An Introduction to Self-Adaptive Systems

A Contemporary Software Engineering Perspective

Danny Weyns
Katholieke Universiteit Leuven, Belgium

WILEY

IEEE PRESS

This edition first published 2021
© 2021 John Wiley & Sons Ltd

Registered Offices
John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, USA
John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, UK

Editorial Office
The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, UK

For details of our global editorial offices, customer services, and more information about Wiley products visit us at www.wiley.com.

Wiley also publishes its books in a variety of electronic formats and by print-on-demand. Some content that appears in standard print versions of this book may not be available in other formats.

Library of Congress Cataloging-in-Publication Data applied for,

Hardback ISBN: 9781119574941

Cover Design: Wiley
Cover Image: © Takeshi.K/Getty Images

Set in 9.5/12.5pt STIXTwoText by SPi Global, Chennai, India

Printed and bound by CPI Group (UK) Ltd, Croydon, CR0 4YY

10 9 8 7 6 5 4 3 2 1

Contents

first time it will provide a broad view of what is now known and practiced. For the experienced professional, it will provide concrete examples and techniques that can be put into practice. For the researcher, it will provide a structured view of the important prior work and of the open challenges facing the field.

February 2020

David Garlan
Professor, School of Computer Science
Carnegie Mellon University

Acknowledgments

This book has been developed in two stages over the past three years. The following colleagues provided me with particularly useful feedback that helped me to improve preliminary versions of this book: Carlo Ghezzi (Politecnico di Milano), Jeff Kramer (Imperial Collage London), Bradley Schmerl (Carnegie Mellon University Pittsburg), Thomas Vogel (Humboldt University Berlin), Gabriel A. Moreno (Carnegie Mellon University), Martina Maggio (Lund University), Antonio Filieri (Imperial Collage London), Marin Litoiu (York University), Vitor E. Silva Souza (Federal University of Espírito Santo), Radu Calinescu (University of York), Jeff Kephart (IBM T. J. Watson Research Center), Betty H.C. Cheng (Michigan State University), Nelly Bencomo (Aston University), Javier Camara Moreno (University of York), John Mylopoulos (University of Toronto), Sebastian Uchitel (University of Buenos Aires), Pooyan Jamshidi Dermani (University of South Carolina), Simos Gerasimou (University of York), Kenji Tei (Waseda University), Dimitri Van Landuyt (Katholieke Universiteit Leuven), Panagiotis (Panos) Patros (University of Waikato), Raffaela Mirandola (Polytechnic University of Milan), and Paola Inverardi (University of L'Aquila). These people have suggested improvements and pointed out mistakes. I thank everyone for providing me with very helpful comments.

I thank the members of the imec-Distrinet research group for their support. I am particularly thankful to my colleague Danny Hughes, his former student Gowri Sankar Ramachandran, and the members of the Network task force for sharing their expertise on the Internet-of-Things. I want to express my sincere appreciation to my colleagues at Linnaeus University, in particular Jesper Andersson, for their continuous support.

I thank M. Usman Iftikhar, Stepan Shevtsov, Federico Quin, Omid Gheibi, Sara Mahdavi Hezavehi, Angelika Musil, Juergen Musil, Nadeem Abbas, and the other students I worked with at KU Leuven and Linnaeus University for their inspiration and collaboration.

I express my sincere appreciation to the monks of the abