migration Manager* is responsible for deciding if a data resource should be migrated based on the results of the *Analyzer Manager*. It is invoked via the *Migration API* with a specific request augmented with the results from the corresponding *Analyzer Strategy*. The *Migration Manager* then forwards this request to the *Dependency Manager*, which first determines the *DAI* that belongs to the requested data resource. Based on this *DAI* the *DAU* is determined only if all security constraints are being met, which in turn is ensured by the *Security Manager*. The retrieved *DAU* provides the foundation for deciding if a data source can and should be migrated. To achieve this the *Migration Manager* relies on pluggable strategies that determine if a migration is feasible and possible. Such a *Migration Strategy* receives the retrieved *DAU* as well as the results from the *Analyzer Manager*. Based on this it does two things, first it determines if a migration is possible by checking the results of the *Analyzer Manager* (e.g., response time, average transfer size) against the *updateFrequency* elements of the *DAU* as well as the constraints of the current infrastructure. If the result of this analysis leads to the conclusion that a migration is possible and

feasible the *Migration Strategy* returns according results augmented with a *confidence value*. Second, the *Migration Strategy* provides a method to execute the actual migration. The framework is flexible regarding the specific migration mechanism and regards this as the responsibility of the strategy itself. One possible variant is the utilization of the *Smart Fabric Framework* [88] since its provides an optimal foundation for infrastructure agnostic deployments (hence migrations) and supports the extended system model. In case of a *Smart Fabric Strategy* the *Infrastructure Specifications (IS)* are taken into account when deciding if a migration is feasible. This means the *Migration Strategy* can check which non functional characteristics apply and can incorporate them in the decision to migrate. Additionally the execution of said migration is started by issuing a transfer requests to the *Smart Fabric Framework*. Based on the previously introduced confidence elasticity mechanism the *Migration Manager* then executes the corresponding *Migration Strategy* or in case none is found relies on a human interaction to perform the migration. Once the migration is finished the *Migration Manager* creates new corresponding *DAIs* to reflect the migrations and updates the corresponding *DAUs*. Futhermore, it publishes a message to the *Handler Queue* and by doings so prompts the *SDD Proxy* to execute a redirection to the migrated data source. Once this is done the *Migration Manager* ensures the migrated data source is updated by registering the *DAIs* at the *Update Manager*.

5.4.6 Update Manager

The role of the *Update Manager* is to ensure that migrated data sources stay up to date if the original source is changed. To enable this it relies on pluggable *Update Strategies* and we basically distinguish the following different types: *Simple Copy Strategies*, which copy either a fraction or the entire data source, *Script Update Strategies*, which apply a more complex update strategy based on a script (e.g., rsync, sql scripts), as well as *Streaming Replication Strategies* for continuous updates. These strategies again utilize the *Confidence Elasticity* mechanism by providing a confidence value. Based on the initially provided confidence interval of the framework the *escalation mechanism* selects an applicable *Update Strategy* or in the case none is found relies on a human interaction to perform the update. Once the update is finished the *Update Manager* updates the corresponding *DAI* via the *Repository Manager*.

5.4.7 Dependency Manager

The *Dependency Manager* is responsible for resolving unit dependencies between the modeled data entities, as described in the system model above. To achieve this the dependencies between data entities in the system model are represented as a tree structure. Based on this tree structure the *Dependency Manager* creates a root node for each *DAU*. It then creates a corresponding leaf node for each *DAI* that is referenced in the dependency section of the *DAU*. After this it checks the related *DIs* and adds them as leaves. For every step the *Dependency Manager* also ensures that all security constraints are being met by checking the specific *DAU* with the *Security Manager*. If access is not permitted the resolution is not successful.

5.4.8 Security Manager

The *Security Manager* is responsible for ensuring that all security constraints that apply to a given *DAU* are being met. To do so it checks to facets of a *DAU*. First, it ensures that a *DAU* description can be accessed by checking the `securityConstraints` element in the `metainformation` section. Second, it ensures that each view can be accessed as well as migrated by checking the corresponding `securityConstraints` element in the `view` sections of the *DAU*. To enable an open and evolvable security system the *Security Manager* relies on pluggable *Security Strategies*. Examples of such strategies are *LDAP* or *OAuth* or other approaches like *RBAC* [42], but can be extended to any other suitable security mechanism.

5.4.9 Repository Manager

The *Repository Manager* provides repositories for *DAUs* and *DAIs* and acts as a distributed registry keeping track of specific deployments and participating entities. It is responsible for storing and retrieving the system model. It manages two distinct system model repositories utilizing distributed key value stores, which store the JSON-LD files that represent *DAUs* and *DAIs* in a structured way. The *Repository Manager* provides a service interface to access these files as well as a search interface to query *DAUs* and *DAIs* based on specific elements. Additionally, it is responsible for managing dependencies between *DAUs* as well as *DAIs* and it seamlessly integrates with *Repository Managers* of other SDD framework deployments ensuring a complete *DAU* and *DAI* lookup.

5.4.10 Implementation

For evaluation purposes we created a proof of concept prototype of our framework based on a set of RESTful microservices implemented in Ruby and packaged as Docker[9] containers. Every component that exposes a service interface relies on the Sinatra[10] web framework. To enable the message-based communication for the *Analyzer, Handler* and *Update queues* we used RabbitMQ[11] as message-oriented middleware.

The *Repository Manager* utilize MongoDB[12] as its storage backend, which enables a distributed, open, and extendable key value store for the *DAU* and *DAI* repositories and provides the foundation for the distributed registry. The *SDD Proxy* was implemented as WEBrick[13] proxy server. Additionally, we patched the default Ruby http class in our prototype implementation to enable the transparent proxy behavior.

We implemented the *Analyzer Manager* with two *Analyzer Strategies*. Specifically, we implemented a *Web Request Response Time Analyzer* as well as *Web Request Response Size Analyzer*, which allowed us to analyze response times as well as average sizes of requests and responses. The *Migration Manager* was implemented with two *Migration Strategies*. The first strategy was

[9] https://www.docker.com/
[10] http://www.sinatrarb.com/
[11] https://www.rabbitmq.com/
[12] https://www.mongodb.org/
[13] https://ruby-doc.org/stdlib-2.0.0/libdoc/webrick/rdoc/WEBrick.html

a *MongoDB Strategy* and supports the migration of MongoDB databases and collections, the second one was a *Docker Strategy* that enables a docker based container migration. For the *Update Manager* we reused the *MongoDB Strategy* as *MondoDB Database Copy Strategy* and *MondoDB Collection Copy Strategy* and additionally implemented a file based *SCP Full Copy Strategy*, which transfers files via Secure Copy (`scp`).

5.5 Evaluation

5.5.1 Setup

As basis for our evaluation we used the URBEM Smart City Application (USCA) [94], a holistic interdisciplinary decision support system detailed in Chapter 9, which has been used for city infrastructure planning tasks especially in the context of energy and mobility systems. We choose the USCA, because it represents an optimal candidate for our evaluation due to the following characteristics: *(i)* It heavily relies on a diverse set of data sources, where most of them belong to stakeholders and are under strict security and compliance regulations. *(ii)* It is an application that has to deal with changing requirements that make it necessary to incorporate new data sources dynamically. *(iii)* Due to the nature of the application as a planning tool for energy and mobility systems it is a common case to incorporate data sources from other cities around the globe.

Due to the strict data security regulations we were not allowed to use the original data from the URBEM domain for our evaluation scenario. To overcome this limitation we created anonymized random samples of the most common data sources that are being used in the USCA. The specific datasets we used included building data with different granularity levels, thermal and electrical network data as well as mobility data. Based on these data sources we created *Data Units (DAUs)* for each of them as well as exemplary data services as *Technical Units (TUs)*. As a next step we looked at the common request patterns for these types of data based on the request history of the USCA. Since the USCA relies on user input via its graphical user interface, we created sample clients, which tested these request patterns in order to enable automated testing. Based on these foundations we created two different evaluation scenarios.

For the first scenario we provisioned two VM instances in our private OpenStack cloud, each with 7.5GB of RAM and 4 virtual CPUs. These two instances represented Data Centers in the cities of Vienna and Melbourne. In order to simulate realistic transfer times between these two different regions we used Linux Advanced Traffic and Routing (`tc`) to simulate the average delay between these regions including common instabilities. Each of these instances was running Ubuntu 16.04 LTS with Docker.

For the second scenario we provisioned three VM instances in the Google Cloud Platform. We used `n1-standard-2` instance types each with 7.5GB of RAM and 2 virtual CPUs. In order to get a realistic geographic distribution we started each of these instances in different cloud regions. Specifically, we started one in the `us-central` representing the city of Berkeley, one in the `europe-west` region representing the city of Vienna and one in the `asia-east` region representing the city of Hongkong. Each of these instances was again running Ubuntu 16.04 LTS with Docker.

For monitoring purposes we used Datadog[14] as monitoring platform. We submitted custom metrics for request and response times, and monitored system metrics for bytes sent as well as overall instance utilization to ensure over utilization had no impacts on our results.

5.5.2 Experiments

In this section we give a detailed overview of the conducted experiments within the two scenarios.

Scenario One

In the first scenario we wanted to evaluate the impact of SDD in the context of a simple scenario using USCA for analytics of two cities. We simulated a case in which stakeholders use USCA to compare the impact of city planning scenarios on the building stock, thermal network and public transport between the city of Vienna and Melbourne.

To achieve this we generated 10 data services based on the *DAU* we defined for buildings, networks and mobility as well as the corresponding *DAIs* and deployed them as docker containers on the instance representing Melbourne. We did the same for the instance in Vienna. In the next step we deployed 5 clients simulating the previously mentioned request patterns on the Vienna instance. This setup of clients and sources represented a common sample size of clients and services used in the current USCA context. As a last step we deployed the SDD framework, specifically three containers: one for the *SDD proxy*, one for the *Repository Manager* and one for the other components of the *SDD Framework*. An overview of this evaluation scenario can be seen in Figure 5.3.

In the context of this scenario we distinguished 4 different request types to three different kinds of *DAUs*. The first *DAU*, *buildings* represented a larger data source with low update frequency (once a year). For this resource we had two request types on two views of this resource specifically */buildings* and */blocks* representing two different levels of detail. The second *DAU* was *networks* again representing a larger data source with low update frequency (once quarterly). For this resource we had one request type on the only view of this resource namely */networks*. The last *DAU* was *mobility* representing a smaller data source with high update frequency in the public transport context.

To establish a baseline we started our evaluation with the *SDD Framework* deactivated and monitored the response times of each of these four request types, as well as the transferred bytes from the Melbourne instance through custom Datadog monitors. After 5 minutes we started the *SDD Framework* by submitting a request to the *SDD Manager* on the Vienna instance via the *SDD API* with *Triggers* for response times longer than 3 seconds and a confidence value that matched our automated *Migrations Strategies* since we wanted to test the automated capabilities of the *SDD Framework*. After submitting the request we continued to monitor the results of the Datadog monitors over a course of additional 5 minutes.

The results of our evaluation can be seen in Figure 5.4. In the figure, we see the different characteristics of the response times for the 4 request types. *Buildings*, *blocks* as well as *networks* with longer response times, *mobility* with a rather short response time. Given the submitted *Triggers*

[14] https://www.datadoghq.com/

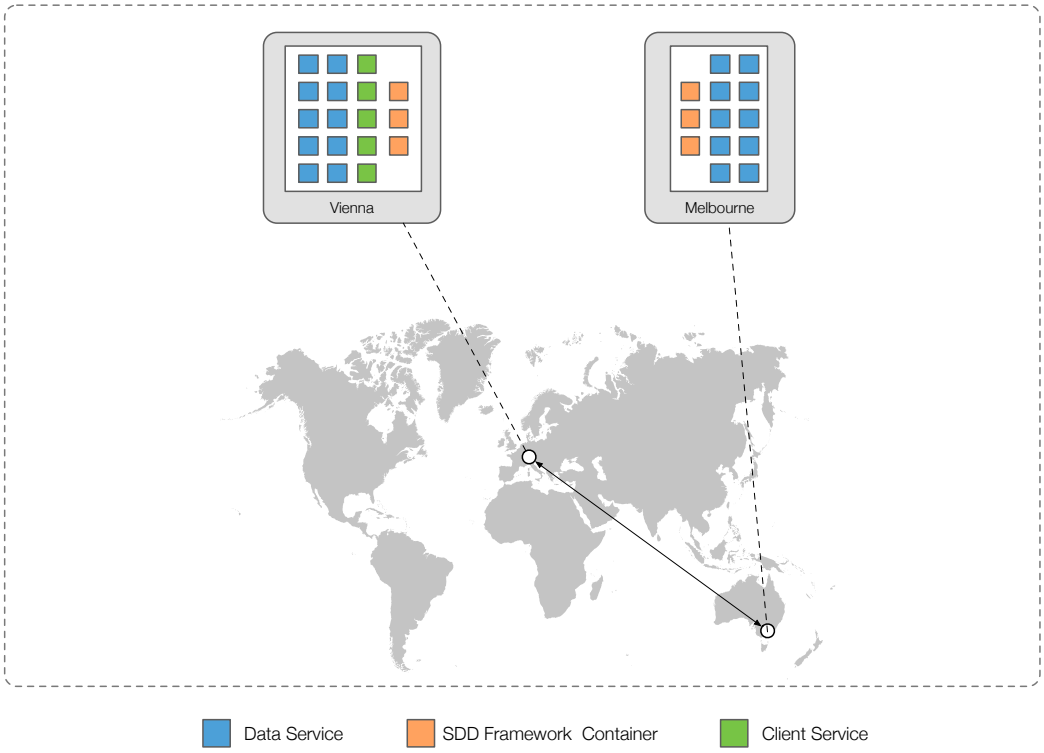


Figure 5.3: Evaluation Setup for Scenario One

as well as the implemented *Migration Strategy* in context with size and update characteristics of the *DAUs*, *Buildings, blocks and networks* qualified for migration. We see that the framework correctly identified these requests and starts migrating the corresponding *DAIs*, around minute 5. The total migration time for all three resources was 59.3 seconds during this time the *SDD proxy* keeps forwarding the requests to the original *DAI*. After the migration has finished and the *SDD Proxy* successfully started redirecting to the new *DAIs*, we see a significant reduction in the average response times for all three request types. The *mobility* source was not migrated since it didn't qualify due to the fact that this resource had response times below the trigger. We also see that the response time for this resource shows an increase, which is due to the specific proxy implementation and caused by the overhead of redirection checks after the framework has been activated and is present for all request types. The efficiency of the specific proxy implementation was not focus of this work and does not influence the validity of the presented results, since the introduced overhead affects all requests equally. Additionally, we see that in this scenario the network transfer from the instance representing Melbourne was reduced by 97% after the migrations finished. This is due to the fact that only the *DAI* for mobility remains active on this instance and no other clients or framework components are actively sending data from Melbourne. The aggregated overview in Table 5.1 shows that average and median response times along with variance in response times for all migrated *DAIs* were reduced significantly.

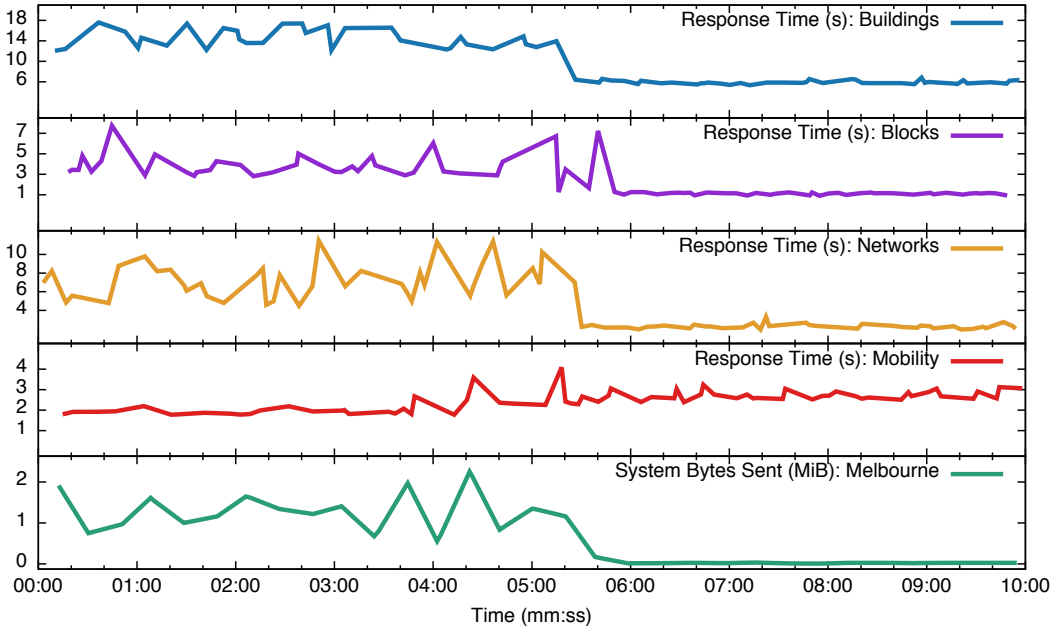


Figure 5.4: Evaluation Results for Scenario One

Request Type	**Status**	**Average Response Time**	**Median Response Time**	**Standard Deviation Response Time**
Buildings	inactive	14.561s	14.272s	1.857s
	active	5.94s	5.848s	0.354s
Blocks	inactive	3.833s	3.425s	1.316s
	active	1.122s	1.154s	0.102s
Networks	inactive	7.21s	6.843s	1.928s
	active	2.298s	2.268s	0.257s
Mobility	inactive	2.196s	1.982s	0.504s
	active	2.727s	2.677s	0.212s

Table 5.1: Average, median and standard deviation for response times per request type in Scenario One

Scenario Two

In the second scenario we wanted to evaluate the impact of SDD in the context of a larger and more complex scenario using USCA for analytics in an internet of cities setup with cities in different regions. We again simulated a case in which stakeholders use USCA to compare the impact of city planning scenarios on the building stock, thermal network and public transport. This time between the cities of Berkeley, Vienna and Hongkong, which were placed in the respective regions of the Google Cloud platform as described in the setup section.

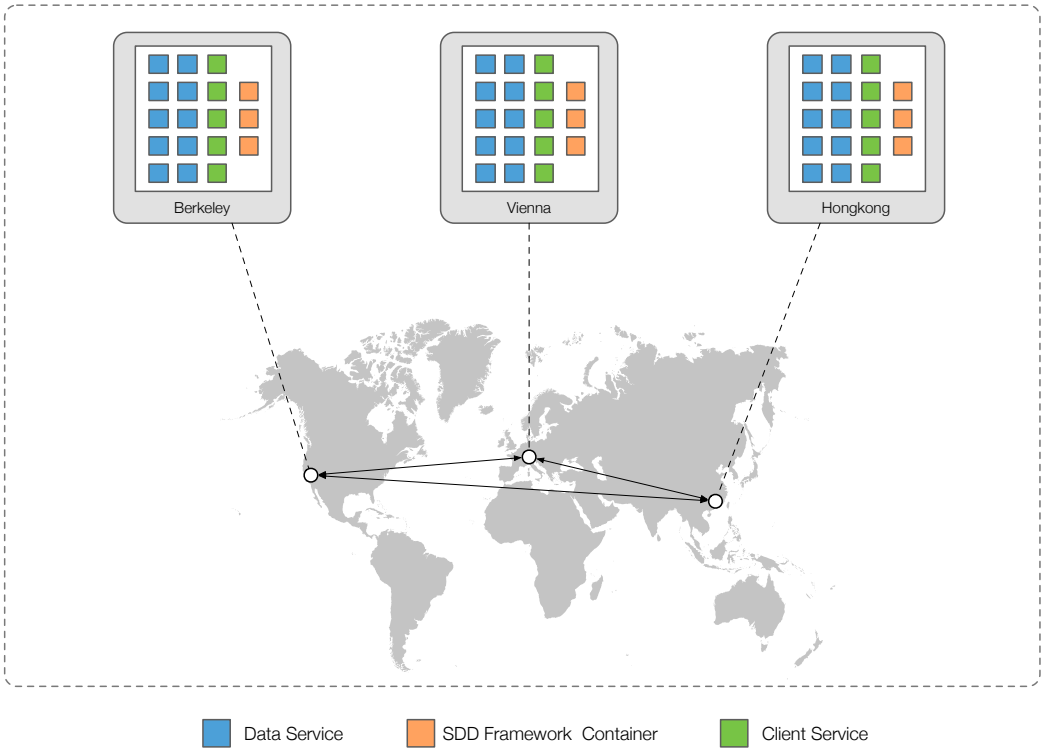


Figure 5.5: Evaluation Setup for Scenario Two

To achieve this we again generated 10 data services based on the *DAU* we defined for buildings, networks and mobility as well as the corresponding *DAIs* and deployed them as Docker containers on all three instances. In the next step we deployed 5 clients per city simulating the previously mentioned request patterns. We then deployed the SDD framework, specifically three containers: one for the *SDD proxy*, one for the *Repository Manager* and one for the other components of the *SDD Framework* on every instance. An overview of this evaluation scenario is depicted in Figure 5.5.

In this scenario we distinguished the same 4 request types as before. To establish a baseline we started our evaluation with the *SDD Framework* deactivated and monitored the response times of each of these four request types, as well as the transferred bytes from all participating instances through custom Datadog monitors. After 5 minutes we started the *SDD Framework* by submitting a request to the *SDD Manager* on each of the three instances via the *SDD API* with *Triggers* for response times longer than 3 seconds and a confidence value that matched our automated *Migrations Strategies* since our focus was again on testing the automated capabilities of the *SDD Framework*. After submitting the requests we continued to monitor the results of the Datadog monitors over a course of additional 5 minutes.

The results of our evaluation can be seen in Figure 5.6. By investigating the Figure, we notice that the framework again correctly identified the three request types to *buildings*, *blocks*

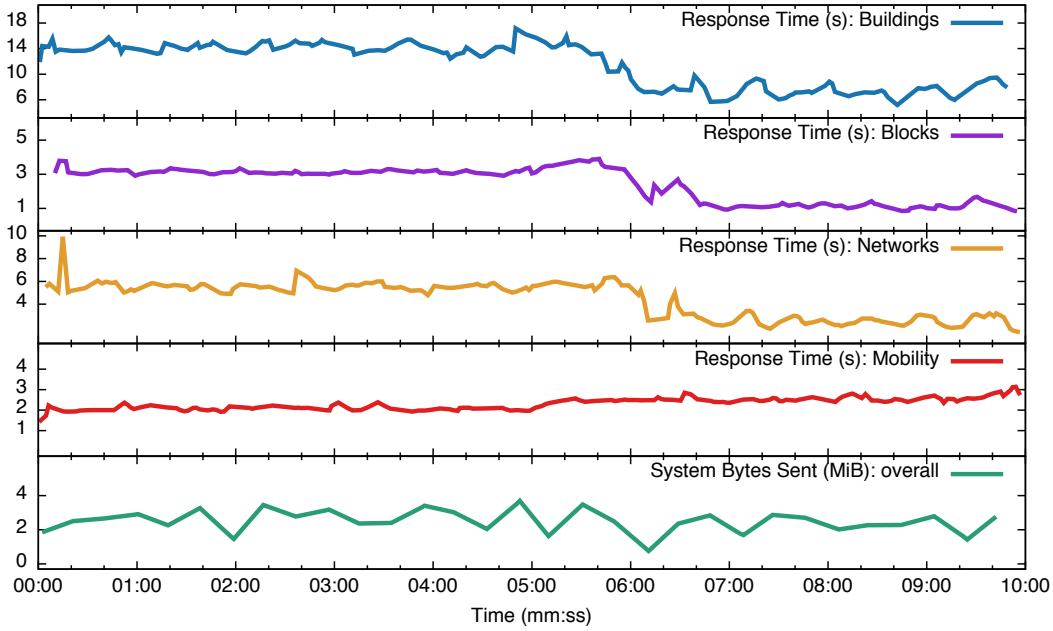


Figure 5.6: Evaluation Results for Scenario Two

and *networks* as candidates for migrations. The framework starts migrating the corresponding *DAIs* around minute 5. In this more complex case the total migration time for all three resources over all three instances was 112.32 seconds. After the migration has finished and the *SDD Proxy* successfully started redirecting to the new *DAIs* we again see a significant reduction in the average response times for all three request types. In terms of network transfer we don't see a significant reduction as opposed to Scenario One since in this setup there are active clients, as well as framework components deployed on all instances that continue to issue requests, hence sending data. The aggregated overview in Table 5.2 shows that average and median response times for all migrated *DAIs* again were reduced significantly. In contrast to the lab environment of Scenario One, a significant reduction in response time variance was not observed, which can be attributed to the performance variability of cloud instances as well as the distribution over the chosen regions.

Our experiments showed that we could significantly reduce the response times and hence the QoS for the URBEM Smart City Application. Specifically we showed a reduction of the response times by 66% on average over all three migrated request types. We also demonstrated that the framework was able to correctly identify the *DAIs* to be migrated utilizing the views specified in the corresponding *DAUs*. Finally we showed that we could produce these results both in a laboratory setting as well as in a geographically dispersed cloud setup.

While the presented system model and framework fulfill the requirements set forth in the context of the previously introduced URBEM Smart City Application, certain threats to the general applicability of SDD remain. The initial evaluation setup for Scenario One used `tc` to introduce the delays between the two instances representing Vienna and Melbourne. It could

Request Type	Status	Average Response Time	Median Response Time	Standard Deviation Response Time
Buildings	inactive	14.214s	14.266s	0.98s
	active	7.825s	7.613s	1.335s
Blocks	inactive	3.203s	3.142s	0.246s
	active	1.273s	1.161s	0.37s
Networks	inactive	5.571s	5.548s	0.582s
	active	2.594s	2.489s	0.578s
Mobility	inactive	2.089s	2.088s	0.162s
	active	2.562s	2.514s	0.156s

Table 5.2: Average, median and standard deviation for response times per request type in Scenario Two

be argued that this simulated setup was not representative for the evaluation. The fact that the experiments showed similar results in a globally distributed deployment refute this claim. Beyond this, the current evaluation relied on simulated clients and data sources for the experiments. To ensure that the used workloads and data sources are realistic and representative, we gathered workload patterns and anonymized example records from the URBEM smart city application used by domain experts.

5.6 Related Work

The recent trend in smart city research towards the introduction of smart city platforms and reference models has been further fanned by the rise of the Internet of Things (IoT). While all of these approaches mention the importance of data management in the context of the massive amount of data, its heterogeneity and multitude of security and compliance constraints, it currently either is not a framework element or they do not provide specific solutions for this problem. Chourabi et al. [23] present a framework to understand the concepts of a smart city on a more abstract level. The authors introduce a conceptual framework to comprehend the vital elements in a smart city by identifying critical factors. In the context of ICT, they identify security, privacy as well as accessibility as central elements for smart city applications. In a similar way, Naphade et al. [76] present innovation challenges for smarter cities. The authors identify the management of information across all cities' information systems including the need for privacy and security as a central challenge in the success of smart cities underlining the importance of a data management approach like ours. On a more concrete level Bonino et al. [15] present ALMANAC, a smart city platform with a focus on the integration of heterogenous services. They identify challenges smart city applications face, also in terms of infrastructure and specifically mention the importance of taking data ownership and exchange between the different smart city stakeholders into account. Compared to our approach however, they do not provide a specific solution to address these challenges, especially none applicable to legacy data.

In the context of IoT where more and more data sources emerge, data management plays

a central role. Jin et al. [50] introduce a smart city IoT infrastructure blueprint. The authors focus on the urban information system starting from the sensory level up to issues of data management and cloud-based integrations. They identify key IoT building blocks for a smart city infrastructure, one of them being the so called Data-centric IoT, in which data management is a central factor. In a similar high level manner Petrolo et al. [81] introduce the VITAL platform as an IoT integration platform to overcome the fragmentation issue in smart cities. They mention data challenges that arise by introducing IoT and specifically underline the importance of privacy and security in this context. The need for data management in order to integrate the produced results also applies to a lot of other IoT platforms in order to make their results accessible for further analytics. Examples of such frameworks are Chen et al. [21], who present a data acquisition and integration platform for IoT. A central element in their architecture is a contextual data platform that needs to integrate with multiple heterogenous data sources. Cheng et al. [22] present CiDAP a city and data analytics platform based on SmartSantander [84] a large-scale testbed that helps with issues arising from connecting and managing IoT infrastructure. They name data management, especially the exchange of data as well as attached semantics as one central challenge. Finally in the context of specific IoT smart city applications, Kyriazis et al. [55] present two sustainable smart city applications in the transportation and energy domain. They clearly identify security in the context of data as a specific challenge in enabling these applications emphasizing the importance of approaches like ours. In the context of IoT platforms and IoT smart city applications our approach provides the missing link for both security aware data management within frameworks, tackling many of the identified challenges as well as providing an ideal way to expose the collected data in a usage-aware distributed way.

A vital element to enable this kind of data management in an efficient way is the ability to migrate data resources. In the context of said migration there are several approaches relevant in the context of our work. Amoretti et al. [4] propose an approach that facilitates a code mobility mechanism in the cloud. Based on this mechanism, services can be replicated to provide a highly dynamic platform and increase the overall service availability. In a similar way, Hao et al. [37] discus a cost model and a genetic decision algorithm that addresses the tradeoff on both service selection and migration in terms of costs, to find an optimal service migration solution. They introduce a framework for service migration based on this adaptive cost model. Opposed to our approach, they focus on the service migration aspect without explicitly addressing the data aspect and do not provide means for incorporating security and other characteristics on a more fine grained data level. In the context of potential migration strategies there are several interesting approaches. Agarwal et al. [1] presents Volley, an automated placement approach for distributed cloud services. Their approach uses logs to derive access patterns and client locations as input for optimization and hence migration. Ksentini et al. [53] introduce a service migration approach for follow me clouds based on a Markov Decision Process (MDP). In a similar way, [117] demonstrate a MDP as a framework to design optimal service migration policies. All these approaches can be integrated as potential migration strategies in our approach.

5.7 Summary

Current smart city application models assume that produced data is managed by and bound to its original application. Such data silos have emerged, in part due to the complex security and compliance constraints governing the potentially sensitive information produced by current smart city applications. While it is essential to enforce security and privacy constraints, we have observed that smart city data sources can often provide aggregated or anonymized data that can be released for use by other stakeholders or third parties. This is especially promising, as such data sources are not only relevant for other stakeholders in the same city, but also other smart cities around the globe. We argue that future smart city applications will use integrated data from multiple sources, gathered from different cities to significantly improve efficiency and effectiveness of city operations, as well as citizen wellbeing. To allow for the creation of such applications, a seamless and efficient mechanism for description and access of available smart city data is required.

In this chapter, we presented Smart Distributed Datasets (SDD), a methodology and framework to enable transparent and efficient distributed data access for data sources in smart city environments. A system model that provides a simple abstraction for the technology-agnostic description of available data sources and their subsets was introduced. Subsets can represent different aspects and granularities of the original data source, along with relevant characteristics common in the smart city domain. Based on this abstraction, we presented the SDD framework, a middleware toolset that enables efficient and seamless data access for smart city applications by autonomously relocating relevant subsets of available data sources to improve Quality of Service (QoS) based on a configurable mechanism that considers request latency, as well as costs for data transfer, storage, and updates. We evaluate the presented framework using a case study in the context of a distributed city infrastructure decision support system and show that selective relocation of data subsets using the SDD framework can improve QoS through significantly reducing response times by 66% on average.

CHAPTER 6

Enabling Distributed Analytical Service Environments for the Smart City domain

In this chapter we introduce Nomads, a framework that enables service mobility in Distributed Analytical Environments (DAEs) while respecting security and compliance constraints. The framework improves the overall satisfiability and therefore also the quality of constrained DAEs. We outline the requirements of a representative DAE scenario, provide a detailed problem formulation, and then discuss the service mobility framework along with our solution finding algorithm. The evaluation demonstrates that the Nomads framework considerably increases the number of successfully performed compositions even in highly constrained environments.

6.1 Introduction

In order to successfully make informed decisions and, more importantly, to plan in the Smart City domain it is elementary to understand the massive amounts of data generated by modern cities. In order to realize this in the smart city domain, stakeholders rely on analyses and models of domain experts. This is accomplished using Distributed Analytical Environments (DAE). DAEs can be considered an instance of Software-defined Elastic Systems for Big Data Analytics [105]. Such a DAE relies on dynamic analytical service compositions which are created based on specific questions from stakeholders to provide the desired results. However, these compositions cannot always be executed due to compliance constraints between service providers. These constraints lead to satisfiability problems of service compositions resulting in the inability to answer stakeholders' questions. In this chapter we propose a framework to overcome this limitation by enabling dynamic service migrations and by doing so delivering a strong quality improvement for constrained DAEs.

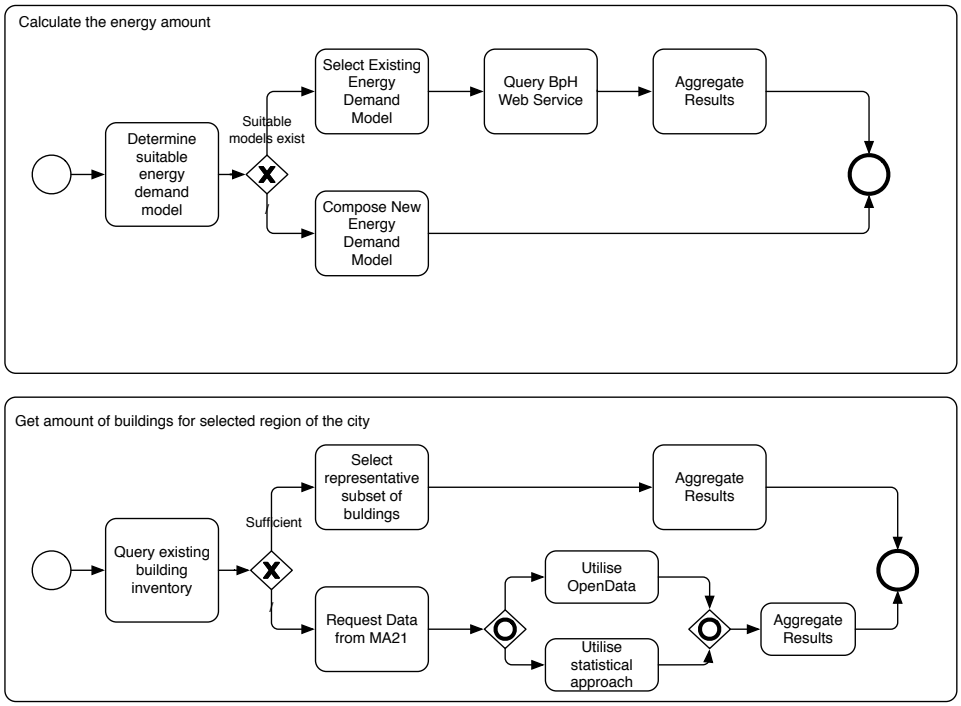


Figure 6.1: Example of an analytical process in URBEM

The remainder of this chapter is structured as follows: In Section 6.2 we present the scenario and outline the specific problem as well as requirements. Section 6.3 introduces the NOMADS framework as an approach to address these problems. This is followed by a validation and short evaluation of our approach in Section 6.4, as well as a discussion of related work in Section 6.5. The chapter concludes with a comprehensive summary in Section 6.6

6.2 Scenario

In this section we present a detailed analysis of the motivating scenario introduced in Chapter 3, specifically the aspect of DAEs in the context of urban planning.Consider the following case of urban planning: The stakeholders want to know the impact photovoltaics would have on the energy grid if they would be installed on every housing area in a specific district. In order to answer this question a plethora of different data needs to run through different models of various domains experts in the areas of building-physics, energy grids and spatial visualization, forming a DAE. Figure 6.1 shows the analytical process underlying such a DAE.

From a technical perspective the models provided by domain experts, as well as the data, are exposed as abstract services, each of these services providing different capabilities. There are services providing data about the spatial aspects of the city, about the thermal and electrical grids,

$$
\begin{array}{c} \\ WL \\ WN \\ WE \\ WIT \\ CoV \\ TU \end{array}
\begin{array}{c}
\begin{array}{cccccc} WL & WN & WE & WIT & CoV & TU \end{array} \\
\left(\begin{array}{cccccc}
\checkmark & \times & \times & \checkmark & \times & \times \\
\times & \checkmark & \times & \checkmark & \times & \times \\
\times & \times & \checkmark & \checkmark & \times & \times \\
\checkmark & \checkmark & \checkmark & \checkmark & \checkmark & \checkmark \\
\checkmark & \checkmark & \checkmark & \checkmark & \checkmark & \checkmark \\
\times & \times & \times & \checkmark & \checkmark & \checkmark
\end{array} \right)
\end{array}
$$

Figure 6.2: Example of data exchange constraints between providers

services that compute the energy demands of buildings, services that provide mobility models and many more. These abstract services are dynamically composed depending on the specific question of the stakeholders. Each of these abstract services can have different specific implementations called concrete services which again originate from multiple providers. In URBEM providers include companies like energy providers and public transportation services, municipalities and city administration, as well as third party solution providers. Due to the variety of data including business critical sensitive information, as well as certain regulatory requirements with numerous compliance aspects there are strict data exchange constraints between these providers, resulting in a constrained DAE. These constraints usually apply to sensitive information (e.g., detailed household energy usage, medical data, etc.), but do not apply to results of analyses that process sensitive data and produce insights or aggregated results that cannot be used to infer the input data from their results.

6.2.1 Problem Description

The first elementary characteristic of constrained DAEs is the fact that only if a concrete service composition is possible the respective question of a stakeholder can be answered, making satisfiability an essential quality metric. The major factor for determining satisfiability are the constraints between providers of specific services. Figure 6.2 shows an example of the constraints matrix between each provider in the URBEM domain.

These constraints can be *bidirectional* as well as *unidirectional*, for example since the liberation of the Austrian energy market the regulatory agency prohibits data exchange from grid infrastructure operators to energy providers. However an exchange in the opposite direction is possible. In the exchange matrix this is represented by each row being the “from” and each column being the “to” provider. This also is important for a migration of a service. Even though the grid operator is not allowed to exchange data with the energy provider, it would be possible for the consuming service of the provider to migrate to infrastructure controlled by the grid operator and successfully produce results while respecting all constraints.

The second elementary aspect of constrained DAEs is the highly dynamic nature of the service composition. Since specific compositions of services are triggered by stakeholders’ questions they are not known a priori. This leads to the need for a dynamic mechanism to move concrete services between providers. However, the migration of certain services might not be possible or feasible. An example for this case are certain sensitive data services or services dealing with large amounts of data.

Figure 6.3 shows an overview of the problem. A specific question of a stakeholder leads to a service composition that is necessary to answer this question, as depicted at the top of the figure. Due to the provider constraints it is however not possible to satisfy this question with the current deployment of concrete services. This leads to a satisfiability problem of the DAE and the inability to answer the question, therefore significantly impacting the quality. However with the ability to migrate concrete services on demand the constraints can be satisfied and the desired results can be produced.

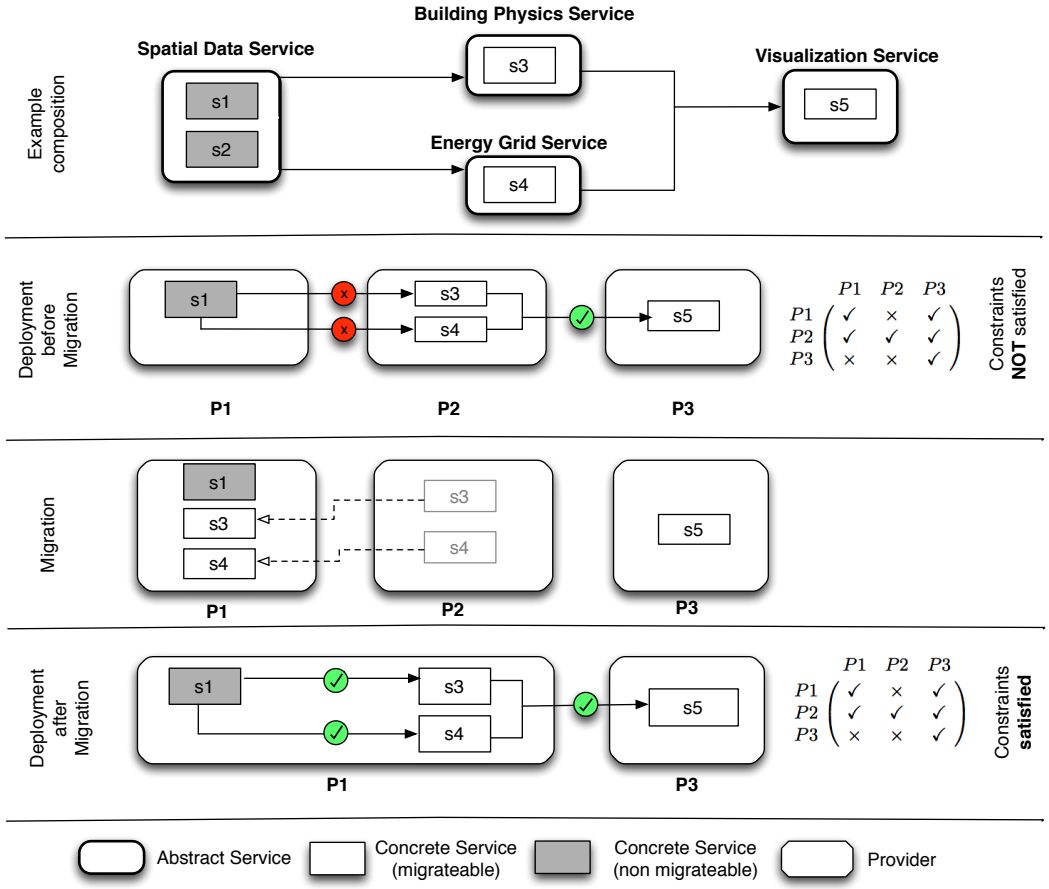


Figure 6.3: Composition Scenario with Exchange Constraints and Service Migration

We can therefore state the following essential requirements in context of the outlined problem.

- Constrained Distributed Analytical Environments depend on the satisfiability of concrete service compositions under consideration of constraints. Service migration is an essential factor to enable this satisfiability.
- Due to the dynamic nature of the service composition triggered by specific stakehold-

$$
\begin{array}{c} \\ P_1 \\ P_2 \\ \vdots \\ P_n \end{array}
\begin{array}{c}
\begin{array}{cccc} P_1 & P_2 & \dots & P_n \end{array} \\
\begin{pmatrix}
1 & E[P_1,P_2] & \dots & E[P_1,P_n] \\
E[P_2,P_1] & 1 & \dots & E[P_2,P_n] \\
\vdots & \vdots & \vdots & \vdots \\
E[P_n,P_1] & E[P_n,P_2] & \dots & 1
\end{pmatrix}
\end{array}
$$

Figure 6.4: Data exchange constraints matrix between providers

ers, which cannot be known a priori, a dynamic mechanism allowing service mobility is required.

6.3 Nomads

In this section we introduce the Nomads framework as an enabling platform for executing complex DAEs. We start by formalizing a system model followed by a detailed framework architecture description.

6.3.1 System Model

This section introduces the system model considered in this chapter. Table 6.1 summarizes the model symbols, and provides a brief example with reference to the scenario of Section 6.2. $\mathscr{P}(X)$ denotes the power set of a given set X, and $M[m,n]$ denotes the entry in row m and column n of a matrix M.

Basic Model. A service composition in our problem domain consists of a multitude of abstract services (A), whose functionality is implemented by one or more concrete services (S). The function $s : A \to \mathscr{P}(S)$ maps abstract to concrete services. The set of providers collaborating within the composition is denoted as P, and each service $s \in S$ originates from one provider $o(s) \in P$. D denotes the set of pairwise data dependencies between abstract services. The abstract services in A (as nodes) and the dependencies D (as edges) span up a directed acyclic graph (DAG), which defines the execution flow of the service composition.

Constraints. The core motivation of this work is the fact that providers have constraints concerning data exchange, which are expressed in our model using a matrix representation E. An example of such a constraint matrix can be seen in Figure 6.4.

Migration and Instantiation. In order to instantiate the composition defined so far in this section, we need to 1) select concrete services, and 2) determine the providers, which are needed to execute the code of each service. That is, a valid instantiation $i \in I$ determines for each abstract service the concrete service and executing provider, expressed by the mapping $A \to S \times P$. Note that, due to data exchange constraints in E, the instantiation may require to migrate the code of some services from one provider to another. Given an instantiation $i \in I$, any concrete service $s \in S$, which is supposed to be executed by a provider $p \in P$ but in fact originates from some other provider, i.e., $o(s) \neq p$, needs to be migrated to p for execution (see definition of function m_i in Equation 6.1). Further details concerning the motivation and necessity of service migration follow in Section 6.3.2.

Symbol	Description	Example
A	Set of abstract services	$A = \{a_1, ..., a_4\}$
S	Set of concrete services	$S = \{s_1, ..., s_8\}$
P	Set of service providers	$P = \{p_1, ..., p_4\}$
$s : A \to \mathscr{P}(S)$	Maps abstract to concrete services	$s : a_1 \mapsto \{s_1, s_2\}, ...$
$o : S \to P$	Provider from which a concrete service originates	$o : s_1 \mapsto p_1, s_2 \mapsto p_1, ...$
$D \subseteq A \times A$	Data dependencies between pairs of abstract services	$D = \{(a_1, a_3), (a_2, a_3), (a_3, a_4)\}$
$E \subseteq \{0,1\}^{\lvert P\rvert \times \lvert P\rvert}$	Data exchange constraint matrix: $E[x,y] = 1$ if provider p_x can exchange data with provider p_y	$E = \begin{pmatrix} 1 & 0 & \cdots & 1 \\ 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 1 \end{pmatrix}$
$I = [A \to S \times P]$	Set of possible runtime instantiations (selection of concrete services and executing providers)	(all instantiations which satisfy the constraints in E, see example below)
$i \in I$	Runtime instantiation	$i : a_1 \mapsto (s_1, p_1), a_2 \mapsto (s_3, p_2), a_3 \mapsto (s_6, p_4), a_4 \mapsto (s_8, p_2)$
$m_i : S \to P$	Services to be migrated to different provider (for instantiation $i \in I$)	$m_i : s_8 \mapsto p_2$
$\pi : S \mapsto \{0,1\}$	Migration policy (is a service allowed to migrate)	$\pi : s_1 \mapsto 0, s_2 \mapsto 1 \ldots$

Table 6.1: Model for Compositions with Constraints and Service Migration

$$\begin{aligned} &\forall i \in I : m_i = \\ &\{(s,p) \mid \exists a \in A : i(a) = (s,p) \wedge o(s) \neq p\} \end{aligned} \tag{6.1}$$

6.3.2 Problem Formulation

Based on the system model introduced in Section 6.3.1 we now provide a detailed formulation of the problem studied in this work.

Given the model of a service composition (A, S, P, D) and the matrix E, we seek for a valid instantiation $i \in I$ such that all constraints in E are satisfied (see Equation 6.2).

$$\begin{aligned} &\forall (a_x, a_y) \in D, i(a_x) = (s_x, p_x), \\ &i(a_y) = (s_y, p_y) : E[p_x, p_y] = 1 \end{aligned} \tag{6.2}$$

The problem is that, without service migration, it may be infeasible to find a valid instantiation under the information exchange constraints defined in E (see Section 6.3.1). Hence, the possibility of migrating services is the central assumption of our approach. Assuming two abstract services $a_x, a_y \in A$ connected via a data dependency, the concrete service s_y for a_y (i.e., $s_y = i(a_y)$) needs to be migrated to the provider of a_x (denoted p_x) if there is a constraint between the providers of the two services and s_y is allowed to migrate (i.e., $\pi(s_y) = 1$) (see Equation 6.3).

$$\begin{aligned} &\exists (a_x, a_y) \in D, s_y = i(a_y), \\ &p_x = o(i(a_x)) : E[p_x, o(s_y)] = 0 \\ &\wedge\, E[o(s_y), p_x] = 1 \;\wedge\; \pi(s_y) = 1 \Longrightarrow (s_y, p_x) \in m_i \end{aligned} \tag{6.3}$$

Finding a valid instantiation i under the conditions in Equations 6.2 and 6.3 is a hard computational problem. A complexity analysis of the problem is out of scope in this chapter, our focus here is to provide a framework for finding valid instantiations and performing the necessary service migrations. Details of our approach are discussed in Section 6.3.3.

6.3.3 Framework Architecture

In this section, we outline the architecture of our framework to address the need for service mobility as discussed in Section 6.3.2. Every service composition to be instantiated is executed within a network of containers to allow for necessary constraint enforcement and enable the novel service migration mechanism. Each provider who is part of a composition instantiation has an on-premise deployment of a Nomads container instance to handle constraints as well as service migrations. Additionally there exist a number of Service containers able to execute local and transfered services. A graphical overview of the Nomads container as well as the Service container in the context of a provider is shown in Figure 6.5.

The coordination between all Nomads containers is handled via a distributed consistent key value store. In our prototypical implementation of the framework we use etcd[1] as key value store which relies on the Raft consensus algorithm [3]. Each Nomads container, in the container network, registers via a unique token per container network. Nomads Container can find and interact with each other via this unique token. If a new provider joins the container network the Nomads container in the provider deployment can register via this unique token. This not only allows us to dynamically extend the Nomads container network if new providers are being added it also enables multiple independent Nomads container networks by using different unique tokens.

Stakeholder questions that need to be answered by a composition instantiation can arrive at any request router (RR) in the container network. The RR then hands over the request to the constraint manager (CM) component to determine if the potential instantiation is feasible based

[1] https://github.com/coreos/etcd

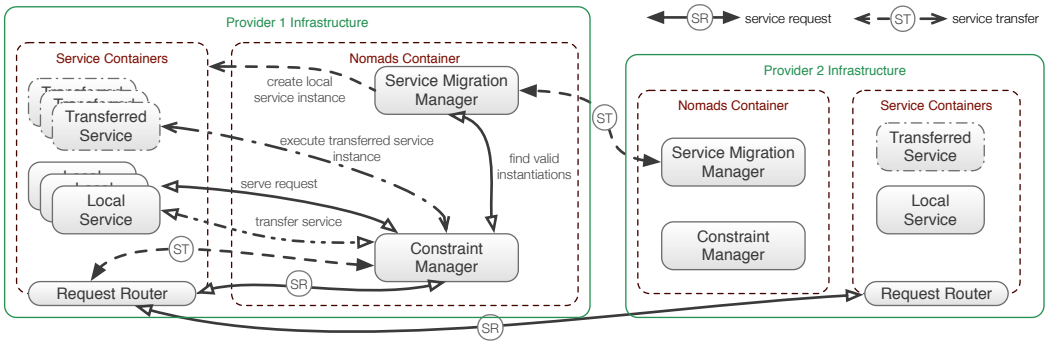


Figure 6.5: Nomads Architecture

on previous requests. If no information about prior executions is available, the CM invokes the service migration manager (SMM) to find a valid instantiation in coordination with all known partner containers. The container SMMs elect one master SMM to perform the search for a valid instantiation. The SMM components share their provider's information exchange constraints with the elected master SMM. The master SMM then searches for a valid instantiation relying on the modularly designed Solution Component (SC) within the SMM. The SC can utilize different pluggable algorithms to find valid instantiations, for demonstration purposes we implemented a very basic algorithm. The pseudo code of the algorithm that is used is depicted in Algorithm 1. Here, we focus on finding valid solutions for previously infeasible composition instances. The algorithm uses a depth-first search to find a possible solution, which is sufficient in the context of this chapter since we focus on finding valid solutions for previously infeasible composition instances. Figure 6.6 demonstrates a simplified traversal through the search space.

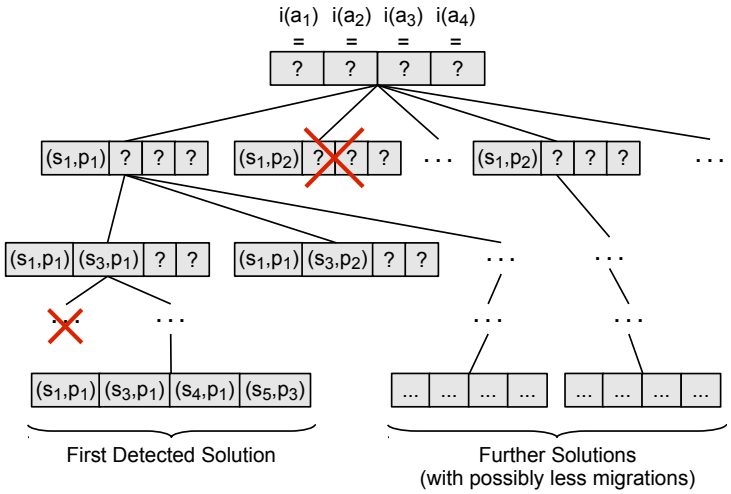


Figure 6.6: Traversal of the search space

Algorithm 1 Find a solution with migrations

```
1: function FINDVALIDSOLUTION(fixedServices, abstractServices)
2:     if abstractServices has elements left then
3:         Take first service of abstractServices
4:         for each concreteService from service do
5:             for each provider from all providers do
6:                 Try assign concreteService to provider based on migration policy
7:                 Add concreteService to fixedServices
8:                 if fixedServices are valid based on constraints then
9:                     Recurse with fixedServices and remaining abstractServices
10:                end if
11:            end for
12:        end for
13:    else
14:        if fixedServices are valid then
15:            return solution
16:        end if
17:    end if
18: end function
```

First the number of Abstract Services is determined, which can vary depending on the service composition. Then for each Abstract Service a possible Concrete Service and provider combination is assigned. This assignment respects possible migration policies. If the assignment is valid according to the Data Constraints the Concrete Service provider pair is fixed and the algorithm recurses over the remaining Abstract Services. This traverses the search space until a possible solution is found.

The SC component can be extended to utilize more sophisticated optimizations like genetic algorithms. This also allows to incorporate multiple objectives for optimizations enabling for example, not just finding an instantiation but the optimal one, or one with minimal migrations etc. When a suitable instantiation was found, the respective service migration managers request all necessary service migrations from their partner containers to initiate a valid instantiation. The migrations are performed by moving Service Containers between providers. The Nomads framework offers a pluggable mechanism to implement different migration approaches. In the current version of the prototype, services are relocated using application container migration, providing an implementation-agnostic way for transferring service code, suitable for stateless services. Control is subsequently returned to the CM, which forwards the request to the appropriate local or transferred service instances to answer the stakeholder's question. An exemplary sequence diagram illustrating request handling is shown in Figure 6.7.

All service requests performed during the execution of a composition instantiation are intercepted by the RR to enforce all modeled constraints and ensure the sandboxed execution of transferred services on "foreign" premises. The CM component is integrated with the SeCoS (Secure Collaboration in service-based Systems) framework [41], but also allows the utilization

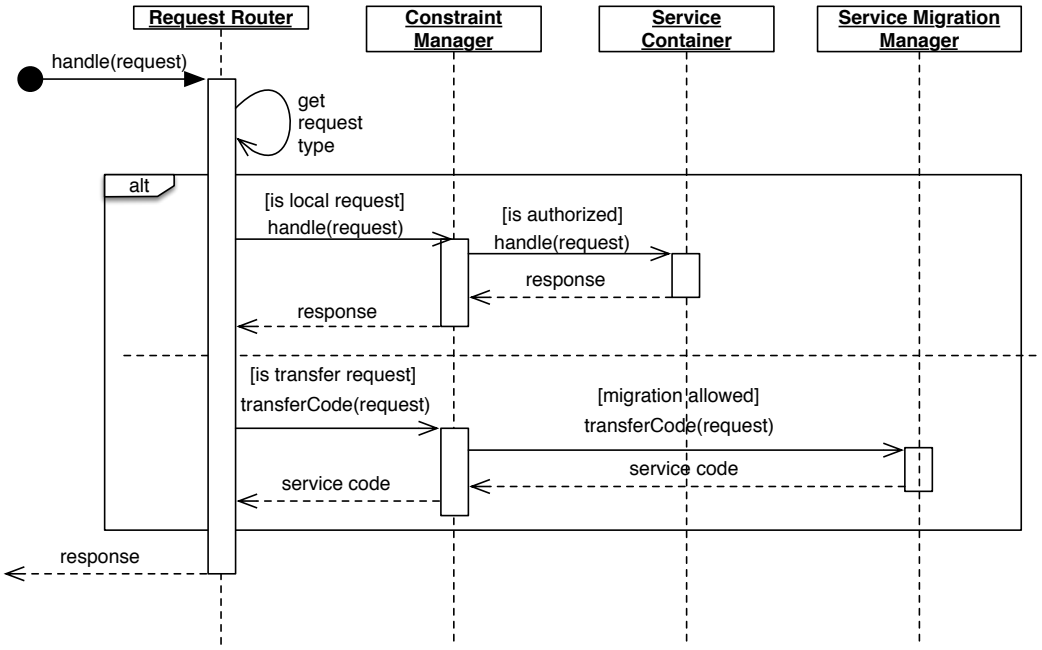


Figure 6.7: Nomads Request Sequence Diagram

of different constraint checking mechanisms.

6.4 Validation

For validation purposes we create three CoreOS[2] based clusters representing different providers from our URBEM scenario, each of them consisting of three CoreOS hosts. On each of these hosts we deploy different Docker[3] based service containers representing various concrete services. Each of these concrete services represents an specific instance of an abstract service relevant to satisfy the example process illustrated in Figure 6.1. Additionally to these service containers we deploy service containers representing random services with varying loads and communication patterns to simulate an environment closer to a real world setting. In each of these clusters we further deploy one Nomads container. In each of these clusters there is a private Docker Registry present. To demonstrate the actual migration of Service containers we transfer Docker images between the private registries and start them accordingly in their new location. This sample deployment is illustrated in Figure 6.8.

We then generate an exemplary exchange constraint matrix ensuring that several service containers need to be transfered in order to satisfy the execution of our example process. Based

[2]https://coreos.com/

[3]https://www.docker.com/

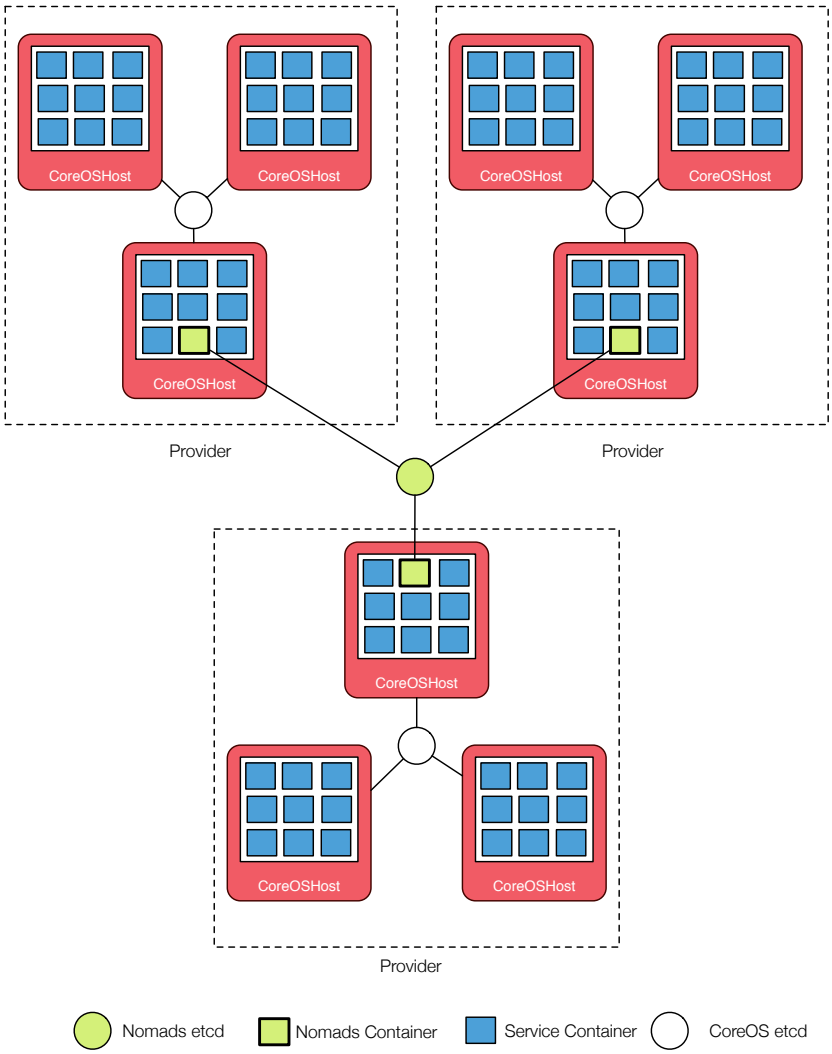
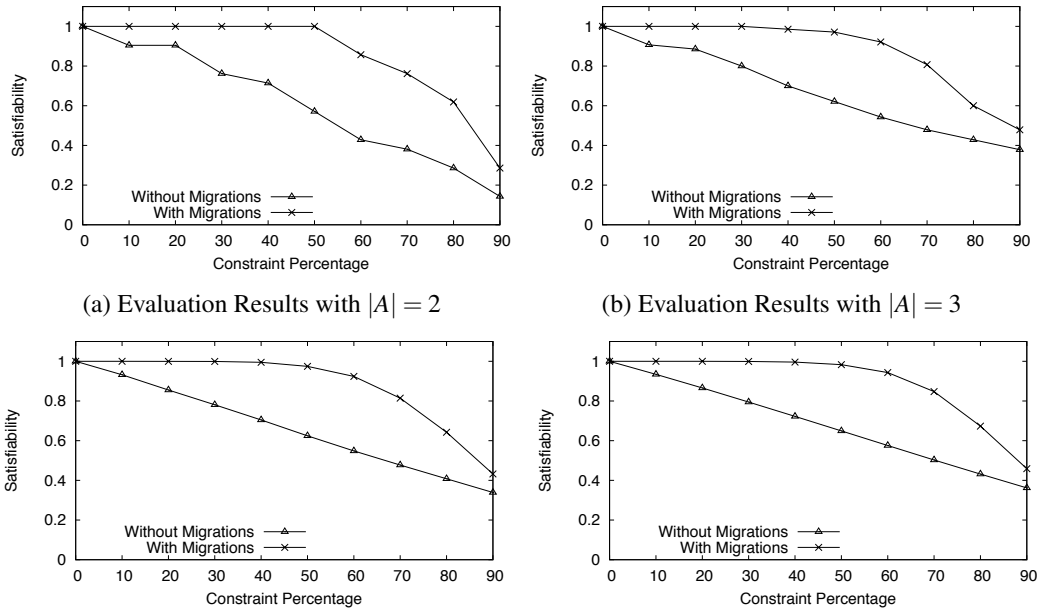


Figure 6.8: Deployment for validation purposes

on these setting we execute the example process and show that the necessary service migrations can be performed in order to satisfy the exchange constraint and successfully execute the process.

6.4.1 Evaluation

Additionally to our validation we perform an evaluation to test the claim that our framework increases the satisfiability of a DAE. For the evaluation, we generate all directed acyclic graphs representing an interaction of up to 7 Abstract Services, which represents the current maximum of Abstract Services in URBEM. This amounts to 2^{21} different graphs representing service



(a) Evaluation Results with $|A| = 2$

(b) Evaluation Results with $|A| = 3$

(c) Evaluation Results with $|A| = 5$

(d) Evaluation Results with $|A| = 7$

Figure 6.9: Evaluation Results

compositions to answer possible stakeholder questions. For these graphs we then generate a random set of Concrete Services with random deployments distributed among the 6 providers in the URBEM scenario. Based on this we create a data constraint matrix representing the Interaction Constraints between the providers.

We start with a theoretical optimum that allows every provider to exchange data with every other provider to determine the theoretical optimal baseline. To show the impact of data exchange constraints on satisfiability of composition instantiations, we increase the number of active constraints in 10% increments up to a maximum of 90%, leading to a scenario where only 10% of all possible provider interactions are allowed. For each constraint percentage and graph we determine the number of possible solutions without and with migrations.

Figure 6.9 shows the results of our evaluation for $|A|$ $2-7$. We see that with the increase of Abstract Services the amount of possible solutions and therefore also the minimal possible satisfiability increases. This also leads to a more linear decrease of satisfiability both with and without migrations. In conclusion Figures 6.9a-6.9d clearly illustrate that our approach with service migrations significantly improves satisfiability of composition instantiations.

6.5 Related Work

Due to the holistic nature of compliance constrained Distributed Analytical Environments relevant related research areas cover the following topic like (i) compliance and constraint based service compositions, (ii) constraint satisfaction and enforcement in Role Based Access Control

(RBAC) systems, and (iii) service and code mobility mechanisms that can be used to deal with constraints in terms of compliance or availability.

Daniel et al. present challenges in SOA-based compliance governance [26], by defining research goals in compliance governance and a compliance management life cycle. One of the goals, which is related to our scenario, is data outsourcing, where privacy constraints are enforced by combining data fragmentation with encryption. In addition to the definition of the basic concepts of compliance, [79] proposes an approach that supports data in both service choreography modeling and analysis. The authors present a model that uses data-aware interactions as the basic event. Based on this model, choreographies can be analyzed with the focus on data compliance, by avoiding state space explosion. Next to compliance another important aspect in service compositions are constraints in general, therefore Zhao et al. present a service composition model based on constraints [123], where requirements are defined as a group of constraints for an abstract service workflow. On top of the defined constraints a concrete service workflow can be generated by binding an activity with an appropriate service in terms of constraint satisfaction. Although the authors present the overall approach, they do not consider the workflow verification and validation at run time. Instead of just defining the overall model for creating constraint-aware service compositions, [116] proposes a constraint-aware service composition method based on two concepts, service intension and service extension. The authors use a graph-based search algorithm to generate all feasible solutions for a general service composition problem. Aggarwal et al. [2] present a constraint driven web service composition approach in the METEOR-S framework, which enables the composition of web services, based on both business and process constraints. The general idea of the authors is to transform the overall service composition problem in a general constraint satisfaction problem. Our problem domain of data exchange constraints also relates to the field of quality attributes and service level agreements (SLAs), where constraints have been intensively studied, for instance in the recent work by Ivanovic et al. [47].

Another related field is the area of access constraints, including Role Based Access Control (RBAC), in the context of web services. Hummer et al. [41] propose *SeCoS*, a framework for model-driven definition of RBAC constraints in service-based business processes. The authors present a runtime enforcement mechanism which intercepts service invocations and therefore prevents the actual invocation in case of a policy violation. As the proposed approach is generic and not limited to RBAC constraints, our framework builds on SeCoS and uses it as the foundation for constraint enforcement. Memon et al. [70] present the SECTISSIMO framework, which aims at modeling constraints in security-critical services. Faravelon et al. [30] propose access restrictions in service compositions based on Computational Tree Logic. In contrast to our work, none of the aforementioned works considers service mobility as a solution to resolve data access and data exchange constraints. Since there are several techniques available that can be facilitated to deal with constraints in service compositions, we focus on the concept of service migration. Amoretti et al. [4] propose an approach that facilitates a code mobility mechanism in the cloud. Based on this mechanism, services can be replicated to provide a highly dynamic platform and increase the overall service availability. Rao et al. [83] provide an extensive survey of automated web service composition methods, together with Sirin et al. [101] this provides a good point of departure for potential composition implementations. Next to a framework for service migration, [37] discusses a cost model and a genetic decision algorithm that addresses the tradeoff on both

service selection and migration in terms of costs, to find a optimal service migration solution. As the mobility of code plays an important role in the overall service migration process, Carzaniga et al. [17, 18] provide a study of code mobility paradigms. The authors classify mobile systems into three categories: remote evaluation, code on demand, and mobile agent. Based on these categories the authors discuss abstractions that are related, to those in traditional architectural styles. The work In addition to the aforementioned categories, [16] describes mobile code paradigms with regard to network security vulnerabilities. Following these definitions and paradigms there are several centralized [19, 67] and decentralized [9, 64] approaches available to implement code mobility. Especially the approach proposed in [9], where Arden et al. describe a decentralized computing platform for running mobile code, based on explicit policies for confidentiality and integrity, will be further investigated in the context of our scenario and possibly applied as one of the migration techniques.

In this chapter we focus on the specific challenges of holistic aspects of smart city analytical environments. The approaches discussed above are orthogonal to our framework and can be considered to implement framework aspects.

6.6 Summary

In this chapter we presented Nomads a framework for service mobility in Distributed Analytical Environments. We described the URBEM scenario as an example of a constrained Distributed Analytical Environment (DAE). Based on this scenario we outlined the main problem and requirements of aforementioned DAEs and delivered a comprehensive problem formalization. We introduced Nomads to address this problem, described its architecture as well as a solution finding algorithm and concluded with an validation and evaluation that clearly showed that the framework is feasible and that we could significantly increase satisfiability of constrained DAEs.

The problem of service mobility is a core issue in DAEs, and we anticipate that it will gain increased momentum, also in the broader service research community and considering trends like Big Data [98] or Microservices [57].

CHAPTER 7

A Continuous Evolution Framework for Container Application Deployments

In this chapter, we introduce Smart Brix, a framework for continuous evolution of container application deployments. Smart Brix integrates and unifies concepts of continuous integration, runtime monitoring, and operational analytics. Furthermore, it allows practitioners to define generic analytics and compensation pipelines composed of self-assembling processing components to autonomously validate and verify containers to be deployed. We illustrate the feasibility of our approach by evaluating our framework using a case study from the smart city domain. We show that Smart Brix is horizontally scalable and runtime of the implemented analysis and compensation pipelines scales linearly with the number of container application packages.

7.1 Introduction

In recent years, we have seen widespread uptake of operating system virtualization based on containers [102] as a mechanism to deploy and manage complex, large-scale software systems. Using containers, developers create self-contained images of application components along with all dependencies that are then executed in isolation on top of a container runtime (e.g., Docker[1], rkt[2], or Triton[3]). By packaging application components into self-contained artifacts, developers can ensure that the same artifact is consistently used throughout the complete software release process, from initial testing to the final production deployment. This mechanism for application deployment has become especially popular with practitioners executing projects following DevOps [44] principles. Based on the convergence of development and operations, DevOps advocates a high degree of automation throughout the software development lifecycle (e.g., to implement continuous delivery [40]), along with an associated focus on deterministic creation,

[1] https://www.docker.com/
[2] https://github.com/coreos/rkt
[3] https://www.joyent.com/

verification, and deployment of application artifacts using Infrastructure as Code (IaC) [77] techniques, such as *Dockerfiles*[4] for containerized applications.

These properties allow for straightforward implementation of immutable infrastructure deployments, as advocated by IaC approaches. Application container images are usually created using a layered structure so that common base functionality can be reused by multiple container images. Application-specific artifacts are layered on top of a base file system so that for subsequent updates only the modified layers need to be transferred among different deployment environments. Container engine vendors such as Docker and CoreOS provide public repositories where practitioners can share and consume container images, both base images for common Linux distributions (e.g., Ubuntu, CoreOS, CentOS, or Alpine) to subsequently add custom functionality, as well as prepared application images that can be directly used in a container deployment. Once uploaded to a repository, a container image is assigned a unique, immutable identifier that can subsequently be used to deterministically deploy the exact same application artifact throughout multiple deployment stages. By deploying each application component in its own container[5], practitioners can reliably execute multiple component versions on the same machine without introducing conflicts, as each component is executed in an isolated container.

However, since each container image must contain every runtime dependency of the packaged application component, each of these dependency sets must be maintained separately. This leads to several challenges for practitioners. Over time, the number of active container images grows due to the introduction of new applications, new application components, and updates to existing applications and their components. This growing number of container images inherently leads to a fragmentation of deployed runtime dependencies, making it difficult for operators to ensure that every deployed container continues to adhere to all relevant security, compliance, and regulatory requirements. Whenever, for instance, a severe vulnerability is found in a common runtime dependency, practitioners either have to manually determine if any active container images are affected, or initiate a costly rebuild of all active containers, irrespective of the actual occurrence of the vulnerability. We argue that practitioners need a largely automated way to perform arbitrary analyses on all container images in their deployment infrastructure. Furthermore, a mechanism is required that allows for the enactment of customizable corrective actions on containers that fail to pass the performed analyses. Finally, in order to allow practitioners to deal with the possibly large number of container images, the overall approach should be able to adapt it's deployment to scale out horizontally.

In this chapter, we present Smart Brix, a framework for continuous evolution of container applications. Smart Brix integrates and unifies concepts of continuous integration, runtime monitoring, and operational analytics systems. Practitioners are able to define generic analytics and compensation pipelines composed of self-assembling processing components to autonomously validate and verify containers to be deployed. The framework supports both, traditional mechanisms such as integration tests, as well as custom, business-relevant processes, e.g., to implement security or compliance checks. Smart Brix not only manages the initial deployment of application containers, but is also designed to continuously monitor the complete application deployment topology to allow for timely reactions to changes (e.g., in regulatory frameworks or discovered

[4] https://docs.docker.com/engine/reference/builder/

[5] https://docs.docker.com/engine/articles/dockerfile_best-practices/

application vulnerabilities). To enact such reactions to changes in the application environment, developers define analytics and compensation pipelines that will autonomously mitigate problems if possible, but are designed with an escalation mechanism that will eventually request human intervention if automated implementation of a change is not possible. To illustrate the feasibility of our approach we evaluate the Smart Brix framework using a case study from the smart city domain. We show that the runtime of the implemented analysis and compensation pipelines scales linearly with the number of analyzed application packages, and that it adds little overhead compared to container acquisition times.

The remainder of this chapter is structured as follows. In Section 7.2 we present a motivating scenario and relevant design goals for our framework. We present the Smart Brix framework in Section 7.3, along with a detailed discussion of the framework components. In Section 7.4 we evaluate our approach using a case study from the smart city domain followed by a detailed discussion of the results in Section 7.5. Related work is discussed in Section 7.6, followed by a comprehensive summary in Section 7.7.

7.2 Motivation

In this chapter, we base our discussion on a multi-domain expert network as presented within the URBEM scenario (Section 3.1). The experts in this scenario rely on a multitude of different models and analytical approaches to make informed decisions based on the massive amounts of data that are available about the city. In turn, these models rely on a plethora of different tools and environments that lead to complex requirements in terms of providing the right runtime environment for them to operate. The used tools range from modern systems for data analytics and stream processing like Cassandra and Spark, to proprietary tools developed by companies and research institutes with a large variance in specific versions and requirements to run them. Additionally, these domains have to deal with a broad range of different stakeholders and their specific security and compliance requirements. Models sometimes need to tailor their runtime environment to specific technology stacks to ensure compliance or to be able to access the data they need. Managing and satisfying all these requirements is a non-trivial task and a significant factor hindering broader adoption. Therefore, this environment offers an optimal case for the advantages that come with the use of container-based approaches. Operations teams that need to integrate these models no longer need to be concerned with runtime specifics. Experts simply build containers that can be deployed in the heterogenous infrastructures of participating stakeholders.

However, several challenges remain. In URBEM the team of experts with their plethora of different models created over 250 different images that serve as the foundation for running containers. The models in these containers are fueled by data from several different stakeholders in the scenario, ranging from research institutions in the City of Vienna to industry stakeholders in the energy and mobility domain. Each of them mandates a very distinct set of security and compliance requirements that need to be met in order to run them. These requirements in turn are subject to frequent changes and the containers need to be able to evolve along with them. Additionally, even though the container approach provides isolation from the host system it is still vital to ensure that the containers themselves are not compromised. This calls for means to

check the systems running inside the container for known vulnerabilities, an issue that is subject to heavy and fast-paced change, again requiring according evolution. A recent study[6] shows that in the case of Docker, depending on the version of the images, more than 70% of the images show potential vulnerabilities, with over 25% of them being severe. This also begs the question of who is responsible for checking and fixing these vulnerabilities, the operations team or the experts who created them? Despite these security and compliance constraints, the ever-changing smart city domain itself makes it necessary for experts to stay on top of the novel toolsets that emerge in order to handle requirements stemming from topics like Big Data or IoT. This leads to a rapid creation and adaptation of models and their according containers, which in turn need be checked against these constraints again. Last but not least, these containers need to comply to certain non-functional requirements that arise from the specific situations they are applied in. This calls for the ability to constantly check containers against certain runtime metrics that need to be met in order to ensure that these systems are able to deliver their excepted results within stakeholder-specific time and resource constraints.

All these factors lead to a complex environment that calls for an ability to easily adapt and evolve containers to their ever-changing requirements. Specifically, we identify the following requirements in the context of our domain:

- The ability to check a large amount of heterogenous containers against an open set of evolving requirements. These requirements can be vulnerabilities, compliance constraints, functional tests, or any other metric of interest for the domain.
- The ability to mitigate issues and evolve these containers based on the the results from the previously mentioned checks.
- An approach that is applicable in the context of operations management, while still enabling the participation of experts both for checking as well as evolution.
- An approach that can be applied to existing deployments as well as utilized to test new ones.

7.3 The Smart Brix Framework

In this section, we introduce the Smart Brix framework for continuos evolution of container-based deployments, which addresses the previously introduced requirements. We start with a framework overview, followed by a detailed description of all framework elements, and conclude with a comprehensive description of our proof of concept implementation including possible deployment variants.

7.3.1 Framework Rationales

The Smart Brix framework follows the microservice [78] architecture paradigm and an overview of the main framework components is shown in Figure 7.1. The framework is logically organized

[6] http://www.banyanops.com/blog/analyzing-docker-hub/

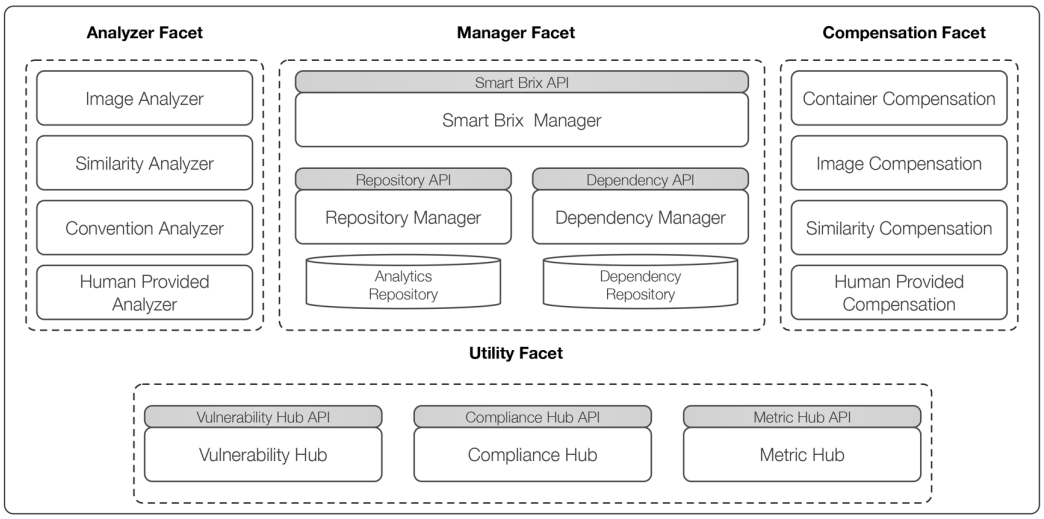


Figure 7.1: Smart Brix Framework Overview

into four main facets, which group areas of responsibility. Each of these facets is composed of multiple components where each of these components represents a microservice. The components in the *Analyzer* and *Compensation Facet* are managed as self-assembling components[7], an approach we presented in Section 4.3.1 and successfully applied in our Smart Fabric framework. Each of these components follows the *Command Pattern* [33] and consists of multiple *processors* that are able to accept multiple inputs and produce exactly one output. This functional approach enables a clean separation of concerns and allows us to decompose complex problems into manageable units.

Figure 7.2 illustrates an example of auto-assembly within the *Analyzer* facet. We see a set of *processors*, where each processor is waiting for a specific type of input and clearly specifies the output it produces. The processors use a message-oriented approach to exchange input and output data, where each output and input is persistently available in the message queue and accessible by any processor. In this example we perform an analysis of a custom-built Debian-based container that hosts the Apache HTTPD server. There are two potential processors for the input *Artifact*, each of them able to handle a different container format. Since in our example the Artifact is a *Docker Container*, only the *Docker Analyzer* reacts and produces as output a *Docker Image*. In the next step there are two active processors, the *Docker Base Image Analyzer* and the *Docker Package System Analyzer*, both taking Docker Images as input. Since the Docker Base Image Analyzer cannot determine a base image for the given Docker Image, it produces no output. However, the Docker Package System Analyzer is able to determine that the image uses a *DPKG*-based package system and produces the according output. Now the *DPKG Package Analyzer* reacts by taking two inputs, the original *Artifact* as well as the DPKG output and inspects the Artifact via the *DPKG command* to produce a *Package List*. In the last step of this

[7] http://techblog.netflix.com/2014/06/building-netflix-playback-with-self.html

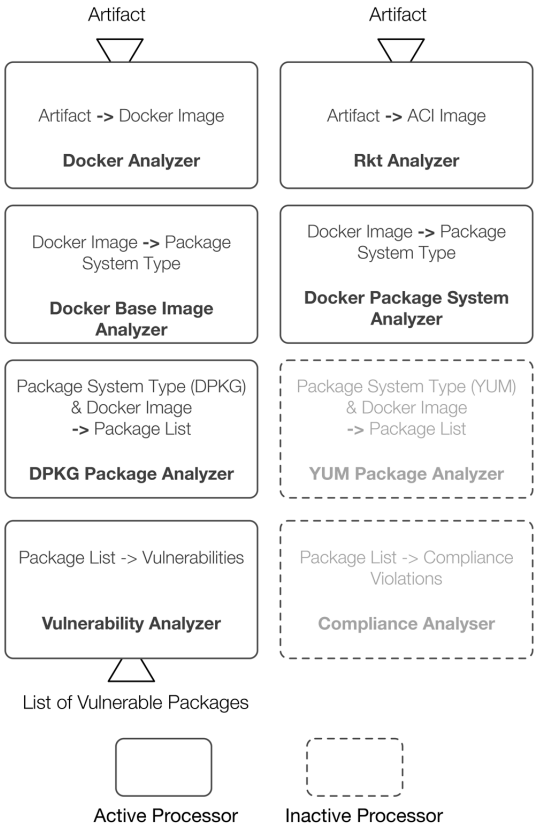


Figure 7.2: Example of auto assembling processors within the analyzer facet.

auto-assembly example the *Vulnerability Analyzer* listens for a Package List and produces a *List of Vulnerabilities*. This enables a straightforward auto-assembly approach, where connecting previous outputs to desired inputs leads to an automatically assembled complex system consisting of simple manageable processors. A processor itself can be anything and is not bound to any specific functionality, so it can be created completely flexible depending on the task at hand. This approach further eliminates the necessity of complex composition and organization mechanisms, enabling dynamic and elastic compositions of desired functionality, where processors can be added on demand at runtime. This enables the previously mentioned creation of open and flexible analytics and compensation pipelines based on this principle.

Additionally, the components in the analyzer and compensation facets follow the principle of *Confidence Elasticity*, which means that a component or processor produces a result that is augmented with a confidence value ($c \in \mathbb{R}, 0 \leq c \leq 1$), with 0 representing no certainty and 1 representing absolute certainty about the produced result. This allows for the specification of acceptable confidence intervals for the framework, which augment the auto-assembly mechanism. The confidence intervals are provided as optional configuration elements for the framework. In case the provided confidence thresholds are not met, the framework follows an escalation model to

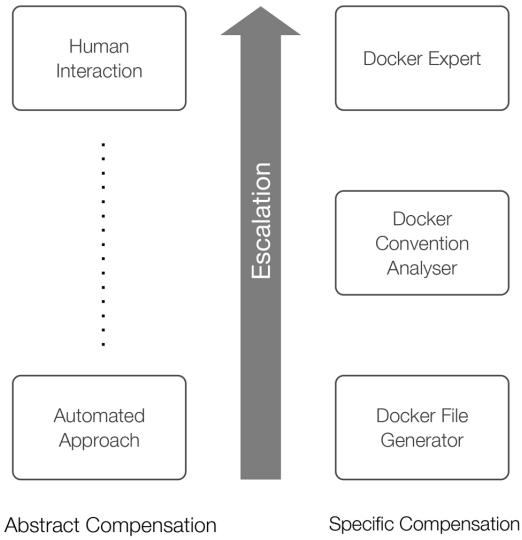


Figure 7.3: Confidence Adaptation Model Escalation

find the next component or processor that is able to provide results with higher confidence until it reaches the point where human interaction is necessary to produce a satisfactory result (illustrated in Figure 7.3). Each processor p_i from the set of active processors P_a provides a confidence value c_i. We define the overall confidence value of all active processors c_a as $c_a = \prod_{p_i \in P_a} c_i$. The compensation stops when c_a meets the specified confidence interval of the framework or a processor represents a human interaction which has a confidence value of ($c_i = 1$).

7.3.2 Smart Brix Manager

In order to initiate a container evolution, the *Smart Brix Manager* is invoked via the *Smart Brix API* with the following parameters: (i) a set of *Containers* to be inspected with (ii) the necessary *Credentials* to analyze and evolve them, as well as an optional (iii) set of *Artifacts* necessary to compensate or analyze the containers. In a first step the *Smart Brix Manager* queries the *Repository Manager* to see if there are already known issues for the supplied containers. If any known issues are found, the Smart Brix Manager creates a corresponding compensation topic via the messaging infrastructure by publishing the container identifiers as well as the found issues. This represents an input that will subsequently be consumed by the corresponding *Compensation Handlers* and starts the previously described auto-assembly process in the *Compensation Facet*.

If no issues were found, the Smart Brix Manager hands off the supplied *Containers*, *Credentials* and *Artifacts* to the *Dependency Manager* that is responsible for storing them in the *Dependency Repository*. As a next step, the Smart Brix Manager creates a corresponding analyzer topic via the messaging infrastructure and publishes the container identifiers to it. This generates an input that will be consumed by the corresponding *Analyzers* and starts another auto-assembly process in the *Analyzer Facet*. The Smart Brix Manager then listens to the created topic and waits for a response from the Analyzer Facet. If any analyzer responds, the manager

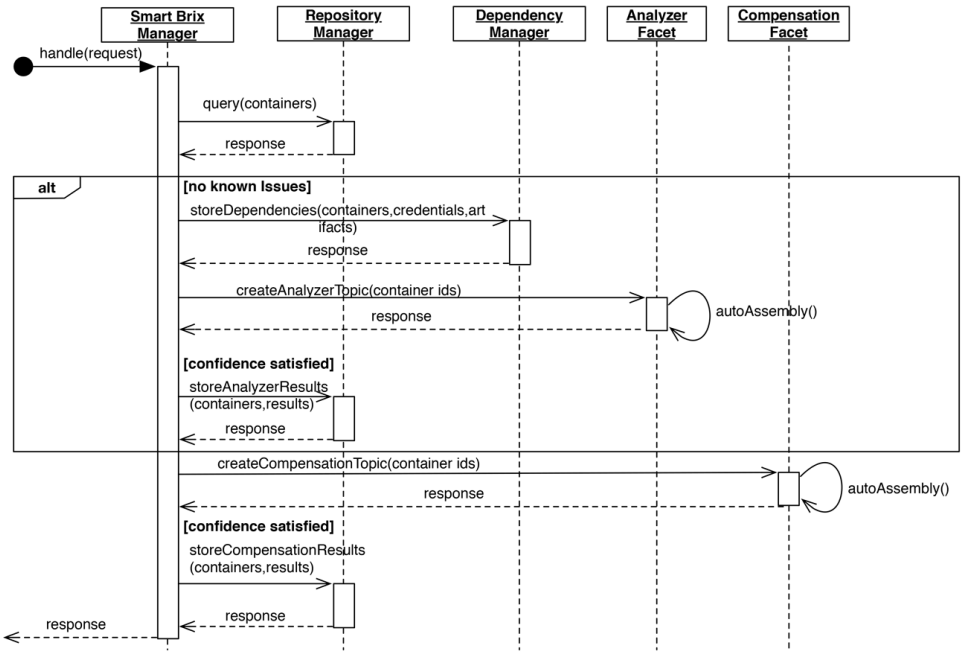


Figure 7.4: Smart Brix Manager Sequence Diagram

checks the confidence value of the provided results against the configured confidence interval of the framework. If the results satisfy the interval it uses the *Repository API* to store them in the *Analytics Repository*. If the confidence intervals are not satisfied, it waits for a configured timeout for additional results to emerge. If this fails the framework escalates according to the principle of *Confidence Elasticity* and marks the containers as required for human interaction. If the confidence interval was met, the Smart Brix Manager initiates the previously mentioned auto-assembly process in the Compensation Facet. The Smart Brix Manager then listens to the created topic and waits for a response from any compensation handler. In case of a response, it checks the confidence values by applying the same approach as for the Analyzer Facet, and stores them as compensations into the Analytics Repository. A corresponding sequence diagram illustrating this is shown in Figure 7.4.

Furthermore, the Smart Brix Manager provides API endpoints to query the results of analytics and compensation processes, as well as the current status via container identifiers.

7.3.3 Repository Manager

The *Repository Manager* provides a repository for storing analytics results of all analyzed containers as well as their corresponding compensations. The *Analytics Repository* itself is a distributed key value store that enables Analyzers as well as Compensation Handlers to store information without being bound to a fixed schema. In addition, this enables the previously mentioned open extensibility of our auto-assembly approach by allowing every component to choose the required

storage format. Finally, the Repository Manager provides a service interface to store and retrieve analytics and compensation information as well as an interface for querying information based on container identifiers or other attributes.

7.3.4 Dependency Manager

The *Dependency Manager* handles necessary credentials and artifacts that are needed for processing containers. The Dependency Manager provides a service interface that allows the Smart Brix Manager to store artifacts and credentials associated with specific containers. Additionally, it provides a mechanism for components in the Analyzer and Compensation Facets to retrieve the necessary credentials and artifacts for the corresponding container IDs. Finally, it acts as service registry for components in the *Utility Facet* and exposes them to the Compensation and Analyzer Facet. The Dependency Manager uses a distributed key value store for its *Dependency Repository* in order to store the necessary information.

7.3.5 Utility Facet

The general role of the *Utility Facet* is to provide supporting services for Analyzers, Compensation Handlers, and Managers of the framework. Components in the Utility Facet register their offered services via the Dependency Manager. This provides an open and extensible approach that allows to incorporate novel elements in order to address changing requirements of container evolution. In our current architecture, the Utility Facet contains three components. First, a *Vulnerability Hub*, which represents a service interface that allows Analyzers as well as Compensation Handlers to check artifacts for vulnerabilities. The Vulnerability Hub can either utilize public repositories (e.g., the National Vulnerability Database[8]), or any other open or proprietary vulnerability repository. The second component is a *Compliance Hub* that allows to check for any compliance violations in the same way the Vulnerability Hub does. This is an important element in heterogenous multi-stakeholder environments, where compliance to all specified criteria must be ensured at all times. The last element is a *Metric Hub*, which allows to check artifacts for certain relevant metrics in order to ensure relevant Quality of Service constraints for containers.

7.3.6 Analyzers

The task of the components within the *Analyzer Facet* is to test containers for potential vulnerabilities, compliance violations or any other metrics. The facet is invoked by the *Smart Brix Manager*, which triggers an auto-assembly process for the given containers that should be analyzed. The Analyzer Facet can contain components for the most prominent container formats like Docker or Rkt, but due to the fact that we utilize the auto-assembly approach, we are able to integrate new container formats as they emerge. For analyzing a container an analyzer follows three basic steps: (i) Determine the base layer of the container in order to know how to access the package list. (ii) Determine the list of installed packages including their current version. (iii) Match the list of installed packages against a set of vulnerabilities, issues, or compliance constraints in order to determine the set of problems.

[8] https://nvd.nist.gov/

Every step can follow a different set of strategies to analyze a container represented as different processors, each of them with a specific confidence value. Possible processors for these steps are: (i) Base Image Processors, which try to determine the base layer of a container by matching their history against known base image IDs. (ii) Similarity Processors that try to select a base layer based on similarities in the history of the container with known containers by performing actions like collaborative filtering and text mining. (iii) Convention Processors that try to determine the base layer by trying common commands and checking their results. (iv) Human Provided Processors, which are human experts that manually analyze a container.

In order to access the containers and to perform analytics, the components within the Analyzer Facet interact with the Dependency Manager. The manager provides them with the necessary credentials for processing containers. Once the analyzers have processed a container, they publish the results, which are augmented with the confidence value, to the corresponding topic where the Smart Brix Manager carries on as previously described.

7.3.7 Compensation Handlers

The components in the *Compensation Facet* generate potential compensations for containers that have been previously identified by the Analyzers. Like the Analyzers, the *Compensation Handlers* are invoked by the Smart Brix Manager, which starts an auto-assembly process for the containers with problems that should be compensated. We provide components for the most prominent container formats, with the ability to extend the list as new formats emerge. The compensation handlers follow three basic steps: (i) Apply a compensation strategy for the container and the identified problem; (ii) Verify if the compensation strategy could be applied by rebuilding or restarting the container; (iii) Verify that the problems could be eliminated or reduced.

Again, every step can utilize a set of different processors, each of them with a specific confidence value, which represent different strategies. Possible processors are: (i) Container Processors, which try to use the base image's package manager to upgrade packages with identified vulnerabilities. (ii) Image Processors that try to build a new image without the vulnerabilities; (iii) Similarity Processor that try to compensate via applying steps from similar containers that do not show these vulnerabilities; (iv) Human Provided Processors, which are human experts that manually compensate a container.

The Compensation Handlers interact with the Dependency Manager in a similar way like the Analyzers to retrieve the necessary credentials to operate. As Image Processors and Similarity Processors build new images in order to compensate, they can request the necessary artifacts associated with an image to be able build them.

7.3.8 Implementation

We created a proof of concept prototype of our framework based on a set of RESTful microservices implemented in Ruby. Each component that exposes a service interface relies on the Sinatra[9] web framework. The *Repository Manager* and the *Dependency Manager* utilize MongoDB[10] as

[9] http://www.sinatrarb.com/

[10] https://www.mongodb.org/

their storage backend, which enables the previously described distributed, open, and extendable key value store for their repositories. We implemented a *Vulnerability Hub* that uses a SQLite[11] storage backend to persist vulnerabilities in a structured format. It holds the recent data from the National Vulnerability Database[12] (NVD), specifically the listed Common Vulnerabilities and Exposures (CVEs). This CVE Hub allows to import the CVEs posted on NVD, stores them in its repository, and allows to search for CVEs by vulnerable software name as well as version via its Sinatra-based REST interface.

To enable the auto-assembly mechanism for each processor within each component in the *Analyzer* and *Compensation Facet*, we use a message-oriented middleware. Specifically, we utilize RabbitMQ's[13] topic and RPC concepts, by publishing each output and listening for its potential inputs on dedicated topics. We implemented a Docker Analyzer component with a *Base Image Processor* and a *Convention Processor*-based strategy. The Docker Analyzer first tries to determine the operating system distribution of the container by analyzing its history. Specifically, it uses the Docker API to generate the history for the container and selects the first layer's ID, which represents the base layer. It then matches this layer against a set of known layer IDs, which matches corresponding operating system distributions to determine which command to use for extracting the package list. If a match is found, it uses the corresponding commands to determine the package list. If the determined operating system is Ubuntu or Debian, it will use `dpkg` to determine the package list. If it was CentOS, `yum` is used, and if it was Alpine, `apk`. After parsing the package command output into a processable list of packages, it checks each package name and version by using the *CVE Hub* via its REST interface. When this step is finished the Analyzer publishes the list of possible vulnerabilities, including analyzed packages along with several runtime metrics. In case the base image strategy fails, the Docker Analyzer tries to determine the base layer including the corresponding operating system via a convention processor. Specifically, it test if the image contains any of the known package managers. Based on the results the analyzer determines the distribution flavor and continues as described above.

We further implemented a Docker Compensation Handler with a *Container Processor* and an *Image Processor* based compensation strategy. The Container Processor tries to upgrade the container using the operating system distribution's package manager. After this operation succeeds, it checks if the number of vulnerabilities are reduced, by comparing the new version of packages against the *CVE Hub*. If this was the case it augments the results with a confidence value based on the percentage of fixed vulnerabilities and publishes the results. The Image Processor tries to fix the container by generating a new container manifest (e.g., Dockerfile). More precisely, it uses the Docker API to generate the image history and then derives a Dockerfile from this history. After this step, the Image Processor exchanges the first layer of the Dockerfile with the newest version of its base image. In cases where it cannot uniquely identify the correct Linux flavor, it generates multiple Dockerfiles, for example one for Ubuntu and one for Debian. It then checks the Dockerfiles' structure for potential external artifacts. Specifically, it searches for any *COPY* or *ADD* commands that are present in the Dockerfile. If this is the case, it contacts the Dependency Manager and attempts to retrieve the missing artifacts. Once this is finished the

[11] https://www.sqlite.org/

[12] https://nvd.nist.gov/

[13] https://www.rabbitmq.com/

Image Processor tries to rebuild the image based on the generated Dockerfile. After this step is finished, the Image Processor again checks the new list of packages against the CVE Hub, and if it could improve the state of the image it publishes the results with the corresponding confidence value.

7.3.9 Deployment Modes

The Smart Brix Framework provides a container for each facet and therefore supports deployment on heterogeneous infrastructures. The framework enables wiring of components and aspects via setting the container's environment variables, enabling dynamic setups. We distinguish between two fundamental deployment modes, *Inspection Mode* and *Introspection Mode*.

Inspection Mode

The *Inspection Mode* allows the framework to run in a dedicated inspection and compensation setting. In this mode the framework ideally runs exclusively without any other containers and utilizes the full potential of the host systems. This means that the Smart Brix Managers wait until they receive an explicit request to analyze and compensate an artifact.

Introspection Mode

The *Introspection Mode* allows the framework to run in an active container setup. In this mode the framework constantly watches deployed containers via the Smart Brix Manager. The Manager can be provided with a list of containers to watch via a configuration setting. This provided list of containers is then analyzed and compensated. If no container lists are supplied, the Manager watches all running containers on the platform. In this case it initiates a check whenever new images are added, an image of a running container changes, or new vulnerabilities are listed in the CVE Hub.

7.4 Evaluation

7.4.1 Setup

For our evaluation we used the following setup. We provisioned three instances in our private OpenStack cloud, each with 7.5GB of RAM and 4 virtual CPUs. Each of these instances was running Ubuntu 14.04 LTS with Docker staged via docker-machine[14]. For our evaluation we choose the *inspection deployment* variant of our framework in order to stress-test the system without other interfering containers. We deployed one manager container representing the *Management Facet*, as well as two utility containers containing the *CVE Hub* and the *Messaging Infrastructure* on one instance. We then distributed 12 analyzer containers with 12 compensation containers over the remaining two instances. Additionally, we deployed a cAdvisor[15] container on every

[14] https://docs.docker.com/machine/install-machine/
[15] https://github.com/google/cadvisor

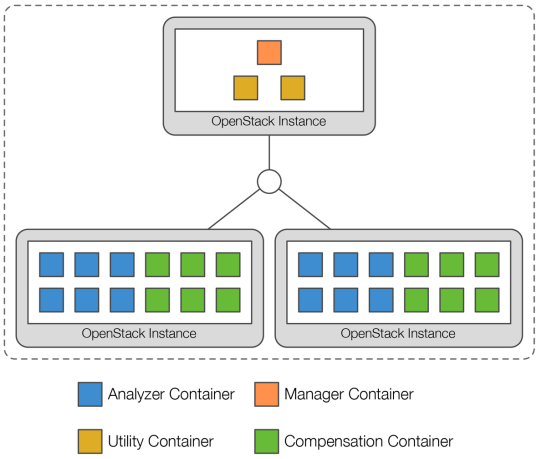


Figure 7.5: Evaluation Setup of Smart Brix running in inspection mode

instance to monitor the resource usage and performance characteristics of the running containers. Figure 7.5 shows an overview of the deployed evaluation setup.

7.4.2 Experiments

Since we currently only have around 250 images in our URBEM setting, we extended the number of images to be evaluated. In order to get a representative set of heterogenous images we implemented a small service to crawl Docker Hub[16]. The Docker Hub is a public repository of Docker container images of different flavors. These images range from base images, like Ubuntu and CentOS etc., to more complex images like Cassandra and Apache Spark. We utilized the search function of the Hub to collect a set of 4000 images ordered by their popularity (number of pulls and number of stars), which ensures that we focus on a set with a certain impact. We then extracted the name and the corresponding pull commands along with the latest tag to form the URI of the image. This set of 4000 URIs represented the source for our experiments, which was then split into 3 sets containing 250, 500, and 1000 images to be tested.

Analyzer Experiments

We started our experiments with a focus on the Analyzer Facet of the framework. First, we started the analyzer containers on one instance and started our tests with the 250 image set. After the run finished we repeated it with the 500 and 1000 image set. After the tests with one instance, we repeated the experiments with two instances where each run was repeated 3 times. During the tests we constantly monitored cAdvisor to ensure that the instances were not fully utilized in order to ensure this would not skew results. The focus of our experiments were not the performance characteristics of our framework, in terms of cpu, memory or disk usage, which is why we used cAdvisor only as a monitor to rule out overloading our infrastructure. We also did not utilize any

[16]https://hub.docker.com/

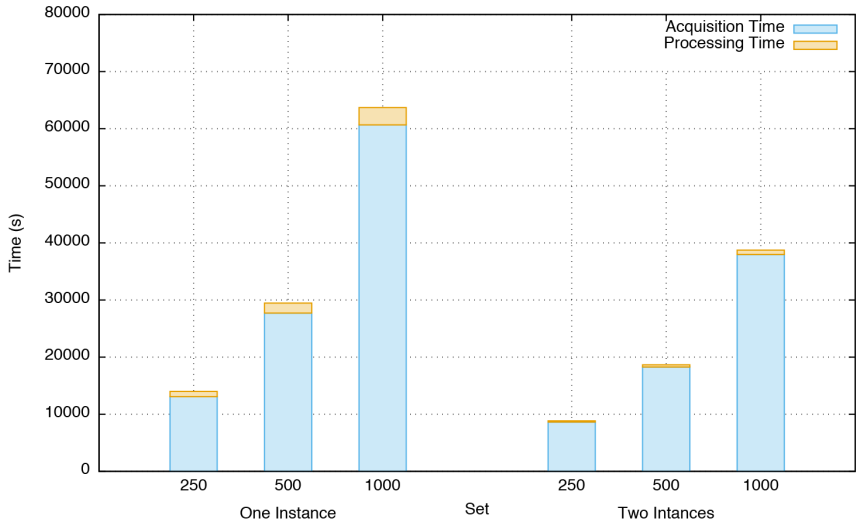


Figure 7.6: Comparison of runtime for analytics between one instance and two instances

storage backend for cAdvisor since this has shown to be a significant overhead which in turn would have skewed our results.

After the runs had finished we evaluated the vulnerability results. The analyzers logged the analyzed images, their base image flavor (e.g. Ubuntu, Debian etc.), processing time to analyze the image, pull time to get the image from the DockerHub as well as the overall runtime, number of packages, size of the image, and number of vulnerabilities.

Over all our experiments the analyzers showed that around 93% of the analyzed images have vulnerabilities. This mainly stems from the fact that our implemented analyzers have a very high sensitivity and check for any potentially vulnerable software with any potentially vulnerable configuration. However, this does not necessarily mean that the specific combination of software and configuration in place shows the detected vulnerability. If we only take a look at the images with a high severity according to their CVSS[17] score, around 40% show to be affected which is conclusive with recent findings[18]. These results underline the importance to implement the measures proposed by our framework. However, the focus of our work and the aim of our experiments was not to demonstrate the accuracy of the implemented vulnerability detection, but the overall characteristics of our framework, which we discuss in the remainder of this section.

We first compared the overall runtime of our analyzers, specifically the difference for one instance vs two instance deployments, the results are shown in Figure 7.6. Based on the results we see that our approach can be horizontally scaled over two nodes leading to a performance improvement of around 40%. The fact that in our current evaluation setting we were not able to halve the overall runtime using two instances stems from several factors. On the one hand, we have a certain overhead in terms of management and coordination including the fact that we only deployed one manager and storage asset. On the other hand, a lot of the runtime is caused by

[17] https://nvd.nist.gov/cvss.cfm

[18] http://www.banyanops.com/blog/analyzing-docker-hub/

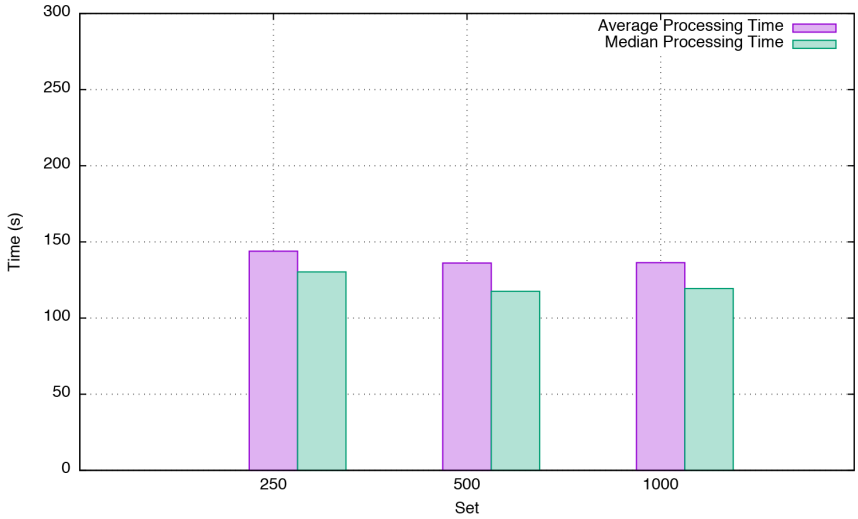


Figure 7.7: Comparison of processing time for analytics with two instances

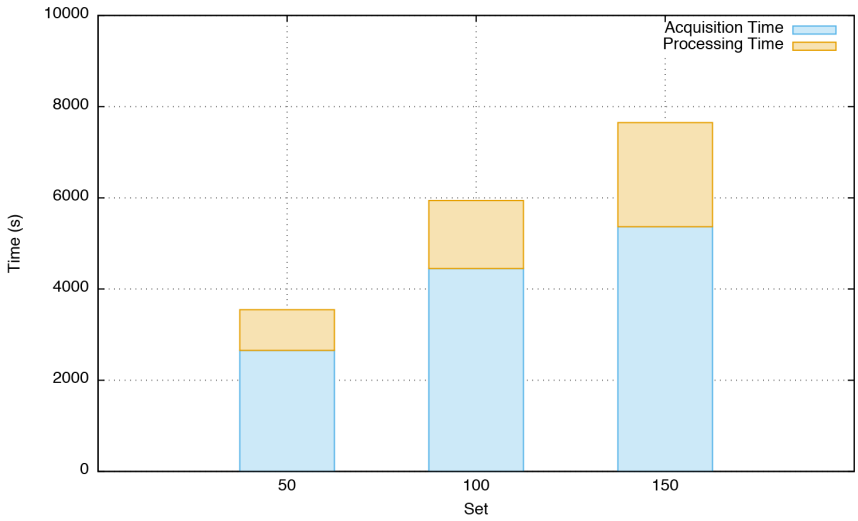


Figure 7.8: Comparison of pulltime and processing time for compensation with two instances

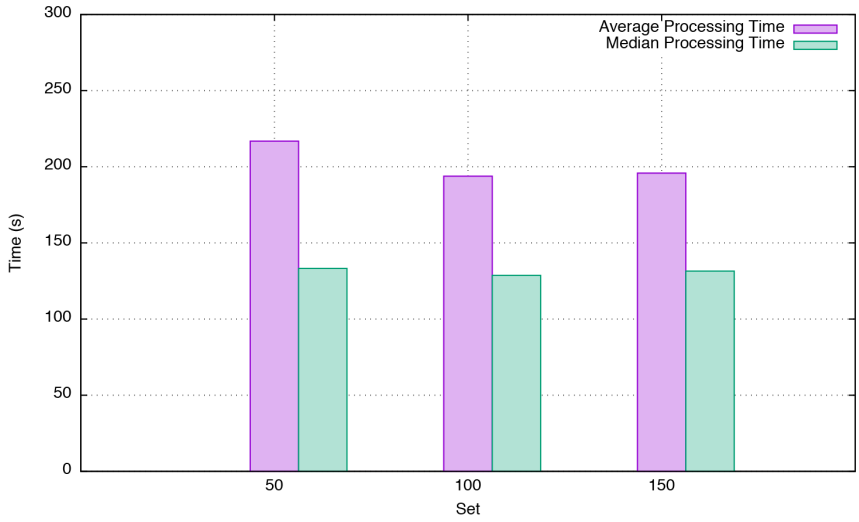


Figure 7.9: Comparison of processing time for compensation with two instances

the acquisition time, which is clearly bound by network and bandwidth. Since our infrastructure is equipped with just one 100 Mbit uplink that is shared by all cloud resources, this is a clear bottleneck. We also see that the majority of wall clock time is spent for acquisition and that the actual processing time only amounts to approximately 3% of the overall runtime. The fact that the acquisition time for the 1000 image set does not grow linearly like the runs with the 250 and 500 image set, stems from Docker's image layer cache. In this case the overall acquisition time grows slower, because a lot of images in the 1000 set share several layers, which, if already pulled by another analyzer in a previous run, do not need to be pulled again, hence reducing the acquisition time. Finally, we demonstrate that the average processing time of our framework is stable, which is shown in Figure 7.7. We further notice a small increase in average processing time for the 250 image set, which is caused by the fact that this set contains more images with larger package numbers compared to the overall amount of images tested, resulting in a slightly higher average processing time. As illustrated in Table 7.1, per-package processing times remain stable throughout the performed experiments, with a median of 0.558s and a standard deviation of 0.257s.

Set	**Median Processing Time**	**Standard Deviation Processing Time**	**No. of packages**
250	0.620s	0.255s	153,275
500	0.564s	0.263s	303,483
1000	0.537s	0.252s	606,721
Overall	**0.558s**	**0.257s**	**1,063,479**

Table 7.1: Median and standard deviation for processing time per package over all runs with two instances

Compensation Experiments

In the the next part of our experiments we focused on the Compensation Facet of our framework. In order to test the ability to automatically handle compensations of vulnerable images, we tested the implemented Container Processor strategy. This strategy compensates found vulnerabilities via automatic upgrades of existing images. It takes no human intervention, has a very high confidence, keeps all artifacts within the images and is therefore optimal to test the auto-compensation ability of our framework. In the process of compensation the Container Processor generates a new image with the upgraded packages. In order to test this image for improvement we have to store it. This means that for every tested image we have to hold the original image as well as its compensated version. Specifically, we choose to test the most vulnerable images (images with the most vulnerable packages) out of the 1000 image set we tested that are also the most prominent images in our URBEM scenario. This left us with 150 images, which we split in three sets with 50, 100, and 150 images and started our compensation tests. We then repeated each run to demonstrate repeatability and to balance our results. Since the Compensation Facet follows the same principle as the Analyzer Facet we omitted testing it on one instance and immediately started with two instances. After the tests finished, we compared the newly created images to the original ones and checked if the number of vulnerabilities could be reduced.

Overall our experiments showed that from the 150 images we were able to auto-compensate 34 images by reducing the number of vulnerabilities. This illustrates that even a rather simple strategy leads to a significant improvement of around 22,6%, which makes this a very promising approach. In a next step, we compared the overall runtime of our compensation handlers for the three tested sets, and the results are shown in Figure 7.8. We again can clearly see that the major amount of time is spent for acquisition, in this case pulling the images that need to be compensated. The compensation itself only takes between 24% and 28% of the overall runtime and shows linear characteristics correlating with the number of images to be compensated. The comparatively low increase in acquisition time for the 150 image set again can be explained with the specific characteristics we see in Docker's layer handling.

In a next step, we compared the average processing time for each set, and the results are shown in Figure 7.9. We again notice similar characteristics as we saw with our analyzers. The average processing time as well as the median processing time are stable. The small increase for the 50 image set is explained with a larger number of images that contain more packages. This fact leads to relatively longer compensation times when upgrading them.

7.5 Discussion

Our experiments showed that our framework is able to scale horizontally. We further demonstrated that the majority of the runtime, both when analyzing and compensating images is caused by the image acquisition, which is bandwidth bound. Given the fact that in most application scenarios of our framework the images will not necessarily reside on Docker Hub, but instead in a local registry, this factor greatly relativizes. The processing time itself scales linearly with the number of analyzed packages, and the same was shown for the compensation approach. Furthermore, the processing time in our current evaluation setup is mostly constrained by the

prototypical vulnerability checking mechanism and the chosen storage system, which both are not the focus of our contribution. The implementation of different vulnerability checkers, along with more efficient storage and caching of vulnerability data could lead to further reduction in processing time and will be tackled in future work. An additional aspect we did not specifically address in this chapter is the fine-grained scale-out of components in all Smart Brix facets. While the presented framework fulfills the requirements set forth in the previously introduced URBEM project, certain threats to the general applicability of Smart Brix remain. Currently, the auto-assembly mechanism introduced in Section 5.4.1 attempts to eagerly construct analysis and compensation pipelines that are loosely structured along the level of specificity of the performed analysis. Hence, the number of created pipelines can grow exponentially with the number of candidate components in the worst case. If all components for a given level of specificity accept all inputs produced in the previous level, and all subsequent components accept all produced outputs in turn, the number of created pipelines would grow exponentially with the number of components per level of specificity. This problem can be mitigated by introducing a transparent consolidation mechanism that delays the propagation of produced outputs of a certain type for a specified amount of time, orders them by the reported confidence values, and only submits one (or a few) of the produced output values with the highest confidence values for further consumption by other components.

7.6 Related Work

The rapid adoption of container-based execution environments for modern applications enables increased flexibility and fast-paced evolution. Next to this fast-paced evolution of containers, new containers are deployed whenever functionality has to be added, which leads to massive amounts of containers that need to be maintained. While the container provides an abstraction on top of the operating system, it is still vital that the underlying system complies to policies or regulations to avoid vulnerabilities. However, checking the plethora of available environments and adapting them accordingly, is not a trivial task.

Among several approaches stemming from the area of SOA like the works of Lowis et al. [66], Yu et al. [120] which deal with classic service vulnerabilities as well as the work of Li et al. [60], Lowis et al. [65] propose a novel method for analyzing cloud-based services for certain types of vulnerabilities. Next to general models and methods for classifying and analyzing applications, several approaches emerged that allow vulnerability testing. They range from service oriented approaches for penetration and automated black box testing introduced by Bau et al. [12] and Li et al. [61] to model based vulnerability testing like the work of [56] as well as automated vulnerability and infrastructure testing methods (e.g. [43, 97]). Antunes and Vieira [7] introduce SOA-Scanner, an extensible tool for testing service-based environments for vulnerabilities. Based on an iterative approach the tool discovers and monitors existing resources, and automatically applies specific testing approaches. More recently also large scale distributed vulnerability testing approaches have been introduced (e.g. [29, 121]). In contrast to our approach, the aforementioned tools solely concentrate on testing and identifying possible security threats, but do not provide means for adapting the observed application or its environment accordingly.

More recently, container-based approaches are applied in the literature to ease develop-

ment and operation of applications. Tosatto et al. [104] analyze different cloud orchestration approaches based on containers, discuss ongoing research efforts as well as existing solutions. Furthermore, the authors present a broad variety of challenges and issues that emerge in this context. Wettinger et al. [118] present an approach that facilitates container virtualization in order to provide an alternative deployment automation mechanism to convergent approaches that are based on idempotent scripts. By applying action-level compensations, implemented as fine-grained snapshots in the form of containers, the authors showed that this approach is more efficient, more robust, and easier to implement as convergent approaches. However, compared to our approach, the authors do not provide a framework for analyzing container application deployments, which based on identified issues triggers according compensation mechanisms. Gerlach et al. [35] introduce Skyport, a container-based execution environment for multi-cloud scientific workflows. By employing Docker containers, Skyport is able to address software deployment challenges and deficiencies in resource utilization, which are inherent to existing platforms for executing scientific workflows. In order to show the feasibility of their approach, the authors add Skyport as an extension to an existing platform, and were able to reduce the complexities that arise when providing a suitable execution environment for scientific workflows. In contrast to our approach the authors solely focus on introducing a flexible execution environment, but do not provide a mechanism for continuously evolving container-based deployments. Li et al. [62] present an approach that leverages Linux containers for achieving high availability of cloud applications. The authors present a middleware that is comprised of agents to enable high availability of Linux containers. In addition, application components are encapsulated inside containers, which makes the deployment of components transparent to the application. This allows monitoring and adapting components deployed in containers without modifying the application itself. Although this work shares similarities with our approach, the authors do not provide a framework for testing container-based deployments, which also supports semi-automatic compensation of found issues.

Next to scientific approaches, also several industrial platforms emerged that deal with the development and management of container-based applications, with the most prominent being Tutum[19] and Tectonic[20]. These cloud-based platforms allow building, deploying and managing dockerized applications. They are specifically built to make it easy for users to develop and operate the full spectrum of applications, reaching from single container apps, up to distributed microservices stacks. Furthermore, these platforms allow keeping applications secure and up to date, by providing easy patching mechanisms and holistic systems views. In contrast to our approach, these platforms only focus on one specific container technology, and are not extensible. IBM recently introduced the IBM Vulnerability Advisor[21], a tool for discovering possible vulnerabilities and compliance policy problems in IBM containers. While IBM's approach shares similarities with our work, they are solely focusing on Docker containers that are hosted inside their own Bluemix environment and therefore do not provide a generic approach. Furthermore, their Vulnerability Advisor only provides guidance on how to improve the security of images, but does not support mechanisms to evolve containers.

[19] https://www.tutum.co

[20] https://tectonic.com

[21] https://developer.ibm.com/bluemix/2015/07/02/vulnerability-advisor/

7.7 Summary

The numerous benefits of container-based solutions have led to a rapid adoption of this paradigm in recent years. The ability to package application components into self-contained artifacts has brought substantial flexibility to developers and operation teams alike. However, to enable this flexibility, practitioners need to respect numerous dynamic security and compliance constraints, as well as manage the rapidly growing number of container images. In order to stay on top of this complexity it is essential to provide means to evolve these containers accordingly. In this chapter we presented *Smart Brix*, a framework enabling continuous evolution of container application deployments. We provided a comprehensive description of URBEM's requirements in terms of container evolution. We introduced Smart Brix to address these requirements, described its architecture, and the proof of concept implementation. Smart Brix supports both, traditional continuous integration processes such as integration tests, as well as custom, business-relevant processes, e.g., to implement security, compliance, or other regulatory checks. Furthermore, Smart Brix not only enables the initial management of application container deployments, but is also designed to continuously monitor the complete application deployment topology and allows for timely reaction to changes (e.g., discovered application vulnerabilities). This is achieved using analytics and compensation pipelines that will autonomously detect and mitigate problems if possible, but are also designed with an escalation mechanism that will eventually request human intervention if automated implementation of a change is not possible. We evaluated our framework using a representative case study that clearly showed that the framework is feasible and that we could provide an effective and efficient approach for container evolution.

CHAPTER 8

A Smart-City Application Ecosystem

In this chapter we present SCALE, the Smart City Application Ecosystem. We outline the basic architecture of SCALE, introduce the Smart City Operating System (SCOS), a core element in enabling SCALE and detail how the previously presented contributions enable the SCOS.

8.1 Introduction

Based on our previous and ongoing research projects, industry collaborations and smart city initiatives (e.g., URBEM) as well as the contributions presented in this thesis, we present a high-level architecture of SCALE, a comprehensive smart city application ecosystem. SCALE is based on a Smart City Operating System (SCOS) that addresses the requirements presented in Section 1.1. Naturally, smart city applications reside in a dynamic environment serving multiple stakeholders. Stakeholders do not only provide data for an application, but contribute functionality, or impose (possibly conflicting) requirements. So far, the fundamental stakeholders in a smart city are energy and transportation providers, as well as government agencies, which offer data about certain aspects (e.g., public transportation) of a city and its citizens. Traditional smart city applications are usually commissioned, operated by, and focused on single providers and their requirements [59], leading to isolated vertical applications, resulting in a significant fragmentation of smart city applications.

8.2 Smart City Application System Architecture

By introducing SCALE we aim to break this traditional notion of industry verticals and according infrastructure silos to closely integrate requirements, functionality, and capabilities of multiple stakeholders in a seamless and manageable way. First, we strive to align smart city application engineering along the three pillars of methodology, modeling, and middleware, as presented in [89]. Second, a comprehensive modeling framework will allow stakeholders to express distinct system capabilities in terms of declarative expectations and properties that allow for realistic simulations

of real-world infrastructures, reducing the risks of developing against real systems, while at the same time increasing developer productivity. The simulation of real-world infrastructures will allow developers to plan ahead of time to develop and test applications for emerging, not yet available, infrastructures. Third, SCALE will serve as a comprehensive communication, integration, and engineering tool throughout the complete application lifecycle, from requirements elicitation, definition, and specification, to development, deployment, and management of application functionality. Finally, by focusing on distinct, reusable units of functionality, SCALE allows for the creation of application components that can easily be composed, coordinated, and tested using a dynamic combination of simulated and real-world infrastructure. To achieve this, we will provide a comprehensive middleware toolkit for engineering, executing, managing, and evolving smart city applications. This middleware toolkit will act as a Smart City Operating System, as depicted in Figure 8.1.

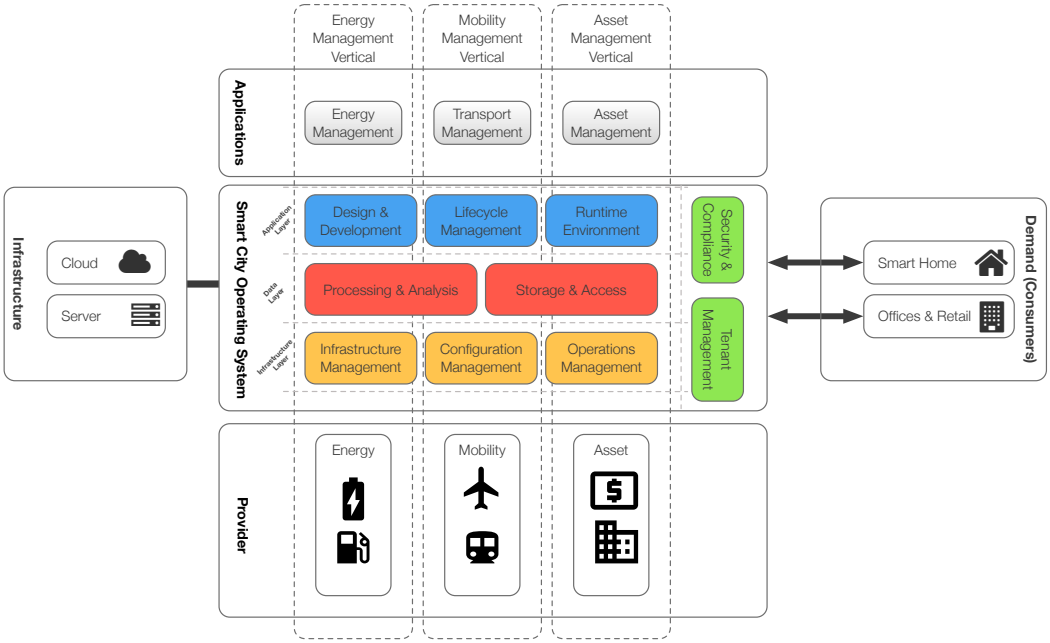


Figure 8.1: Smart City Application Ecosystem – Architecture

8.3 Smart City Operating System (SCOS)

In order to build SCOS on top of a future-proof architectural principle that allows for clean separation of concerns, easy extendability and high scalability, we chose a microservice architecture. This architectural principle allows us to break out of traditional layered architectures, which gives us the opportunity that each component of SCOS can interact with any other, leading to novel ways of synergies between SCOS components. By enabling this flexible interaction model we allow every component in SOS to benefit from any other component, leading to an evolvable and

extensible system. In the following, we introduce the main components of SCOS and discuss them in more detail.

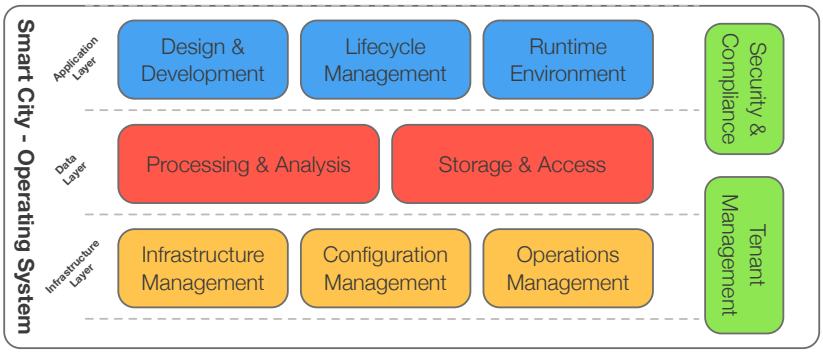


Figure 8.2: Smart City Operating System

8.3.1 Infrastructure Layer

The first and also bottom layer of SCOS is the *Infrastructure Layer*. Essentially, this layer manages the underlying infrastructure resources, configures and provisions them, and constantly monitors these resources.

Infrastructure Management

Since in a smart city we have various heterogeneous types of infrastructure resources like traditional servers, cloud computing resources [10], edge and emerging IoT devices [25], the infrastructure layer allows to integrate them using an *Infrastructure Management* subsystem. First, the infrastructure management provides a mechanism that enables stakeholders of SCOS to locate and identify their owned or leased resources. Second, after locating the necessary resources that should be managed, these resources need to be accessible for SCOS in order to get integrated into the overall system. Therefore, the infrastructure management provides a pre-built list of drivers for communicating with the resources, as well as an access management component for securely storing the necessary credentials or keys. Finally, to provide an extensible approach, the infrastructure management subsystem provides APIs that enable stakeholders to build custom drivers in order to integrate emerging types of resources that require new forms of interactions.

Configuration Management

Once the infrastructure resources are integrated into and are accessible for SCOS, it is necessary to facilitate the processing power of these resources for e.g., running and executing applications. In order to do this, the *Configuration Management* subsystem allows for provisioning, deploying, and configuring these resources transparently and efficiently. Since the various types of infrastructure resources have different capabilities and environments, configuration management needs to

respect these constraints. Hence, it provides a scalable and elastic provisioning solution that can be specifically tailored for each type of infrastructure [111]. The overall provisioning approach installs platform-specific software packages that allow leasing and releasing, monitoring, and deploying of resources in a generic and uniform manner. Next, for tailoring the connected and provisioned infrastructure to stakeholder requirements, the configuration management subsystem provides interfaces that enable SCOS to configure several aspects (e.g., pull- vs. push-based updates) via the provisioned software packages. Finally, the configuration management supports the seamless deployment of single applications, complete application topologies, and additional necessary packages onto connected infrastructure resources by considering both the computational capabilities as well as available execution environments [109]. Furthermore, since smart city applications need the ability to evolve over time in order to react to changing requirements or regulations, configuration management enables the seamless migration of application topologies among deployment targets as demonstrated in Chapter 4. Thus, allowing the independent evolution of applications, their topologies, and infrastructure resources.

Operations Management

After the infrastructure is provisioned and ready for deployment, SCOS needs a mechanism that enables monitoring and analyzing the performance of these resources. Thus, *Operations Management* supports the constant monitoring and collection of information from connected resources, by using available monitoring capabilities of the respective infrastructure (e.g., cloud monitoring APIs[1], commonly applied monitoring tools like Ganglia [68]), or by provisioning software capabilities that allow gathering performance measurements (e.g., tailored profilers for edge devices [111]). In addition to monitoring, operations management also provides mechanisms to manage gathered logs, events, and faults. Based on collected information from the underlying infrastructure resources, operations management is able to conduct performance analyses that can be used for optimizing resource utilization or evolve the overall infrastructure deployment. Furthermore, fine-grained analysis information can be used by other subsystems of SCOS to adapt application topologies in order to react to defined requirements like SLAs. Additionally it provides mechanisms for continuous evolution of applications and their components via the approach presented in Chapter 7. Finally, operations management provides APIs that allow operators of SCOS to define custom adaptation routines (e.g., scaling algorithms) to guarantee a defined availability for infrastructure resources or deal with network outages [45].

8.3.2 Data Layer

The second and middle layer of SCOS is the *Data Layer*, which is responsible for on the one hand storing and providing access for data that is residing in our ecosystem, and on the other hand processing and analyzing data that can be further used by other layers of SCOS in order to generate valuable insights.

[1] e.g., https://cloud.google.com/monitoring/api/

Storage & Access

Since in modern smart cities running applications, infrastructure resources, and citizens produce an ever growing amount of data, which is commonly referred to as big data [14], the data layer provides the *Storage & Access* subsystem for managing and handling data. It allows stakeholders of SCOS to store and consume large sets of diverse data by providing generic and extendable APIs. For efficiently managing data in SCOS, the subsystem allows considering the plethora of available data formats, the intrinsic diversity of data, and also enables respecting potentially noisy data [103] that is produced by the underlying infrastructure with its millions of managed resources. Additionally, the subsystem supports the ability for handling both, static data that is not frequently accessed or processed, as well as dynamic data that is constantly and relentlessly changing. To address the challenges that emerge from handling these different types of data, the storage & access subsystem provides a flexible approach that supports various storage facilities like traditional relational databases, document-oriented, and complex unstructured data stores. Providing data storages in a Data as a Service (DaaS) fashion enables the seamless integration of data facilities into SCOS, eases the integration of new data storages, and also allows for easy and uniform data access as well as storage functionality for components inside and applications on top of SCOS. Furthermore, the storage & access subsystem provides mechanisms for merging and combining different types of data, which can be used as foundation for analysis and planning operations in upper layers. Finally, since the ownership of data is an important concern in SCOS, the storage & access subsystem incorporates novel concepts that protect data, but also allow open data exchange where different levels of data owners can share data, by integrating the principle of a Hub of all Things[2] as well as the approach presented in Chapter 5. Following this approach enables the clear concept of ownership and allows addressing emerging data compliance and security requirements.

Processing & Analysis

After enabling the efficient access and storage of data in the data layer, the *Processing & Analysis* subsystem allows stakeholders to efficiently transform, mediate, and analyze these large sets of diverse data. In addition, since the processing of streaming data as well as historical data is an important aspect in current smart cities, the processing & analysis subsystem provides an extensible set of drivers for integrating and supporting various processing engines (e.g., Apache Storm[3], Amazon IoT[4], Google Cloud Dataflow[5], or Esc [85]). On top of basic processing capabilities, the processing & analysis subsystem provides a novel processing approach that is based on a lambda architecture[6]. Employing lambda architectures allows SCOS to balance throughput, latency, and fault-tolerance by simultaneously executing batch processing on batch data, and stream processing on real-time data. Next to processing, the analysis of processed data is another vital aspect in a smart city environment. Thus, the data layer provides data aggregation mechanisms, novel

[2] http://hubofallthings.com/
[3] http://storm.apache.org/index.html
[4] https://aws.amazon.com/iot/
[5] https://cloud.google.com/dataflow/
[6] http://lambda-architecture.net

querying capabilities [20], and a transparent environment for executing tailored processing and analysis routines. For building and deploying executable routines, the subsystem provides on the one hand a pre-built set of commonly applied processing and analysis logic, and on the other hand a development kit for developers for building and deploying custom code.

8.3.3 Application Layer

The third and topmost layer of SCOS, the *Application Layer*, provides a comprehensive set of methodologies and tools for efficient design, development, distribution, and operation of smart city applications.

Design & Development

One of the main goals of the SCOS *Application Layer* is to enable practitioners to create smart city applications in a simple, structured, and well-defined way. To accomplish this, the *Design & Development* subsystem is built around a comprehensive methodology for architecting, developing, and operating cloud-native smart city applications based on the MADCAT [46] methodology. The methodology provides actionable guidelines for iteratively architecting and implementing smart city applications, both for new applications, as well as for migrating existing applications to a cloud-native architecture [5] suitable for the SCOS smart city application ecosystem. Furthermore, SCOS provides a unified mechanism for describing smart city applications and their components along with deployment properties and requirements, similar to TOSCA [13]. This mechanism allows for easy sharing of applications and application components between SCOS deployments and provides a clear separation between application component dependencies and infrastructure requirements to create infrastructure-agnostic application descriptions. Such applications can then be offered in a smart city application market [107], where users can buy and sell applications and their components using an open self-service platform.

Runtime Environment

To allow for seamless execution of smart city applications, the SCOS *Runtime Environment* provides a configurable and adaptive execution environment for cloud-based applications that is independent of the underlying physical infrastructure. The execution environment incorporates a pluggable, unifying infrastructure abstraction [88] to transparently support and manage multiple application deployment mechanisms, such as container-based deployments (e.g., Docker[7]) and virtual machine-based deployments that are provisioned using predominant cloud offerings (e.g., OpenStack[8] or Amazon EC2[9]). The runtime environment furthermore provides a service mobility mechanism [90] that allows for seamless migration of application components between data centers and stakeholder premises. By moving processing logic closer to data sources and/or data sinks, network overhead and associated costs can be reduced. Additionally, component migration

[7] https://docker.com
[8] https://openstack.org
[9] https://aws.amazon.com/ec2

allows for the execution of applications that could otherwise not be executed due to compliance constraints.

Lifecycle Management

Closely integrated with the runtime environment, the *Lifecycle Management* subsystem is responsible for managing the complete lifecycle of smart city applications in SCOS. For applications developed using the SCOS methodology discussed above, the lifecycle management subsystem provisions required resources according to the specified requirements as well as applicable constraints, and deploys all application components according to their deployment manifests [88, 109]. Stakeholders can then start, stop, or pause applications. During runtime, the lifecycle management subsystem will continuously monitor deployed smart city applications to optimize application deployment topologies [113]. Furthermore, monitoring data is pushed to the data layer, which enables consumption and further processing by applications. In addition to monitoring the overall operations of executed application components, the lifecycle management subsystem continuously monitors and verifies mandatory compliance criteria and enforces them by initiating component migrations if possible as well as instructing the runtime environment to deny access to critical resources if necessary.

8.3.4 Cross-Cutting Concerns

The final layer of SCOS represents cross-cutting concerns. Components or applications of SCOS require common functionality (e.g., authentication) that span across several layers. Since such functionality is affecting the overall system, it is centralized in one place in order to avoid updating components throughout the system in case a certain behavior (e.g., logging) has to be changed.

Tenant Management

Smart city applications operate under complex compliance and security regulations. Furthermore, since these applications have to operate at large scale, are maintained by varying stakeholders, and provided in different possible facets, a plethora of constraints need to be efficiently managed. Therefore, the *Tenant Management* subsystem supports the magnitude of participating stakeholders and allows them to specify their own security and compliance guidelines. Next, in order to allow large-scale interactions of stakeholders in a smart city environment, tenant management supports a flexible interaction approach that allows expressing specific constraints that need to be respected. For example, considering an interaction among stakeholders in this context, constraints that are valid for two stakeholder having direct interaction can become invalid if another stakeholder joins the interaction, which is triggered by the complex data regulations in such environments. Another important aspect of tenant management is enabling the clean separation and isolation of any type of data, but especially for sensitive data. Thus, tenant management enables each stakeholder of SCOS to clearly define the following constraints regarding its data. First, tenants specify which data they provide and in which quality. Second, tenants can decide which data can be shared or consumed. Third, tenants can describe with whom they want to share data, or who is specifically allowed to consume provided data. Finally, tenants can specify

which data and from whom they want to consume data. Based on this specification, the tenant management subsystem derives a constraint matrix that clearly regulates data exchange in SCOS, which avoids undesirable data transfer by respecting various forms of interactions (e.g., direct or transitive). Nevertheless, this approach still empowers novel mechanisms that allow data processing in highly constrained interaction scenarios by using capability migration like presented in Chapter 6. In addition to data concerns, tenant management also manages a consolidated view on resources that are consumed by and available to tenants of SCOS. Based on the underlying transparent infrastructure layer, tenant management not only uniformly provides cloud resources like virtual machines, but also offers other forms of infrastructure resources such as IoT devices. This enables tenants and their respective applications to lease and release resources following a consistent, but flexible and extensible model.

Security & Compliance

Stakeholders in smart city environments implicitly expect and demand services to be secure, as well as to preserve their privacy. Thus, SCOS provides a *Security & Compliance* subsystem that allows addressing both, basic and complex security aspects. First, since data in SCOS is constantly flowing among different components or applications, which can reside inside or on top of SCOS, the security & compliance subsystem provides mechanisms to protect data in transit by using strong encryption mechanisms. In addition, components of SCOS that are dealing with sensitive data are also provided with approaches for securely storing this data. Second, since SCOS must be able to deal with a broad variety of stakeholders and users, the security & compliance subsystem provides capabilities that facilitate strong authentication mechanisms (e.g., biometric and multi-factor authentication) that can be used by components of SCOS to clearly specify who can access a specific service. Third, next to authentication, SCOS also provides authorization capabilities that allow enforcing permissions before accessing applications or manipulating data. Fourth, in order to allow operators to manage SCOS more efficiently, the security & compliance subsystem provides auditing and logging functionality on component level. Fifth, in order to keep the overall stack of components in SCOS secure, the security & compliance subsystem provides configuration management for automatically delivering software and security updates for different layers of SCOS. Sixth, since applications in the smart city domain need the ability to adapt to users, they come with various configuration alternatives that depend on the user's preferences. Thus, it is important for SCOS to allow applications to collect user-specific data in order to characterize a specific user. However, this form of characterization, which comprises both behavior and preferences, represents a possible threat for users. Therefore, the security & compliance subsystem provides mechanisms that explicitly assure and preserve the user's privacy. In addition to these common security capabilities, the security & compliance subsystem also deals with security requirements that emerge from the underlying infrastructure layer. Given the large number of resources that are available, SCOS provides security management that is able to deal with the intrinsic scalability requirements. Next, since especially IoT resources embody a vital aspect not only in enterprise systems, but also in consumer solutions, the security & compliance subsystem enables flexible security models. Based on these models, SCOS can adapt to and respect emerging complex security requirements from the various domains it is operating in.

8.4 Related Work

The rapid adoption of the smart city paradigm and its broad coverage in research initiatives have led to several approaches covering abstract concepts as well as specific tools to address emerging complexities. SmartSantander [84] proposes a city-scale experimental research facility supports applications and services in the smart city context. SmartSantander focuses primarily on the IoT element of a smart city and aims to provide a large-scale testbed that helps with issues arising from connecting and managing IoT infrastructure elements. Jin et al. [50] introduce an information framework for creating smart cities through the IoT. The authors focus on the urban information system as an omnibus volume starting from the sensory level up to issues of data management and cloud-based integrations. In a similar way Mitton et al. [71] propose the combination of cloud and sensors in a smart city environment. The focus of their work lies on the design of a pervasive infrastructure, where services interact with their surrounding environment with a clear focus on sensors in the IoT context. In a more abstract conceptual direction Chourabi et al. [23] present a framework to understand the concepts of smart cities. The authors provide a conceptual framework to comprehend the vital elements in a smart city by identifying critical factors. The framework suggests research directions and outlines practical implications. In a similar way, Nam et al. [74] try to identify how a city can be considered smart, by aligning strategic principles in the context of technology, people, and institutions, which represent the main dimensions of a smart city.

CHAPTER 9

A Smart City Application to enable a Holistic, Interdisciplinary Decision Support System for Sustainable Smart City Design

In this chapter, we present results from our ongoing efforts towards engineering next-generation smart city applications to provide stakeholders with a holistic and tailored view on their problem domain to support them in managing relevant aspects of the city, and furthermore provide effective assistance for important decision processes. We introduce the URBEM Smart City Application (USCA), an interdisciplinary decision support system, and present different views on its use by involved experts from four central smart city domains in the context of a smart city research initiative in the city of Vienna.

9.1 Introduction

Todays cities are evolving into complex behemoths consisting of a myriad of sophisticated entangled systems. The recent advent and rapid adoption of the smart city paradigm that enables vital new possibilities, also has significantly contributed to the intrinsic complexity [76] of such systems. Complex systems and models from multiple domains, such as e-government, traffic and transportation management, logistics, building management, smart health care, and smart grids, have become essential drivers for sustained innovation and improvement of citizen wellbeing. In order to enable sustainable, supply-secure, and future-proof planning that can keep up with today's rapid city growth and urbanization, it is vital to enable stakeholders to make well-informed decisions. To achieve this they rely on the expertise of domain experts who in turn use complex models for analyzing and simulating various aspects of a city ranging from building physics, energy and mobility systems, to sociological and behavior models. The most vital part, however,

is the ability to effectively integrate these models, allowing them to stimulate each other, which presents the enabling key for creating the foundation for sustainable and future-proof smart city design.

In this chapter, we present results from our work in the context of the URBEM smart city research initiative, specifically we introduce the USCA, a smart city application enabling the scenario introduced in Section 3.1 by utilizing the SCOS. Specifically we emphasise the following intrinsic requirements that need to be addressed to fully enable the crucial collaboration of stakeholders and domain experts in URBEM. First, we need the ability to integrate heterogenous, multidimensional data sources that are omnipresent when operating applications in smart city ecosystems. Second, since in smart cities and especially in URBEM, there are various stakeholders involved that enforce and must respect a plethora of different compliance and privacy regulations, we need mechanisms that allow for respecting these constraints, without impeding stakeholder interaction. Finally, to fully support domain experts in URBEM, we can not only provide them with pre-built services and applications, but must allow them to integrate and facilitate their own established and well-known heterogenous tool stacks. Based on requirements, we show how we achieve this vital integration by developing the URBEM Smart City Application (USCA) and outline how this has benefited involved domain experts and stakeholders.

The remainder of this chapter is structured as follows. In Section 9.2, we present USCA, a representative smart city application that emerged as a result of the URBEM research initiative, and discuss how it tackles the identified challenges. In Section 9.3, we outline how domain experts and stakeholders use and benefit from USCA, followed by a comprehensive summary in Section 9.4.

9.2 The URBEM Smart City Application

In this section we present the URBEM Smart City Application (USCA). USCA allows for integrating models of multiple domain experts that operate in domains such as building physics, electrical and thermal energy, energy demand modeling, as well as mobility and sociological behavior modeling to provide an interactive, explorable, and dynamic visualization for stakeholders. It is an application within the Smart City Application Ecosystem (SCALE), as introduced in Chapter 8. Figure 9.1 shows an overview of USCA in the context of SCALE.

In USCA, stakeholders interact with a dynamic, web-based, geo-spatial *Visualization* that allows them to freely explore the city as well as different evolving aspects in the context of multiple scenarios with predictions up to the year 2050. Stakeholders can not only explore the city as a whole, but also inspect it in varying levels of detail, from districts, over blocks, down to individual buildings. They can enrich their view with the results of domain expert models by dynamically adding and removing additional layers. This enables them to get a detailed look at various aspects of the city in a dynamic and integrated fashion. Figure 9.2 shows an example where specific natural gas uplinks for several building blocks in Vienna are explored.

Each of these model interactions spawns specific requests, which are handled by the *Request Router*. The Request Router acts as a smart request proxy and is responsible for elastically scaling up and down the necessary infrastructure resources based on the request patterns of USCA. To achieve this it utilizes the capabilities provided by the *Infrastructure & Resource Management*

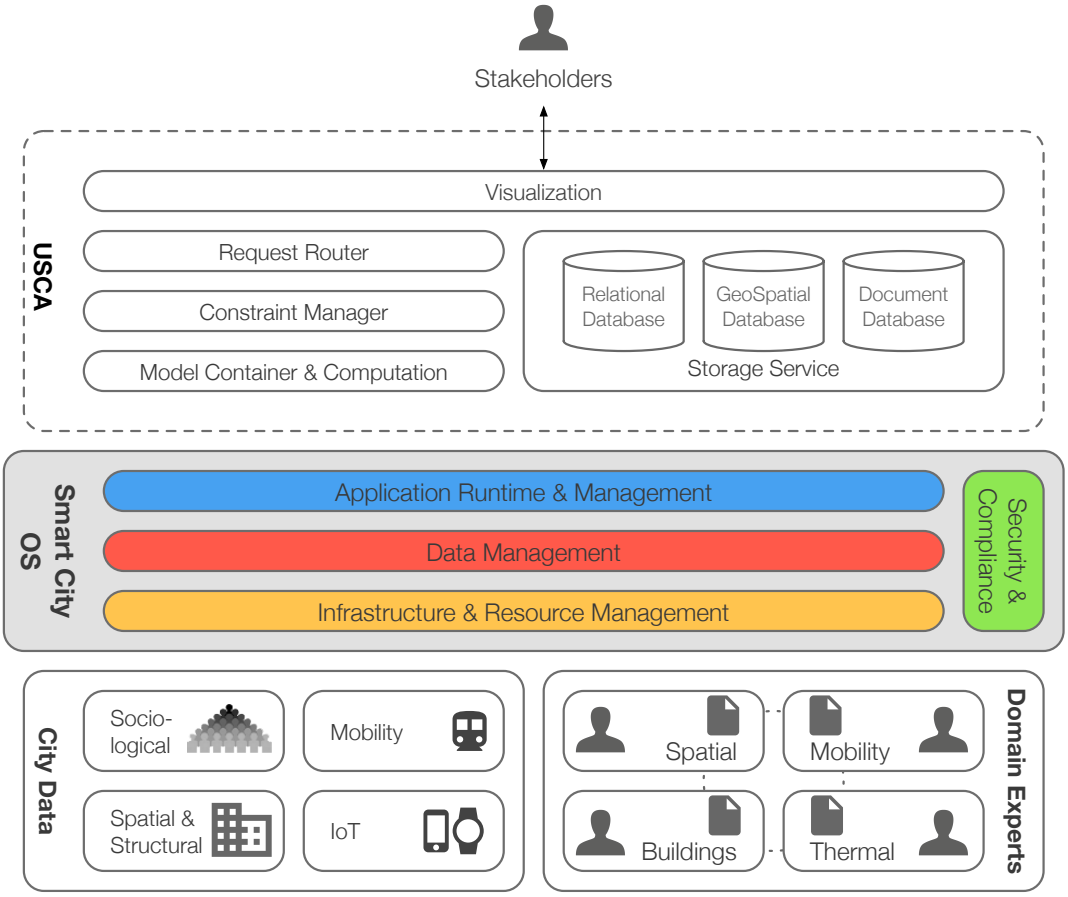


Figure 9.1: URBEM Smart City Application Overview

Layer of the Smart City Operating System (SCOS) and the SYBL [24] language. This allows USCA to maintain a small footprint as it can ensure that resources are only consumed when needed. Additionally, the *Infrastructure & Resource Management Layer* enables infrastructure-agnostic deployments via the approaches presented in Chapter 4 so that USCA can be executed on a variety of different platforms, which is an important factor in the heterogeneous infrastructure landscape of current smart cities. It is further able to manage and operate edge infrastructure resources using LEONORE [108] and DIANE [109]. The Request Router then passes each request to the *Constraint Manager*, which in turn is responsible for ensuring that USCA meets the aforementioned complex compliance and privacy regulations. The Constraint Manager inspects each request to check which data sources and domain experts' models are needed to fulfill the requests. Based on this information it ensures that no privacy or compliance constraints are violated and forwards the specific requests to the *Model Container & Computation* component. If constraints are violated, the Constraint Manager can in turn utilize SCOS's *Security & Compliance Layer* to offer ad hoc compensations using capability migrations provided by Nomads

Figure 9.2: Visualization of gas heating uplinks for building blocks in Vienna.

(Chapter 6). The Model Container & Computation component ensures that the domain experts' models are correctly executed and are supplied with all necessary data. Along with the *Storage Service*, these components represent the core elements of USCA and are key to enabling a holistic, integrated city view. The Model Container & Computation component provides means for provisioning and executing containers. We currently support two popular container formats, Docker[1] and Rkt[2]. This allows domain experts to continue using their well-known and established tool stacks without sacrificing the ability to integrate them into USCA. The containers are packaged to include all necessary runtime artifacts. Additionally, they can be checked and verified to ensure that compliance and privacy constraints are not violated by utilizing the Smart Brix framework (Chapter 7). This vital feature is enabled via the *Application Runtime & Management Layer* of SCOS. To give models access to required data, data containers are transparently integrated by injecting necessary container links. This mechanism enables a minimally invasive approach that allows domain experts to integrate provided capabilities into their own tools. Domain experts then simply access data in the data container via the established link and store the results of their models in the same container. In the background the *Storage Service* is used to provide necessary data via these links, as well as to store the model results in the appropriate data store. Additionally, the Constraint Manager can check at all times if data in transit can be consumed by the respective domain expert, which ensures that all compliance and privacy constraints are also met on the data level. The final element in empowering the Storage Service is the ability to utilize the *Data Management Layer* of SCOS. It enables the Storage Service to access a wide range of city data in various formats ranging from traditional relational data and documents, to live streaming data from the Internet of Things. All this data in turn can be incorporated into

[1] https://www.docker.com
[2] https://github.com/coreos/rkt

domain models as well as directly into the visualization.

9.3 Domain Expert Perspectives

In this section, we discuss the use of USCA by experts and their models from four different smart city domains. We briefly outline the specifics of each model, how it utilizes and benefits from USCA, and conclude with the observed benefits from the stakeholder's perspective.

9.3.1 Building Models

One of the key elements in urban city planning is to develop a proper method that allows simulating the effects of different urban development strategies (e.g., for 2020, 2030, and 2050) focusing on all buildings within a district or even an entire city. Therefore, different urban development scenarios are used as initial parameters to run building simulations for the focused building stock. Individual indicators (e.g., heating demand or refurbishment rates) are usually insufficient to run commercial building simulation tools. In order to maintain good performance and a time-efficient calculation period to simulate an entire urban environment, the simulation efforts for single buildings must be as low as possible. This model generates scalable density functions for both, residential and non-residential buildings by considering particular construction periods, different HVAC[3] technologies as well as individual occupancies in the course of a social milieu-based approach. The result is a comprehensive matrix of simulated density functions consisting of all possible combinations of the parameters mentioned above. The ability to expose this model within USCA allows the electrical and thermal grid models to utilize generated results that enable them to use hourly load profiles, which in turn are required for the technical simulations. In order to generate high-resolution load profiles (both temporal and geographic) for each building, only the input of the urban development scenarios generated within the energy demand model and the number of buildings is needed. This significantly increases the possible level of detail for planning decision support in this domain.

9.3.2 Energy Demand Models

The model concerning the perspectives of building energy demand and supply mainly handles the long-term development of heating and cooling demands. Additionally, it is concerned with the demand for domestic hot water (DHW) in buildings and the interactions with grid-bound heating supply, specifically focusing on gas and district heating. Since the building stock causes a large part of the energy demand of modern cities and the realization of the European energy targets[4]require a decrease in this demand, a reduction of fossil fuels (e.g., gas and oil), as well as the integration of renewable energy sources. This can be achieved by thermal refurbishments of buildings, by changing the heating systems, and by using a different energy carrier. The used model simulates the long-term investments in the building sector and optimizes the investments in the expansion of the district heating and/or gas infrastructure. This model not only considers

[3]heating, ventilating, and air conditioning

[4]http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2013:178:0107:0108:EN:PDF

the current legislative and policies, but also assumptions for the future development of them [32]. The emerging results are spatially resolved. As this analysis is from an economic point of view, the integrated approach within USCA allows enriching the model with more technical details. Based on a thermal grid analysis of the status quo for the base year, information about the spatial heat losses or remaining capacities without expansion can be pointed out and are used as input for the economic model. Subsequently, the long-term results regarding heating and cooling demand, and the expansion of the district heating network for several years within the considered horizon, are the basis for thermal grid analysis. The results of the analysis allow finding appropriate measures regarding the grid to react to these changes.

9.3.3 Electrical Grid Models

The model of electric supply networks can predict reliability, overloads, and network utilization considering the limits of operational equipment used inside the network. In addition, the model is capable of making statements about which requirements will arise for future power grids through increasing integration of decentralized energy resources, decentralized storage, and energy combined supply networks (energy hubs), while considering demographic change. Modeling and simulating an electric supply network for urban areas requires an approach that allows for incorporating large amounts of network data. Therefore, power flow studies [36] for distribution areas fed by substations are performed. Transmission areas inside an urban area are neglected. Furthermore, energy and infrastructure (heat, gas, and electrical) combined systems are simulated by direct current power flow calculations within an optimization using an energy hub approach [34]. Possible objective functions of the optimization are the minimization of CO_2 emissions, or the minimization of line utilization, with the explicit constraint that network capacities (electricity, gas and district heating) should not be overloaded. Results from the energy demand and building models inside USCA form the framework conditions and input parameters for the electric supply network model. Based on a technical analysis for a base year, scenarios with increasing integration of renewables and cooling demand that are mainly covered by electric energy reveal the limits of currently used network equipment (lines, transformers, etc.). The obtained results affect future investment decisions in network utilities to address changing requirements within the electric supply networks.

9.3.4 Thermal Grid Models

Current developments of the European energy market are influencing the operational strategy of heat suppliers. Especially providers of district heating systems fed by conventional heat production have to react with appropriate measures to these changes. The integration of thermal storages, decentralization of heat production, changing heating technologies or adjusting the temperature of district heating networks make it necessary to simulate and analyze existing and future designs of district heating systems. In order to achieve comparable conclusions about operating behavior of district heating systems, it is essential to create a corresponding model including main components like pipes, pumps, storages, and valves. The basic idea of the created numerical model is the combination of a steady state hydraulic and a transient thermal calculation of the district heating network. The results of the iterative hydraulic calculation are the pressure and velocity distribu-

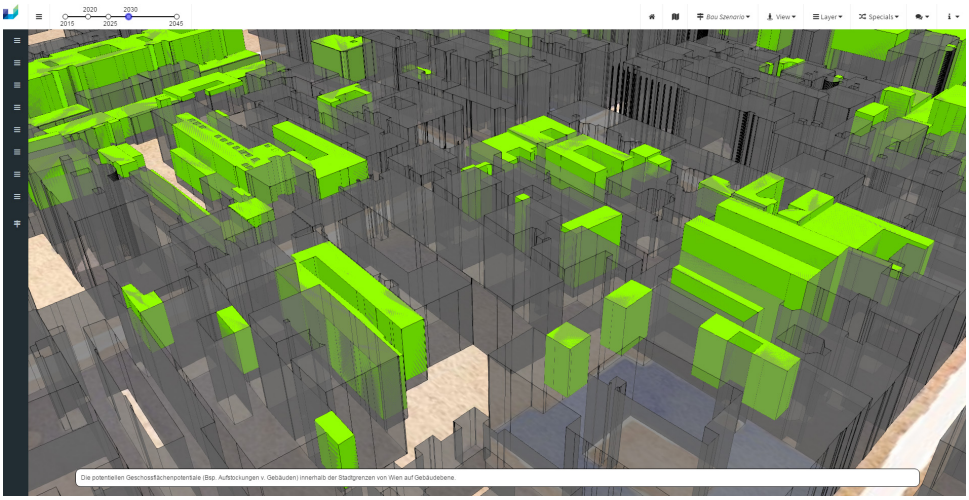


Figure 9.3: Floor-space potentials for individual buildings up to the year 2030.

tion of the pipe network [114]. These results serve as input parameter for the thermal calculation. To simulate the thermal behavior of the district heating network a discretized one-dimensional pipe model is used. The discretization is done using the finite-volume method and the resulting equation system can be solved explicitly or implicitly. A common way to define the topology of networks is the usage of a node-edge matrix. This so-called incidence matrix is generated automatically from given GIS data. The usage of the simulation model within USCA increases the capability in terms of interactions with more detailed data provided by other models that are integrated in USCA. The output of the model can be used to support economic analysis from a technical point of view or serve as additional input for analysis of energy combined systems. The possibility to link models of different disciplines extends the scope of the overall application.

9.3.5 Spatial Modeling and Visualization

The visualization aspect represents a vital instrument to communicate the results of, as well as to interact with, the models of the domain experts via a simple and intuitive interface. The ability to spatially resolve the results of the models and to evolve them over time is an essential factor in understanding the impacts of complex systems on the city. USCA allows to seamlessly incorporate and combine city data with the results of domain expert models, which enable novel ways for illustrating vital elements for city design. In Figure 9.3 we see a spacial placement of forecasted floor-space potentials for individual buildings in the year 2030. Through USCA it is possible to incorporate various city data sources to get an accurate picture of specific development potentials, which is an important factor for spatial modeling. We could also already demonstrate the applicability and benefits of this approach in [31].

The development of smart cities requires the integration of multiple stakeholders from different fields. Therefore, USCA provides an easy to manage tool for displaying all relevant information, whereas the visualization enables all involved entities to get an overview about the

Figure 9.4: Energy Demand Visualization for the the 11th district of Vienna

complexity of the system and to gain an understanding about the main influences and challenges within other disciplines. The consequences of decisions (e.g., investment decisions, legislation, definition of subsidies) within other fields and additional required measures can be highlighted using the visualization. A representative example that illustrates these integration benefits can be seen in Figure 9.4. Figure 9.5 illustrates detailed district heating demands of all blocks within Vienna's 11th district. The foundation for this is the energy demand model in combination with city data enriched by the building models, which in turn provides detailed load profiles. These high-resolution load profiles are then used by the electrical and thermal grid models to deliver specific grid impacts, which can be visualized at varying spatial detail levels via simply zooming in or out in the Visualization (Figure 9.5, Figure 9.6).

9.4 Summary

The smart city paradigm led to a transformation of today's cities to complex systems of systems with a plethora of increasingly complex dependencies and interactions. A large number of stakeholders from multiple different domains pose complex requirements on these systems that might be conflicting and will change over time. Efficient design, engineering, and operation of such systems is increasingly challenging but represents an essential ingredient in supporting stakeholders to make well-informed decisions.

In this chapter, we presented results from our ongoing efforts towards engineering and operating next-generation smart city applications that aim to provide stakeholders with a holistic as well as customized view on their problem domain. Such smart city applications must be designed to support stakeholders from different domains in managing and affecting relevant aspects of the city and provide effective assistance for important decision processes. To address these

Figure 9.5: Energy Grid High Level Overview for a district

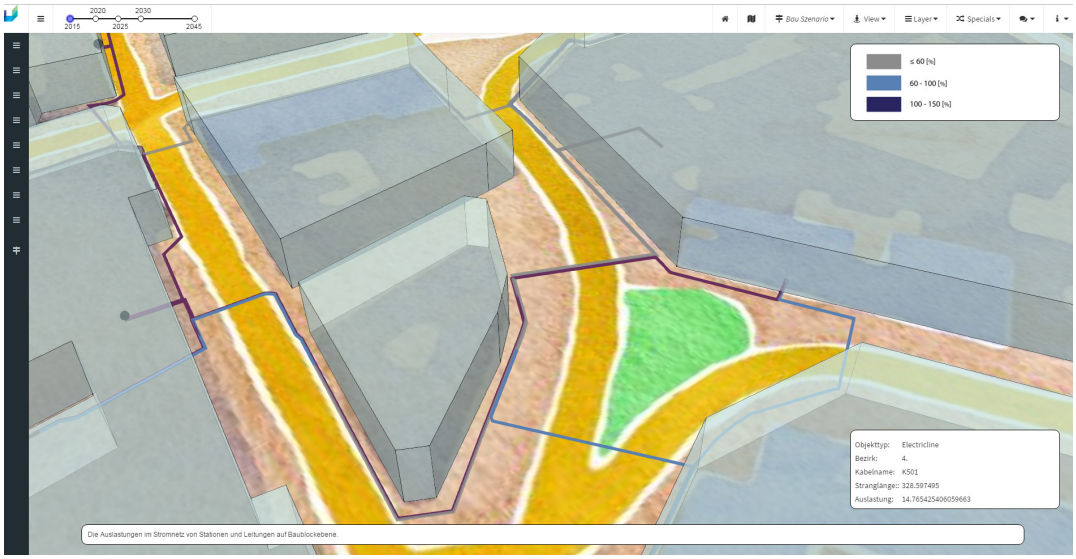


Figure 9.6: Detailed Energy Grid for a specific set of buildings

challenges, we introduced USCA, an interdisciplinary decision support system for holistic city planning and management. USCA is a cloud-based application built upon our recent work on smart city application ecosystems that uses a smart city operating system as its foundation. The application provides a holistic, integrated view on multiple complex domains based on models provided by different domain experts to support complex decision processes, while rigorously respecting relevant confidentiality and security constraints. We furthermore reported on the use

of USCA by stakeholders from four central smart city domains in the context of a smart city research initiative in the city of Vienna.

CHAPTER 10

Application Architecture Blueprints for Future Smart City Applications

In this chapter we present architectural blueprints for future smart city applications, based on the previously presented Smart City AppLication Ecosystem (SCALE). We select three representative types of smart city applications and indentify their key requirements along with architectural guidelines to implement them based on our experience with SCALE.

10.1 Introduction

The rapid rise and pervasive adoption of the smart city paradigm in conjunction with the Internet of Things (IoT) are reshaping the world's metropolises. Today's cities have evolved to become behemoth of connected devices cutting through all vital domains, starting with building management and operations to smart grids and multi-modal traffic management up to social media driven citizen participation processes. To stay on top of this increasing complexity it is essential to enable experts and stakeholders alike to utilize this plethora of novel information in order to ensure optimal city planning, operations, and management. It is vital to provide stakeholders with capabilities to understand the massive amounts of data as well as support the efficient operation and management of infrastructures that are getting increasingly complex. This new city concept also comes with the ability to let citizens participate, which shows to be an integral element in advancing the city as a whole. To provide citizens with a toolset that allows building novel kinds of applications, we introduced the Smart City Application Ecosystem (SCALE) (Chapter 8) that allows practitioners in a smart city environment to build applications that can utilize novel capabilities emerging through the IoT as well as accessing the massive amounts of data in an efficient way. These applications in turn become composable interchangeable abstractions of capabilities much like the applications known from todays smart phones, but on a much larger scale. This evolution also shows to be a vital stepping stone to reach the so called Internet of Cities [89], an open market place where applications can interact and be exchanged between cities. This

Internet of Cities allows practitioners to use best practices and solutions from other cities to solve common problems in their own environment. In order for practitioners to understand where and how to apply SCALE, we identified common application cases todays smart cities are currently facing. In this chapter we present best practices (blueprints) for these common cases, show how they can be addressed, and which elements of SCALE and more specifically its core the Smart City Operating System (SCOS) have to be applied to enable them.

10.2 Blueprints for Smart City Applications

In this chapter we present blueprints for three different cases we identified as relevant throughout our smart city research. We identify their integral requirements and present approaches for each case, that demonstrate how best practices for each case should look like.

10.2.1 Smart City Planning

Case

Todays cities and their respective stakeholders want to provide their citizens with a sustainable supply secure, affordable, and livable city. In order to provide stakeholders in a smart city environment with the optimal foundations for a reflected planning and decision process to support such a lasting evolution it is essential to enable a holistic view on the city. To achieve this it is vital to integrate every important aspect of the city seen as a complex system as well as to provide an understandable view for the stakeholders. This can be achieved by enabling domain experts from relevant fields like energy, mobility, sociology or buildings, among others, to integrate their findings and views with one another. Such an integration allows for novel views as well as more meaningful integrated insights. For example understanding the social milieu of a district allows deriving certain mobility patterns that in turn enable novel aspects of multi modal transport planning and so on and so forth. The fuel for these integrated insights are the domains experts' different models that express certain facets of their respective domains. In order to enable these models it is vital to supply them with the most accurate and novel data about the city. This in turn allows them to provide a more precise analysis of the current state as well as to support more accurate forecast and prediction models.

Requirements

To enable the aforementioned case the following requirements need to be met. In order for the domain experts to be able to use the most current and accurate data in their respective models, they need to be able to access all the relevant data provided by the city. This massive amount of data comes from a wide range of different sources ranging from the Internet of Things (IoT), to open data sources up to old documents. This means that there needs to be a way to *manage massive amounts of diverse data* as well as to *integrate heterogeneous data sources*. Since the experts rely on proven tool stacks to implement their models, which in turn consist of a number of legacy systems combined with older mature system stacks, it is vital to *support legacy systems and tool stacks for model building*. Additionally, all these models need the computational

capacity to provide their results in a reasonable amount of time, which calls for *infrastructure resource provisioning for efficient model execution*. Finally, the smart city domain with its many stakeholders and data providers has a very complex set of security and compliance constraints. Therefore, to optimally use all the available data, it is essential to provide measures for *compliance and security enforcement*.

Approach

To enable the presented case we need to address the outlined requirements. In this section we introduce a blueprint, which accomplishes this by utilizing several components of the Smart City Operating System (SCOS). This Smart City Planning Blueprint is depicted in Figure 10.1 and consist of four main parts. In order to support the stakeholders in understanding and exploring the complex aspect of the city in a holistic way, we introduce a *Visualization Component*. It enables geospatial visualizations of different aspects of the city by integrating the results of the domain experts' models. The seconds component relevant for stakeholder interaction is the *Decision Support Component*, which provides detailed analytics and statistics based on the aforementioned integration of different domain model results. Both components rely on the SCOS support in two main areas. First, they need *Processing & Analysis*'s pre-built services in order to be able to effectively prepare the data for visualization and decision support. Additionally, they rely on *Infrastructure Management* to provide them with the necessary infrastructure resources to efficiently perform their tasks. Both components in turn rely on the *Model Container & Computation* component, which is responsible for executing the domain experts models. This component is tightly integrated with the *Data Management* component, which ensures that the experts are able to access all the diverse city data that is in turn used as the basis for their models. Additionally, it facilitates the standard and pre-built services provided by the SCOS's *Processing & Analysis* to support the model building when feasible. The *Data Management* component acts as a domain-specific data managing entity and is used by all other components. It ensures that the complex compliance and security requirements originating from the different data providers are being met by utilizing SCOS's *Security & Compliance*. Additionally, it utilizes the *Storage & Access* as well as the *Infrastructure Management* to ensure efficient and effective data handling of the massive amount of city data.

10.2.2 Smart City Infrastructure Management and Operation

Case

A city consists of a network of complex systems that interfuse all layers of its infrastructure. From building and traffic management, up to grid management, and controlling all these systems becomes more data-intensive and complex, but also more capable. This is increasingly the case with the rise of the Internet of Things where more and more sensor and devices become critical elements of said infrastructure. For stakeholders it is therefore of strong importance to have means that enable the effective management and operation of these complex systems in a transparent and efficient way. First and foremost, stakeholders need means to configure and control this cyber-physical infrastructure so they can manage it in a logically centralized way, a vital factor

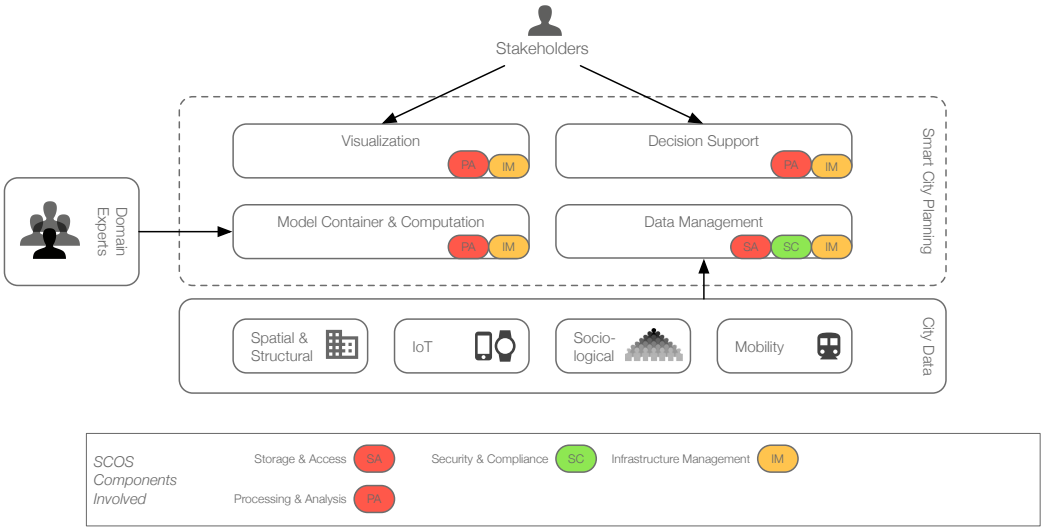


Figure 10.1: Smart City Planning Blueprint

for cost efficiency. The second important facet is the distributed autonomous and timely control of this infrastructure in order to be able to act and react to fast changing circumstances. This is not only essential in areas like building management where systems like HVAC, lighting or access control need to be managed, but becomes even more important in areas like traffic and grid management. The ability to have fine grained instant control is crucial to enable multi modal transport management or pave the way for the advent of autonomous cars where crossroad management becomes a highly sophisticated task. The integration of smart metering allows the smart management of grid utilizations and leads to significant cost reductions for operators and consumers a like. Finally, all these information needs to be composed in a smart manner to allow operators to control the systems as well as the physical infrastructure in an efficient way.

Requirements

In order to allow such smart infrastructure management and operation, several requirements need to be met. First, there needs to be a way to provide *IoT device lifecycle management*. This includes ways for device provisioning, management, and integration. Next it is vital to provide *IoT cloud application lifecycle management*, which calls for means to provide the necessary cloud infrastructure resources to enable these IoT devices. Additionally, means for *IoT Applications lifecycle management* need to be covered to ensure applications can be deployed and updated including the vital element of fault handling. The large amounts of data these systems generate call for a way to *manage this massive amount of diverse date*. Last but not least, these integrations touch many security critical domains so it is essential to provide *compliance and security enforcement*.

Approach

To address the identified requirements for the presented case, we introduce the Smart City Infrastructure Management and Operation blueprint in Figure 10.2. In essence the blueprint consists of four main parts, which are supported by various components of SCOS, to manage and operate available smart city infrastructures like buildings, hard infrastructure (e.g., traffic lights, bridges, roads), energy grids, or public transportation. First, to allow stakeholders to easily gain insights of the managed smart city infrastructure, we introduce a *Dashboard Component*. Dashboards support displaying both real-time, processed, and historical performance data (e.g., energy consumption) of the infrastructure. Furthermore, dashboards are specifically built for non-technical users, to avoid the need for custom programming, while supporting great flexibility by providing visual designers that allow stakeholders to create visualizations that are tailored to the available data and demands. Second, next to dashboards, another important aspect in infrastructure management and operation is the *Statistics & Reporting Component*. This component supports stakeholders in creating customized reports and statistics about the managed infrastructure, which can be used to deliver transparency into the gained performance data and furthermore promote compliance. Third, to provide the required data for the first two components, we introduce an *Analysis Component*. Based on the gathered and processed data that is provided by SCOS via the *Storage & Access*, the analysis component is able conduct specific operations like examining trends in collected infrastructure data (e.g., power conditions or temperature readings). Additionally, the Analysis component facilitates standard and pre-built services provided by SCOS's *Processing & Analysis*. The analysis results can the be used to identify problems in the future or areas that require maintenance. Following this approach can increase uptime of the infrastructure by dealing with issues before they cause downtimes. Finally, to collect data from the actual infrastructure and furthermore execute business logic, we facilitate a set of *Domain-specific IoT Apps*. These domain-specific apps can either be executed in SCOS, or directly in the actual smart city infrastructure. The apps gather data by facilitating the magnitude of IoT resources that are residing in the smart city infrastructure and are managed by SCOS. Furthermore, next to data collection, the domain-specific apps are able to control IoT resources to react to changes in the environment or optimize energy consumption of the managed infrastructure. To discover and communicate IoT resources, the applications are using the *Infrastructure Management* and *Configuration Management* of SCOS. While the apps are executed by facilitating the *Runtime Environment* of SCOS, SCOS also manages their complete application lifecycle via the *Lifecycle Management*. Furthermore, since both the Dashboard and Statistics & Reporting component require data and resources of multiple domains, SCOS handles and possibly restricts access to shared resources via the *Tenant Management*.

10.2.3 Smart City Citizen Participation and Engagement

Case

Another core tenet of Smart Cities is the close integration of citizens with various aspects of the city to increase citizen wellbeing and quality of living. Examples of such citizen participation and engagement applications are smartphone-based city transport information systems that allow users to quickly, efficiently, and ecologically travel through the city while anonymously providing

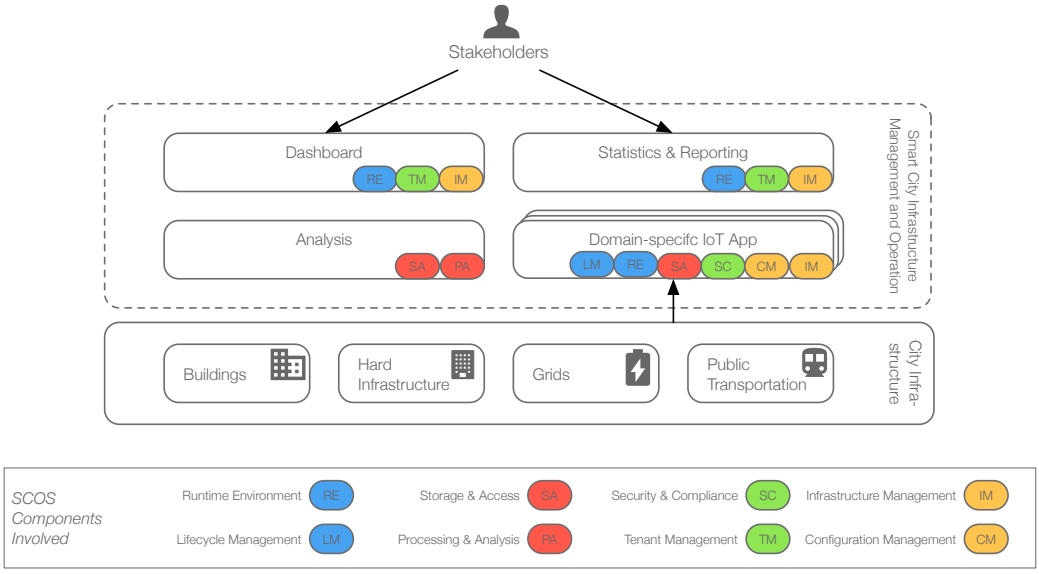


Figure 10.2: Smart City Infrastructure Management and Operation Blueprint

data to transportation providers and city administrators. This data in turn can be used to more accurately estimate upcoming demand, optimize city traffic by redirecting citizens away from congestions in both, public and individual transports, as well as to guide infrastructure planning. Another case is the integration of social media feeds to detect potential trends in citizen movement or concentration points, to help first responders to prevent or shorten the detection time of disaster scenarios. The utility of such applications is manifold and not only benefits providers and citizens alike but also creates a better understanding of the city as an adaptive system which benefits engagement. To enable the creation of such services it is essential to provide means for composing and integrating data from multiple smart city applications ranging from transport management and infrastructure management to grid control up to applications in the social media domains. Based on these data fast feedback loops need to be able that allow citizens and providers alike to act on new information as soon as possible.

Requirements

Such citizen participation and engagement applications are characterized by a number of requirements. Since these applications heavily rely on data from many different stakeholders in the smart city, ranging from public transport providers, road traffic authority up to energy providers, there must be a way to *securely and efficiently access* data from multiple other smart city applications in a unified way. Due to the integration and enrichment of information from multiple sources, there must be a way to *effectively manage large amounts of diverse data*. Integration of data about citizen movement and use of city infrastxructure further requires that *compliance and privacy regulations* must be guaranteed and that relevant security policies as well as *tenant*

requests are enforced. Additionally, the runtime infrastructure for the application must be elastically provisioned and, if possible, deployed on cloud and edge infrastructure to minimize latency and communication costs while maximizing utility for citizens.

Approach

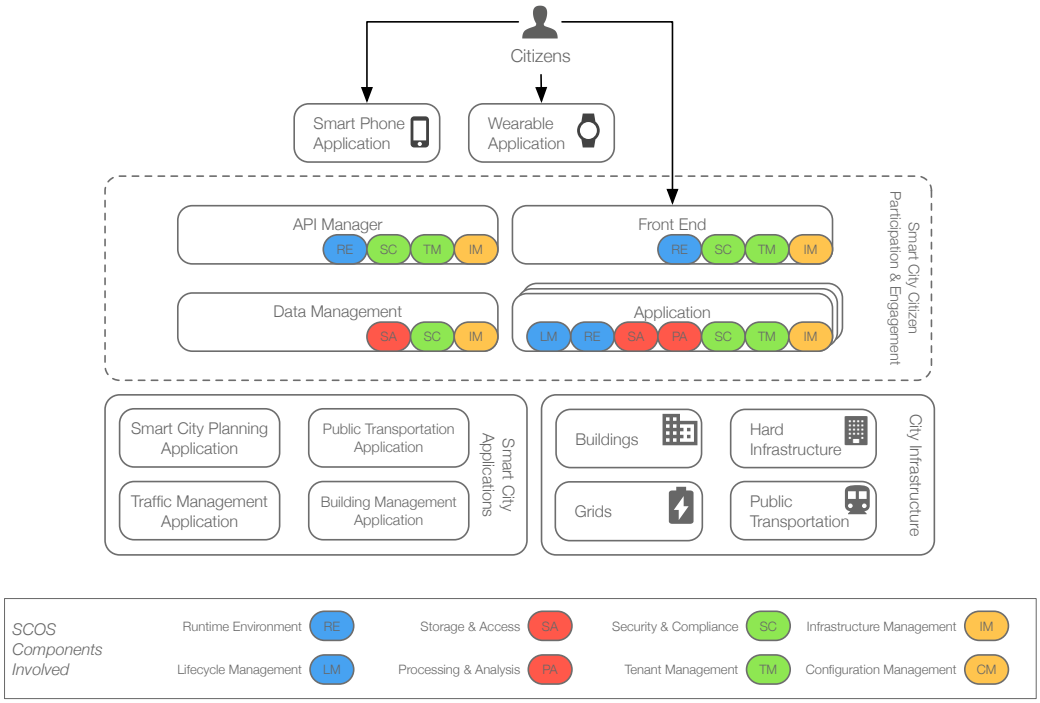


Figure 10.3: Smart City Citizen Participation and Engagement Blueprint

The blueprint shown in Figure 10.3 serves as a guideline to successfully implement citizen participation and engagement applications in Smart Cities based on four core building blocks. Citizens access the application using either web-based *Front Ends*, or using native applications deployed on smart phones or wearable devices (e.g., smart watches) that access the application through a managed API, provided by the *API Manager*. These two components are deployed on the SCOS utilizing the *Runtime Environment* and handle all citizen-facing interactions. Additionally they rely on *Security & Compliance* to ensure the critical compliance and privacy regulations are met as well as on *Tenant Management* to respect the various different tenant requests. Lastly they rely on *Infrastructure Management* to provide the necessary resources for elastically running them on cloud or edge infrastructures. The *Data Management* component provides a unified and secure abstraction for access to data from multiple underlying smart city applications. To ensure this it relies on *Storage and Access* for data integration smart city applications and city infrastructure. Additionally it utilizes *Security & Compliance* to ensure no data regulations are infringed as

well as the *Infrastructure Management* to acquire the necessary resources. At the core of every citizen participation and engagement application is the custom application logic that actually provides added value and comfort to citizens. *Applications* can make use of a number of SCOS components to ease provisioning and management as well as deployment and runtime.

10.3 Related Work

The broad coverage of the smart city paradigm in research initiatives around the globe has led to several approaches to address the inherent complexities. The approaches itself mainly focus either on the IoT aspect or on general complexity and integration issues surrounding smart cities. SmartSantander [84] proposes a city-scale experimental research facility that supports applications and services in the smart city context. The primary focus is the IoT element of a smart city and SmartSantander aims to provide a large-scale testbed that helps with issues arising from connecting and managing IoT infrastructure elements. Jin et al. [50] present a framework for creating smart cities through IoT. Their work is primarily centered around the IoT aspect of smart cities. Another approach in the context of IoT comes from Mitton et al. [71], where the authors propose the combination of cloud and sensors in a smart city environment. In their work they focus on pervasive infrastructures, where services interact with their surrounding environment. In terms of integration Wan et al. [115] present an event based architecture for M2M Communications for Smart Cities based on the SOFIA project with a case study in the vehicular context. On a more abstract level Chourabi et al. [23] present a framework to understand the concepts of smart cities. The authors introduce a conceptual framework to comprehend the vital elements in a smart city by identifying critical factors. This approach is similar to Nam et al. [74] who try to identify how a city can be considered smart, by aligning strategic principles in the context of technology, people, and institutions, which represent the main dimensions of a smart city. In this broader context Mulligan et al. [72] discuss the architectural implications of smart city business models. The authors specifically discuss the architectural evolution required for ensuring a smooth rollout and deployment of smart city technologies. In the context of citizen participation Khan et. al. [52] introduce a cloud based architecture for context aware citizen services in smart cities. The authors argue that cloud computing can provide a suitable computing infrastructure for smart city applications and also emphasize the importance of security considerations in this context. In a similar manner Jalali et. al [48] present a smart city architecture for community level services through IoT. Their smart city architecture enables community service providers and citizens alike to have access to real time data, which has been gathered through IoT as basis for decision processes and future planning.

10.4 Summary

As Smart Cities evolve towards the Internet of Cities, an open market place where applications can interact and be exchanged between cities shows to be an essential stepping stone. In order to enable these applications and to encourage a rapid adoption and deployment, it is essential to understand common problems and how to address them. In this chapter we outlined blueprints for three representative types of smart city applications, we identified their key requirements

along with architectural guidelines to implement them based on our experience with the Smart City AppLication Ecosystem (SCALE). While smart city applications are diverse, there are multiple common aspects that should be present in a smart city application platform, primarily infrastructure and data management, security and compliance management, along with a lifecycle and runtime environment for applications. An ecosystem like SCALE which incorporates these elements does allow practitioners to efficiently and effectively implement complex smart city applications that are secure, scalable, and resilient.

CHAPTER 11

Conclusion and Future Research

In this chapter, we summarize the main results of this thesis. In Section 11.1 we discuss the core outcomes of the conducted work and how the state of the art in research was advanced as part of this work. Then, the research questions posited in Section 1.2 are revisited and critically analyzed in Section 11.2. Finally, in Section 11.3 we present open topics for future research

11.1 Summary of Contributions

The rapid adoption of the smart city paradigm enabled a vast array of new possibilities to evolve and enhance the cities around the globe. However, these new possibilities also significantly increased the complexity, making it challenging to enable holistic approaches that allow stakeholders to successfully manage and plan in this domain. To enable such approaches practitioners and stakeholders need the ability to focus on the development of novel smart city applications, without the necessity to concern themselves with the arising complexities, by relying on the simplification capabilities of smart city application ecosystems.

In this thesis we presented novel approaches for infrastructure, operations and data management in such smart city applications ecosystems. We integrated these novel approaches into a comprehensive middleware toolkit, called the Smart City Operating System (SCOS) that serves as the central element for enabling smart city application ecosystems. This SCOS provides the essential simplifications to overcome the aforementioned complexities by introducing abstractions for infrastructure, data management and operations management, while still respecting the vital security and compliance aspects of the smart city domain. Specifically, our presented approach for infrastructure-agnostic artifact deployment allows the easy integration of heterogenous smart city infrastructures as well as the independent evolution of smart city applications and infrastructures. This enables a broader integration of, as well as an easy migration between, infrastructures for smart city applications. To address the key challenge of data management, we presented an approach for modeling and management of data sources in the smart city domain. Our approach allows for efficient, distributed data access for applications and introduces a simple technology-agnostic description of data sources for stakeholders. This mechanism enables the exposure of

relevant data sources not only for other stakeholders in the same city, but also in other smart cities around the globe significantly extending the application spectrum of smart city applications. Finally, in the context of operation management, we presented solutions to enable and improve the operation as well as evolution of smart city applications, while respecting the complex security and compliance constraints. We demonstrated that our service mobility approach can enable the execution, as well as significantly improve the results of, distributed analytical environments. Additionally, we presented a method for continuous evolution of container application deployments, capable of integrating security and compliance constraints. By doing so, we further enabled the use of software containers for building smart city applications, leading to a substantial increase in flexibility for developers and operation teams alike.

We evaluated the results of our investigations in the context of multiple scenarios and showed that the contributions of this thesis significantly improved infrastructure, operations, and data management in smart city ecosystems. By further enabling such ecosystems through the integration of our approaches into the SCOS, we significantly eased the creation and management of holistic interdisciplinary smart city applications. Furthermore, such an ecosystem represents an essential step towards the creation of composable, interchangeable abstractions of capabilities similar to the apps we know from todays smartphones, which will pave the way for an Internet of Cities.

11.2 Research Questions Revisited

Research Question I:
How can smart city application topologies be deployed while specifically respecting the heterogenous, evolving infrastructure landscape in smart city environments?

We have addressed this question with the contributions in Chapter 4. We introduced Smart Fabric, a methodology and accompanying toolset for infrastructure-agnostic deployment of application artifact topologies based on a constraint-based, declarative specification of the required deployment infrastructure. Current approaches do not sufficiently consider the specific, practical problems of dealing with evolving deployment infrastructure and closely tie application artifacts to their deployment targets. Smart Fabric allows for seamless, independent evolution of both, application components, as well as the underlying infrastructure. Moreover, our approach enables transparent application deployment and evolution between deployment targets, i.e., across traditional infrastructure boundaries (e.g., migrating applications between on-premise and PaaS offerings, or between PaaS and IaaS), without changes to application code.

Research Question II:
How can complex, heterogenous smart city data sources be efficiently exposed in distributed smart city environments?

We have addressed this question with the contributions in Chapter 5. We presented a methodology and toolset to model available smart city data sources and enable efficient, distributed

data access in smart city environments. A system model that provides a simple abstraction for the technology-agnostic description of available data sources and their subsets was introduced. Subsets can represent different aspects and granularities of the original data source, along with relevant characteristics common in the smart city domain. Based on this abstraction, we presented the SDD framework, a middleware toolset that enables efficient and seamless data access for smart city applications. It does so by autonomously relocating relevant subsets of available data sources to improve Quality of Service (QoS) based on a configurable mechanism that considers request latency, as well as costs for data transfer, storage, and updates.

Research Question III:
How can smart city applications be operated and evolved while respecting the complex and changing compliance requirements in smart city environments?

We have addressed this question with the contributions in Chapters 6 and 7. First, we presented Nomads, a framework for service mobility in Distributed Analytical Environments (DAEs). We showed that Nomads could significantly improve satisfiability of necessary service compositions to operate DAEs in complex constrained environments. Second, we introduced Smart Brix, a framework enabling continuous evolution of container application deployments, supporting traditional continuous integration processes as well as custom, domain-relevant processes, to implement security and compliance checks. Smart Brix not only enables the initial management of large-scale smart city application container deployments, but is also designed to continuously monitor the complete application deployment topology and allows for timely reaction to changes. This, in combination with its support for domain-relevant processes, enables the operation and evolution of large scale container based smart city applications under consideration of complex compliance requirements.

11.3 Future Work

In this thesis we presented different approaches for infrastructure, operations and data management in smart city application ecosystems. However, based on the discussion in Section 11.1, it is apparent that a number of relevant challenges were out of scope for this thesis. In the following, we outline some challenges and possibilities for future work.

- In the context of the presented approach in Chapter 4, we see the need to extend the presented framework with mechanisms to automatically gather entities of the system model. Specifically infrastructure specifications from available cloud services, as well as deployment units and deployment instances from existing deployment directives to simplify adoption of our approach. Furthermore, we aim to extend and integrate Smart Fabric with our work on IoT cloud applications [108, 109].

- With respect to the migration approaches presented in Chapter 5, we will integrate additional optimization mechanisms to further improve framework performance. Furthermore we will create additional smart city applications, covering different application areas in

collaboration with domain experts from URBEM, as well as other smart city initiatives. In the context of our research on the future Internet of Cities, we will extend the SDD framework to support autonomous, ad-hoc coordination of globally distributed SDD proxies to further optimize data instance placement in smart city application ecosystems.

- For the framework presented in Chapter 6, we plan to further minimize the number of migrations by incorporating heuristic-based approaches and genetic algorithms. Additionally, we plan to investigate the utility of Nomads in scenarios with less coordination, for example in the context of decentralized service choreography.

- Finally, to broaden the applicability of our approach presented in Chapter 7, we will extend the presented framework to incorporate more sophisticated checking and compensation mechanisms. We plan to integrate mechanisms from machine learning, specifically focusing on unsupervised learning techniques as a potential vector to advance the framework with autonomous capabilities.

Bibliography

[1] Sharad Agarwal, John Dunagan, Navendu Jain, Stefan Saroiu, Alec Wolman, and H. Bhogan. Volley: Automated data placement for geo-distributed cloud services. *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, pages 2–2, 2010. http://portal.acm.org/citation.cfm?id=1855713.

[2] R. Aggarwal, K. Verma, J. Miller, and W. Milnor. Constraint driven web service composition in METEOR-S. In *IEEE International Conference onServices Computing, 2004. (SCC 2004). Proceedings. 2004*, pages 23–30. IEEE, 2004. ISBN 0-7695-2225-4. doi:10.1109/SCC.2004.1357986. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1357986.

[3] Prathima Agrawal. Raft: A recursive algorithm for fault tolerance. In *ICPP*, pages 814–821, 1985.

[4] Michele Amoretti, Maria Chiara Laghi, Fabio Tassoni, and Francesco Zanichelli. Service migration within the cloud: Code mobility in SP2A. In *2010 International Conference on High Performance Computing & Simulation*, pages 196–202. IEEE, June 2010. ISBN 978-1-4244-6827-0. doi:10.1109/HPCS.2010.5547130. http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=5547130.

[5] Vasilios Andrikopoulos, Tobias Binz, Frank Leymann, and Steve Strauch. How to adapt applications for the Cloud environment. *Computing*, 95(6):493–535, 2012. ISSN 1436-5057. doi:10.1007/s00607-012-0248-2. http://dx.doi.org/10.1007/s00607-012-0248-2.

[6] Vasilios Andrikopoulos, Anja Reuter, Santiago Gómez Sáez, and Frank Leymann. A GENTL Approach for Cloud Application Topologies. In *Service-Oriented and Cloud Computing*, volume 8745, pages 148–159. Springer Berlin Heidelberg, 2014. ISBN 978-3-662-44878-6. doi:10.1007/978-3-662-44879-3_11.

[7] Nuno Antunes and Marco Vieira. SOA-Scanner: An Integrated Tool to Detect Vulnerabilities in Service-Based Infrastructures. In *Proc. Intl. Conf. on Services Computing*, pages 280–287. IEEE, 2013. ISBN 978-0-7695-5026-8. doi:10.1109/SCC.2013.28.

[8] Danilo Ardagna, Elisabetta Di Nitto, Giuliano Casale, Dana Petcu, Parastoo Mohagheghi, Sébastien Mosser, Peter Matthews, Anke Gericke, Cyril Ballagny, Francesco D'Andria, Cosmin-Septimiu Nechifor, Craig Sheridan, Elisabetta Di Nitto, Politecnico Milano, Dana

Petcu, Craig Sheridan, Cyril Ballagny, Francesco D Andria, and Peter Matthews. MODAClouds: A Model-driven Approach for the Design and Execution of Applications on Multiple Clouds. In *Proceedings of the 4th International Workshop on Modeling in Software Engineering*, pages 50–56. IEEE, 2012. ISBN 978-1-4673-1757-3.

[9] Owen Arden, Michael D. George, Jed Liu, K. Vikram, Aslan Askarov, and Andrew C. Myers. Sharing Mobile Code Securely with Information Flow Control. In *2012 IEEE Symposium on Security and Privacy*, pages 191–205. IEEE, May 2012. ISBN 978-1-4673-1244-8. doi:10.1109/SP.2012.22. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6234413.

[10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A View of Cloud Computing. *Communications of the ACM*, 53(4):50–58, 2010.

[11] Michael Armbrust, Ion Stoica, Matei Zaharia, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, and Ariel Rabkin. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010. doi:10.1145/1721654.1721672.

[12] Jason Bau, Elie Bursztein, Divij Gupta, and John Mitchell. State of the art: Automated black-box web application vulnerability testing. In *Proc. Symposium on Security and Privacy*, pages 332–345. IEEE, 2010. ISBN 9780769540351. doi:10.1109/SP.2010.27.

[13] T. Binz, G. Breiter, F. Leyman, and T. Spatzier. Portable Cloud Services Using TOSCA. *Internet Computing, IEEE*, 16(3):80–85, 2012. ISSN 1089-7801.

[14] Christian Bizer, Peter Boncz, Michael L. Brodie, and Orri Erling. The meaningful use of big data. *ACM SIGMOD Record*, 40(4):56, January 2012. ISSN 01635808. doi:10.1145/2094114.2094129.

[15] Dario Bonino, Claudio Pastrone, and Maurizio Spirito. Towards a Federation of Smart City Services. *International Conference on Recent Advances in Computer Systems (RACS 2015)*, (Racs 2015):163–168, 2015.

[16] R. Brooks. Mobile code paradigms and security issues. *IEEE Internet Computing*, 8(3): 54–59, May 2004. ISSN 1089-7801. doi:10.1109/MIC.2004.1297274. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1297274.

[17] Antonio Carzaniga, Gian Pietro Picco, and Giovanni Vigna. Designing distributed applications with mobile code paradigms. In *19th International Conference on Software Engineering (ICSE)*, pages 22–32. ACM Press, May 1997. ISBN 0897919149. doi:10.1145/253228.253236. http://dl.acm.org/citation.cfm?id=253228.253236.

[18] Antonio Carzaniga, Gian Pietro Picco, and Giovanni Vigna. Is Code Still Moving Around? Looking Back at a Decade of Code Mobility. In *29th International Conference on Software Engineering (ICSE'07 Companion)*, pages 9–20. IEEE, May 2007. ISBN 0-7695-2892-9.

[19] D Chandra, C Fensch, W Hong, L Wang, E Yardimci, and M Franz. Code generation at the proxy: An infrastructure-based approach to ubiquitous mobile code. In *ECOOP Workshop on Object-Orientation and Operating Systems*, 2002.

[20] Qiming Chen and Meichun Hsu. Cut-and-Rewind: Extending Query Engine for Continuous Stream Analytics. In Abdelkader Hameurlain, Josef Küng, Roland Wagner, Alfredo Cuzzocrea, and Umeshwar Dayal, editors, *Transactions on Large-Scale Data- and Knowledge-Centered Systems XXI*, volume 9260 of *Lecture Notes in Computer Science*, pages 94–114. Springer Berlin Heidelberg, 2015. ISBN 978-3-662-47803-5. doi:10.1007/978-3-662-47804-2_5. http://link.springer.com/10.1007/978-3-662-47804-2http://dx.doi.org/10.1007/978-3-662-47804-2_5.

[21] Y. S. Chen and Y. R. Chen. Context-oriented data acquisition and integration platform for internet of things. In *2012 Conference on Technologies and Applications of Artificial Intelligence*, pages 103–108, Nov 2012. doi:10.1109/TAAI.2012.64.

[22] B. Cheng, S. Longo, F. Cirillo, M. Bauer, and E. Kovacs. Building a big data platform for smart cities: Experience and lessons from santander. In *2015 IEEE International Congress on Big Data*, pages 592–599, June 2015. doi:10.1109/BigDataCongress.2015.91.

[23] Hafedh Chourabi, Taewoo Nam, Shawn Walker, J. Ramon Gil-Garcia, Sehl Mellouli, Karine Nahon, Theresa A. Pardo, and Hans Jochen Scholl. Understanding Smart Cities: An Integrative Framework. In *2012 45th Hawaii International Conference on System Sciences*, pages 2289–2297, 2012. ISBN 978-1-4577-1925-7. doi:10.1109/HICSS.2012.615. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6149291.

[24] Georgiana Copil, Daniel Moldovan, Hong Linh Truong, and Schahram Dustdar. SYBL: an extensible language for controlling elasticity in cloud applications. In *Int. Symposium on Cluster, Cloud, and Grid Computing*, pages 112–119. IEEE, 2013. doi:10.1109/CCGrid.2013.42. http://doi.ieeecomputersociety.org/10.1109/CCGrid.2013.42.

[25] Li Da Xu, Wu He, and Shancang Li. Internet of Things in Industries: A Survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, 2014. doi:10.1109/TII.2014.2300753.

[26] Florian Daniel, Fabio Casati, Vincenzo D'Andrea, Emmanuel Mulo, Uwe Zdun, Schahram Dustdar, Steve Strauch, David Schumm, Frank Leymann, Samir Sebahi, Fabien de Marchi, and Mohand-Said Hacid. Business Compliance Governance in Service-Oriented Architectures. In *International Conference on Advanced Information Networking and Applications*, pages 113–120. IEEE, 2009. ISBN 978-1-4244-4000-9. doi:10.1109/AINA.2009.112. http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=5076188.

[27] UN Desa. World urbanization prospects, the 2014 revision. *United Nations*, 2014. https://esa.un.org/unpd/wup/Publications/Files/WUP2014-Report.pdf.

[28] Docker. https://www.docker.com/what-docker.

[29] Nathan S Evans, Azzedine Benameur, and Matthew Elder. Large-scale evaluation of a vulnerability analysis framework. In *7th Workshop on Cyber Security Experimentation and Test (CSET 14)*, 2014.

[30] Aurélien Faravelon, Stéphanie Chollet, Christine Verdier, et al. Configuring private data management as access restrictions: from design to enforcement. In *Service-Oriented Computing*, pages 344–358. Springer, 2012.

[31] Julia Forster, Sara Fritz, Johannes M Schleicher, and Nikolaus Rab. Developer Tools for Smart Approaches to Responsible-Minded Planning Strategies. In *Real Time - Proceedings of the 33rd eCAADe Conference - Volume 1, Vienna University of Technology, Vienna, Austria, 16-18 September 2015, pp. 545-551*, page to appear, 2015.

[32] Sara Fritz. How public interventions in buildings energy efficiency affect the economic feasibility of a district heating network - a case study for vienna. *38th IAEE International Conference*, May 2015.

[33] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[34] Martin Geidl, Gaudenz Koeppel, Patrick Favre-Perrod, Bernd Klockl, Goran Andersson, and Klaus Frohlich. Energy hubs for the future. *IEEE Power and Energy Magazine*, 5(1): 24, 2007.

[35] Wolfgang Gerlach, Wei Tang, Kevin Keegan, Travis Harrison, Andreas Wilke, Jared Bischof, Mark DSouza, Scott Devoid, Daniel Murphy-Olson, Narayan Desai, and Folker Meyer. Skyport - Container-Based Execution Environment Management for Multi-cloud Scientific Workflows. In *Proc. Intl. Workshop on Data-Intensive Computing in the Clouds*, pages 25–32. IEEE, 2014. ISBN 978-1-4799-7034-6. doi:10.1109/DataCloud.2014.6.

[36] D. J. Gotham and G. T. Heydt. Power flow control and power flow studies for systems with facts devices. *IEEE Transactions on Power Systems*, 13(1):60–65, Feb 1998. ISSN 0885-8950. doi:10.1109/59.651614.

[37] Wei Hao, I-Ling Yen, and Bhavani Thuraisingham. Dynamic Service and Data Migration in the Clouds. In *2009 33rd Annual IEEE International Computer Software and Applications Conference*, volume 2, pages 134–139. IEEE, 2009. ISBN 978-0-7695-3726-9. doi:10.1109/COMPSAC.2009.127. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5254135.

[38] Robert G Hollands. Will the real smart city please stand up? intelligent, progressive or entrepreneurial? *City*, 12(3):303–320, 2008.

[39] Eman Hossny, Sherif Khattab, Fatma Omara, and Hesham Hassan. A case study for deploying applications on heterogeneous paas platforms. In *Proceedings of the 2013 International Conference on Cloud Computing and Big Data*, pages 246–253. IEEE, 2013. ISBN 9781479928293. doi:10.1109/CLOUDCOM-ASIA.2013.13.

[40] Jez Humble and David Farley. *Continuous Delivery*. Pearson Edu., 2010. ISBN 0321670221.

[41] Waldemar Hummer, Patrick Gaubatz, Mark Strembeck, Uwe Zdun, and Schahram Dustdar. Enforcement of entailment constraints in distributed service-based business processes. *Information and Software Technology*, 55(11):1884–1903, November 2013. ISSN 09505849. doi:10.1016/j.infsof.2013.05.001. http://www.sciencedirect.com/science/article/pii/S0950584913001006.

[42] Waldemar Hummer, Orna Raz, Onn Shehory, Philipp Leitner, and Schahram Dustdar. Testing of data-centric and event-based dynamic service compositions. *Software Testing, Verification and Reliability*, 23(6):465–497, 2013. doi:10.1002/stvr.1493.

[43] Waldemar Hummer, Florian Rosenberg, Fábio Oliveira, and Tamar Eilam. Testing Idempotence for Infrastructure as Code. In *Proceedings of the ACM/IFIP/USENIX 14th International Middleware Conference*, pages 368–388. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-45065-5. doi:10.1007/978-3-642-45065-5_19.

[44] Michael Hüttermann. *DevOps for Developers*. Apress, 2012. ISBN 978-1430245698. doi:10.1007/978-1-4302-4570-4.

[45] Christian Inzinger, Waldemar Hummer, Benjamin Satzger, Philipp Leitner, and Schahram Dustdar. Generic Event-Based Monitoring and Adaptation Methodology for Heterogeneous Distributed Systems. *Software: Practice and Experience*, 44(7):805–822, July 2014. doi:10.1002/spe.2254. http://dsg.tuwien.ac.at/staff/inzinger/dl/SPE_2014_monina.pdf.

[46] Christian Inzinger, S Nastic, S Sehic, M Vögler, Fei Li, and Schahram Dustdar. MADCAT: A Methodology for Architecture and Deployment of Cloud Application Topologies. In *Proc. SOSE'14*, pages 13–22. IEEE, 2014. doi:10.1109/SOSE.2014.9.

[47] Dragan Ivanović, Manuel Carro, and Manuel V Hermenegildo. A constraint-based approach to quality assurance in service choreographies. In *Service-Oriented Computing*, pages 252–267. Springer, 2012.

[48] Roozbeh Jalali, Khalil El-khatib, and Carolyn McGregor. Smart city architecture for community level services through the internet of things. In *Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on*, pages 108–113, feb 2015. ISBN 9781479918669. doi:10.1109/ICIN.2015.7073815.

[49] P Jamshidi, A Ahmad, and C Pahl. Cloud Migration Research: A Systematic Review. *IEEE Transactions on Cloud Computing*, 1(2):142–157, 2013. doi:10.1109/TCC.2013.10.

[50] J. Jin, J. Gubbi, S. Marusic, and M. Palaniswami. An information framework for creating a smart city through internet of things. *IEEE Internet of Things Journal*, 1(2):112–121, April 2014. ISSN 2327-4662. doi:10.1109/JIOT.2013.2296516.

[51] Ali Khajeh-Hosseini, David Greenwood, and Ian Sommerville. Cloud migration: A case study of migrating an enterprise IT system to IaaS. In *Proceedings of the IEEE 3rd International Conference on Cloud Computing*, pages 450–457, 2010. ISBN 9780769541303. doi:10.1109/CLOUD.2010.37.

[52] Zaheer Khan and Saad Liaquat Kiani. A Cloud-Based Architecture for Citizen Services in Smart Cities. In *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, UCC '12, pages 315–320, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4862-3. doi:10.1109/UCC.2012.43. http://dx.doi.org/10.1109/UCC.2012.43.

[53] Adlen Ksentini, Tarik Taleb, and Min Chen. A Markov Decision Process-based service migration procedure for follow me cloud. In *2014 IEEE International Conference on Communications, ICC 2014*, pages 1350–1354, 2014. ISBN 9781479920037. doi:10.1109/ICC.2014.6883509.

[54] Young Woo Kwon and Eli Tilevich. Cloud refactoring: Automated transitioning to cloud-based services. *Automated Software Engineering*, 21(3):345–372, 2014. ISSN 15737535. doi:10.1007/s10515-013-0136-9.

[55] D. Kyriazis, T. Varvarigou, D. White, A. Rossi, and J. Cooper. Sustainable smart city iot applications: Heat and electricity management amp; eco-conscious cruise control for public transportation. In *2013 IEEE 14th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, pages 1–5, June 2013. doi:10.1109/WoWMoM.2013.6583500.

[56] Franck Lebeau, Bruno Legeard, Fabien Peureux, and Alexandre Vernotte. Model-based vulnerability testing for web applications. In *Proc. Intl. Conf. on Software Testing, Verification and Validation Workshops*, pages 445–452. IEEE, 2013. ISBN 978-0-7695-4993-4. doi:10.1109/ICSTW.2013.58.

[57] James Lewis and Martin Fowler. Microservices, March 2014. http://martinfowler.com/articles/microservices.html.

[58] Frank Leymann, Christopg Fehling, Ralph Mietzner, Alexander Nowak, and Schahram Dustdar. Moving Applications To the Cloud: an Approach Based on Application Model Enrichment. *International Journal of Cooperative Information Systems*, 20(03):307–356, 2011. ISSN 0218-8430. doi:10.1142/S0218843011002250.

[59] Fei Li, Michael Vögler, S Sehic, S Qanbari, S Nastic, Hong-Linh Truong, and Schahram Dustdar. Web-Scale Service Delivery for Smart Cities. *Internet Computing, IEEE*, 17(4):78–83, 2013. doi:10.1109/MIC.2013.79.

[60] Huan-Chung Li, Po-Huei Liang, Jiann-Min Yang, and Shiang-Jiun Chen. Analysis on Cloud-Based Security Vulnerability Assessment. In *Proc. Intl. Conf. on E-Business Engineering*, pages 490–494, 2010. ISBN 978-1-4244-8386-0. doi:10.1109/ICEBE.2010.77.

[61] Richard Li, Dallin Abendroth, Xing Lin, Yuankai Guo, Hyun-Wook Baek, Eric Eide, Robert Ricci, and Jacobus Van der Merwe. Potassium: penetration testing as a service. In *Proc. Symposium on Cloud Computing*, pages 30–42. ACM, 2015. ISBN 9781450336512. doi:10.1145/2806777.2806935.

[62] Wubin Li, Ali Kanso, and Abdelouahed Gherbi. Leveraging Linux Containers to Achieve High Availability for Cloud Services. In *Proc. Intl. Conf. on Cloud Engineering*, pages 76–83. IEEE, 2015. ISBN 978-1-4799-8218-9. doi:10.1109/IC2E.2015.17.

[63] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, and Dawn Leaf. Nist cloud computing reference architecture. *NIST special publication*, 500(2011):292, 2011.

[64] Pu Liu and M.J. Lewis. Mobile code enabled web services. In *IEEE International Conference on Web Services (ICWS)*, pages 167–174 vol.1, July 2005. doi:10.1109/ICWS.2005.80.

[65] L. Lowis and R. Accorsi. Vulnerability Analysis in SOA-Based Business Processes. *IEEE Trans. Serv. Comput.*, 4(3):230–242, 2011. ISSN 1939-1374. doi:10.1109/TSC.2010.37.

[66] Lutz Lowis and Rafael Accorsi. On a Classification Approach for SOA Vulnerabilities. In *Proc. Intl. Computer Software and Applications Conf.*, pages 439–444, 2009. ISBN 9780769537269. doi:10.1109/COMPSAC.2009.173.

[67] H Lufei and W Shi. Fractal: A mobile code based framework for dynamic application protocol adaptation in pervasive computing. In *19th IEEE International Parallel and Distributed Processing Symposium*, 2005.

[68] Matthew L Massie, Brent N Chun, and David E Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004. ISSN 01678191. doi:10.1016/j.parco.2004.04.001.

[69] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.

[70] Mukhtiar Memon, Michael Hafner, and Ruth Breu. Sectissimo: a platform-independent framework for security services. In *Modeling Security Workshop at MODELS*, 2008.

[71] Nathalie Mitton, Symeon Papavassiliou, Antonio Puliafito, and Kishor S Trivedi. Combining Cloud and sensors in a smart city environment. *EURASIP Journal on Wireless Communications and Networking*, 2012(1):247–257, 2012. ISSN 1687-1499. doi:10.1186/1687-1499-2012-247.

[72] Catherine E A Mulligan and Magnus Olsson. Architectural implications of smart city business models: an evolutionary perspective. *IEEE Communications Magazine*, 51(6): 80–85, jun 2013. ISSN 0163-6804. doi:10.1109/MCOM.2013.6525599.

[73] Aitor Murguzur, Johannes M Schleicher, Hong-Linh Truong, Salvador Trujillo, and Schahram Dustdar. Drain: an engine for quality-of-result driven process-based data analytics. In *International Conference on Business Process Management*, pages 349–356. Springer International Publishing, 2014.

[74] Taewoo Nam and Theresa a. Pardo. Conceptualizing smart city with dimensions of technology, people, and institutions. In *Proceedings of the 12th Annual International Digital Government Research Conference on Digital Government Innovation in Challenging Times - dg.o '11*, pages 282–291, 2011. ISBN 9781450307628. doi:10.1145/2037556.2037602.

[75] Taewoo Nam and Theresa A Pardo. Conceptualizing smart city with dimensions of technology, people, and institutions. In *Proceedings of the 12th annual international digital government research conference: digital government innovation in challenging times*, pages 282–291. ACM, 2011.

[76] Milind Naphade, Guruduth Banavar, Colin Harrison, Jurij Paraszczak, and Robert Morris. Smarter Cities and Their Innovation Challenges. *Computer*, 44(6):32–39, 2011. doi:10.1109/MC.2011.187.

[77] Stephen Nelson-Smith. *Test-Driven Infrastructure with Chef*. O'Reilly Media, 2 edition, 2014. ISBN 9781449372200.

[78] Sam Newman. *Building Microservices*. O'Reilly Media, Inc., February 2015. ISBN 1491950315.

[79] Huu Nghia Nguyen, Pascal Poizat, and Fatiha Zaidi. Automatic skeleton generation for data-aware service choreographies. In *24th International Symposium on Software Reliability Engineering (ISSRE)*, pages 320–329. IEEE, 2013. ISBN 978-1-4799-2366-3. doi:10.1109/ISSRE.2013.6698885. http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6698885.

[80] Michael P. Papazoglou and Willem Jan Van Den Heuvel. Blueprinting the cloud. *Internet Computing, IEEE*, 15(6):74–79, 2011. ISSN 10897801. doi:10.1109/MIC.2011.147.

[81] Riccardo Petrolo, Valeria Loscrì, and Nathalie Mitton. Towards a smart city based on cloud of things, a survey on the smart city vision and paradigms. *Transactions on Emerging Telecommunications Technologies*, 2015.

[82] B Rajkumar, CS Yeo, S Venugopal, J Broberg, and I Brandic. Cloud computing and emerging it platforms. *Future Generation Computer Systems. Elsevier Press, Inc*, 2009.

[83] Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In *Semantic Web Services and Web Process Composition*, pages 43–54. Springer, 2005.

[84] Luis Sanchez, Luis Muñoz, Jose Antonio Galache, Pablo Sotres, Juan R. Santana, Veronica Gutierrez, Rajiv Ramdhany, Alex Gluhak, Srdjan Krco, Evangelos Theodoridis, and Dennis Pfisterer. SmartSantander: IoT experimentation over a smart city testbed. *Computer*

Networks, 61(November):217–238, 2014. ISSN 13891286. doi:10.1016/j.bjp.2013.12.020. http://www.sciencedirect.com/science/article/pii/S1389128613004337.

[85] Benjamin Satzger, Waldemar Hummer, Philipp Leitner, and Schahram Dustdar. Esc: Towards an elastic stream computing platform for the cloud. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 348–355. IEEE, 2011.

[86] Benjamin Satzger, Waldemar Hummer, Christian Inzinger, Philipp Leitner, and Schahram Dustdar. Winds of change: From vendor lock-in to the meta cloud. *Internet Computing, IEEE*, 17(1):69–73, 2013. ISSN 10897801. doi:10.1109/MIC.2013.19.

[87] J. M. Schleicher, M. Vögler, S. Dustdar, and C. Inzinger. Application architecture for the internet of cities: Blueprints for future smart city applications. *IEEE Internet Computing*, 20(6):68–75, Nov 2016. doi:10.1109/MIC.2016.130.

[88] Johannes M Schleicher, Michael Vögler, Christian Inzinger, and Schahram Dustdar. Smart fabric-an infrastructure-agnostic artifact topology deployment framework. In *2015 IEEE International Conference on Mobile Services*, pages 320–327. IEEE, 2015.

[89] Johannes M Schleicher, Michael Vögler, Christian Inzinger, and Schahram Dustdar. Towards the internet of cities: A research roadmap for next-generation smart cities. In *Proceedings of the ACM First International Workshop on Understanding the City with Urban Informatics*, pages 3–6. ACM, 2015.

[90] Johannes M Schleicher, Michael Vögler, Christian Inzinger, Waldemar Hummer, and Schahram Dustdar. Nomads-enabling distributed analytical service environments for the smart city domain. In *Web Services (ICWS), 2015 IEEE International Conference on*, pages 679–685. IEEE, 2015.

[91] Johannes M Schleicher, Michael Vögler, Schahram Dustdar, and Christian Inzinger. Enabling a smart city application ecosystem: requirements and architectural aspects. *IEEE Internet Computing*, 20(2):58–65, 2016.

[92] Johannes M Schleicher, Michael Vögler, Christian Inzinger, and Schahram Dustdar. Modeling and management of usage-aware distributed datasets for global smart city application ecosystems. *PeerJ Computer Science, under review*, 2016.

[93] Johannes M Schleicher, Michael Vögler, Christian Inzinger, and Schahram Dustdar. Smart brix—a continuous evolution framework for container application deployments. *PeerJ Computer Science*, 2:e66, 2016.

[94] Johannes M Schleicher, Michael Vögler, Christian Inzinger, Sara Fritz, Manuel Ziegler, Thomas Kaufmann, Dominik Bothe, Julia Forster, and Schahram Dustdar. A holistic, interdisciplinary decision support system for sustainable smart city design. In *International Conference on Smart Cities*, pages 1–10. Springer International Publishing, 2016.

[95] Mohamed Sellami, Sami Yangui, Mohamed Mohamed, and Samir Tata. PaaS-independent provisioning and management of applications in the cloud. In *Proceedings of the IEEE International Conference on Cloud Computing*, pages 693–700. IEEE, 2013. ISBN 978-0-7695-5028-2. doi:10.1109/CLOUD.2013.105.

[96] N Serrano, J Hernantes, and G Gallardo. Service-Oriented Architecture and Legacy Systems. *Software, IEEE*, 31(5):15–19, 2014. doi:10.1109/MS.2014.125.

[97] H Shahriar and M Zulkernine. Automatic Testing of Program Security Vulnerabilities. In *Proc. Intl. Computer Software and Applications Conf.*, volume 2, pages 550–555. IEEE, 2009. ISBN 978-0-7695-3726-9. doi:10.1109/COMPSAC.2009.191.

[98] Shashi Shekhar, Viswanath Gunturi, Michael R Evans, and KwangSoo Yang. Spatial big-data challenges intersecting mobility and cloud computing. In *Proceedings of the Eleventh ACM International Workshop on Data Engineering for Wireless and Mobile Access*, pages 1–6. ACM, 2012.

[99] Gabriel Costa Silva, Louis M. Rose, and Radu Calinescu. A Systematic Review of Cloud Lock-In Solutions. In *Proceedings of the IEEE 5th International Conference on Cloud Computing Technology and Science*, pages 363–368, 2013. ISBN 978-0-7695-5095-4. doi:10.1109/CloudCom.2013.130.

[100] Gabriel Costa Silva, Louis M. Rose, and Radu Calinescu. Towards a Model-Driven Solution to the Vendor Lock-In Problem in Cloud Computing. In *Proceedings of the 5th International Conference on Cloud Computing Technology and Science*, pages 711–716. IEEE, 2013. ISBN 978-0-7695-5095-4. doi:10.1109/CloudCom.2013.131.

[101] Evren Sirin, James Hendler, and Bijan Parsia. Semi-automatic composition of web services using semantic descriptions. In *1st Workshop on Web Services: Modeling, Architecture and Infrastructure*, pages 17–24, 2003.

[102] Stephen Soltesz, Herbert Pötzl, Marc E Fiuczynski, Andy Bavier, and Larry Peterson. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. *SIGOPS Oper. Syst. Rev.*, 41(3):275–287, 2007. doi:10.1145/1272998.1273025.

[103] John Stankovic et al. Research directions for the internet of things. *Internet of Things Journal, IEEE*, 1(1):3–9, 2014.

[104] A Tosatto, P Ruiu, and A Attanasio. Container-Based Orchestration in Cloud: State of the Art and Challenges. In *Proc. Intl. Conf. on Complex, Intelligent, and Software Intensive Systems*, pages 70–75. IEEE, 2015. ISBN 978-1-4799-8870-9. doi:10.1109/CISIS.2015.35.

[105] H. L. Truong and S. Dustdar. Principles of software-defined elastic systems for big data analytics. In *2014 IEEE International Conference on Cloud Engineering*, pages 562–567, March 2014. doi:10.1109/IC2E.2014.67.

[106] Michael Vögler. Efficient iot application delivery and management in smart city environments, Dissertation, Technische Universität Wien, 2016.

[107] Michael Vögler, Fei Li, Markus Claeßens, Johannes M Schleicher, Sanjin Sehic, Stefan Nastic, and Schahram Dustdar. Colt collaborative delivery of lightweight iot applications. In *Internet of Things. User-Centric IoT*, pages 265–272. Springer International Publishing, 2015.

[108] Michael Vögler, Johannes Schleicher, Christian Inzinger, Stefan Nastic, Sanjin Sehic, and Schahram Dustdar. Leonore–large-scale provisioning of resource-constrained iot deployments. In *Service-Oriented System Engineering (SOSE), 2015 IEEE Symposium on*, pages 78–87. IEEE, 2015.

[109] Michael Vögler, Johannes M Schleicher, Christian Inzinger, and Schahram Dustdar. Dianedynamic iot application deployment. In *2015 IEEE International Conference on Mobile Services*, pages 298–305. IEEE, 2015.

[110] Michael Vögler, Johannes M Schleicher, Christian Inzinger, and Schahram Dustdar. Ahab: A cloud-based distributed big data analytics framework for the internet of things. *Software: Practice and Experience*, 2016.

[111] Michael Vögler, Johannes M Schleicher, Christian Inzinger, and Schahram Dustdar. A scalable framework for provisioning large-scale iot deployments. *ACM Transactions on Internet Technology (TOIT)*, 16(2):11, 2016.

[112] Michael Vögler, Johannes M Schleicher, Christian Inzinger, Schahram Dustdar, and Rajiv Ranjan. Migrating smart city applications to the cloud. *IEEE Cloud Computing*, 3(2): 72–79, 2016.

[113] Michael Vögler, Johannes M Schleicher, Christian Inzinger, Bernhard Nickel, and Schahram Dustdar. Non-intrusive monitoring of stream processing applications. In *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 162–171. IEEE, 2016.

[114] Heimo Walter and Anton Glaninger. Berechnung von rohrnetzwerken mit baumstruktur. *KI. Luft-und Kältetechnik*, 40(11):460–464, 2004.

[115] Jiafu Wan, Di Li, Caifeng Zou, Keliang Zhou, Information Technology, Jiafu Wan, Di Li, Caifeng Zou, and Keliang Zhou. M2M Communications for Smart City: An Event-Based Architecture. In *Computer and Information Technology (CIT), 2012 IEEE 12th International Conference on*, pages 895–900, oct 2012. ISBN 9780769548586. doi:10.1109/CIT.2012.188.

[116] P. Wang, Z. Ding, C. Jiang, and M. Zhou. Constraint-Aware Approach to Web Service Composition. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 44(6):770–784, June 2014. ISSN 2168-2216. doi:10.1109/TSMC.2013.2280559. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6637082.

[117] Shiqiang Wang, Rahul Urgaonkar, Murtaza Zafer, Ting He, Kevin Chan, and Kin K. Leung. Dynamic service migration in mobile edge-clouds. *Proceedings of 2015 14th IFIP Networking Conference, IFIP Networking 2015*, 2015. ISSN 01665316. doi:10.1109/IFIPNetworking.2015.7145316.

[118] Johannes Wettinger, Uwe Breitenbücher, and Frank Leymann. Compensation-Based vs. Convergent Deployment Automation for Services Operated in the Cloud. In *Service-Oriented Computing*, volume 8831 of *Lecture Notes in Computer Science*, pages 336–350. Springer, 2014. ISBN 978-3-662-45390-2. doi:10.1007/978-3-662-45391-9{_}23.

[119] Lamia Youseff, Maria Butrico, and Dilma Da Silva. Toward a unified ontology of cloud computing. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. IEEE, 2008.

[120] W.D. Yu, D. Aravind, and P. Supthaweesuk. Software Vulnerability Analysis for Web Services Software Systems. In *Proc. Symposium on Computers and Communications*, pages 740–748. IEEE, 2006. ISBN 0-7695-2588-1. doi:10.1109/ISCC.2006.151.

[121] Dazhi Zhang, Donggang Liu, Christoph Csallner, David Kung, and Yu Lei. A distributed framework for demand-driven software vulnerability detection. *Journal of Systems and Software*, 87:60–73, 2014.

[122] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, 2010. ISSN 1867-4828. doi:10.1007/s13174-010-0007-6.

[123] Haiyan Zhao and Hongxia Tong. A Dynamic Service Composition Model Based on Constraints. In *6th Int. Conf. on Grid and Cooperative Computing (GCC)*, pages 659–662, 2007. ISBN 0-7695-2871-6. doi:10.1109/GCC.2007.7. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4293843.

APPENDIX A

Curriculum Vitae

Johannes Michael Schleicher
Keutschacherstraße 116
9220 Velden, Austria

Born July 6, 1982
Email schleicher@dsg.tuwien.ac.at
Web http://dsg.tuwien.ac.at/staff/jschleicher

Work Experience

Project Assistant at the Distributed Systems Group TU Wien http://dsg.tuwien.ac.at/	2014 – 2016
Chief Information Officer Simpliflow GmbH http://www.simpliflow.com/	2011 – 2013
Project Manager and Consultant ilogs GmbH http://www.ilogs.com	2011
Chief Executive Officer and Founding Partner Blackwhale GmbH http://www.blackwhale.at	2008 – 2011
Consultant, Freelance Software Engineer Frequentis AG http://www.frequentis.com ilogs GmbH http://www.ilogs.com	2006 – 2008
Senior Software Engineer ilogs GmbH http://www.ilogs.com	2004 – 2008

Education

Ph.D. in Computer Science at the Distributed Systems Group, TU Wien	2013 – 2017
Dipl.-Ing. (M.Sc.) in Software Engineering & Internet Computing, TU Wien	2006 – 2010
B.Sc. in Software & Information Engineering, TU Wien	2001 – 2004

Publications

Journal Papers

- J. M. Schleicher, M. Vögler, S. Dustdar, and C. Inzinger. Application architecture for the internet of cities: Blueprints for future smart city applications. *IEEE Internet Computing*, 20(6):68–75, Nov 2016. doi:10.1109/MIC.2016.130
- Johannes M Schleicher, Michael Vögler, Christian Inzinger, and Schahram Dustdar. Smart brix—a continuous evolution framework for container application deployments. *PeerJ Computer Science*, 2:e66, 2016
- Michael Vögler, Johannes M Schleicher, Christian Inzinger, and Schahram Dustdar. Ahab: A cloud-based distributed big data analytics framework for the internet of things. *Software: Practice and Experience*, 2016
- Michael Vögler, Johannes M Schleicher, Christian Inzinger, Schahram Dustdar, and Rajiv Ranjan. Migrating smart city applications to the cloud. *IEEE Cloud Computing*, 3(2): 72–79, 2016
- Johannes M Schleicher, Michael Vögler, Schahram Dustdar, and Christian Inzinger. Enabling a smart city application ecosystem: requirements and architectural aspects. *IEEE Internet Computing*, 20(2):58–65, 2016
- Michael Vögler, Johannes M Schleicher, Christian Inzinger, and Schahram Dustdar. A scalable framework for provisioning large-scale iot deployments. *ACM Transactions on Internet Technology (TOIT)*, 16(2):11, 2016

Conference/Workshop Proceedings

2016

- Johannes M Schleicher, Michael Vögler, Christian Inzinger, Sara Fritz, Manuel Ziegler, Thomas Kaufmann, Dominik Bothe, Julia Forster, and Schahram Dustdar. A holistic, interdisciplinary decision support system for sustainable smart city design. In *International Conference on Smart Cities*, pages 1–10. Springer International Publishing, 2016

- Michael Vögler, Johannes M Schleicher, Christian Inzinger, Bernhard Nickel, and Schahram Dustdar. Non-intrusive monitoring of stream processing applications. In *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 162–171. IEEE, 2016

2015

- Johannes M Schleicher, Michael Vögler, Christian Inzinger, and Schahram Dustdar. Towards the internet of cities: A research roadmap for next-generation smart cities. In *Proceedings of the ACM First International Workshop on Understanding the City with Urban Informatics*, pages 3–6. ACM, 2015

- Johannes M Schleicher, Michael Vögler, Christian Inzinger, and Schahram Dustdar. Smart fabric-an infrastructure-agnostic artifact topology deployment framework. In *2015 IEEE International Conference on Mobile Services*, pages 320–327. IEEE, 2015

- Michael Vögler, Johannes M Schleicher, Christian Inzinger, and Schahram Dustdar. Dianedynamic iot application deployment. In *2015 IEEE International Conference on Mobile Services*, pages 298–305. IEEE, 2015

- Johannes M Schleicher, Michael Vögler, Christian Inzinger, Waldemar Hummer, and Schahram Dustdar. Nomads-enabling distributed analytical service environments for the smart city domain. In *Web Services (ICWS), 2015 IEEE International Conference on*, pages 679–685. IEEE, 2015

- Michael Vögler, Johannes Schleicher, Christian Inzinger, Stefan Nastic, Sanjin Sehic, and Schahram Dustdar. Leonore–large-scale provisioning of resource-constrained iot deployments. In *Service-Oriented System Engineering (SOSE), 2015 IEEE Symposium on*, pages 78–87. IEEE, 2015

- Michael Vögler, Fei Li, Markus Claeßens, Johannes M Schleicher, Sanjin Sehic, Stefan Nastic, and Schahram Dustdar. Colt collaborative delivery of lightweight iot applications. In *Internet of Things. User-Centric IoT*, pages 265–272. Springer International Publishing, 2015

2014

- Aitor Murguzur, Johannes M Schleicher, Hong-Linh Truong, Salvador Trujillo, and Schahram Dustdar. Drain: an engine for quality-of-result driven process-based data analytics. In *International Conference on Business Process Management*, pages 349–356. Springer International Publishing, 2014

http://dsg.tuwien.ac.at/staff/jschleicher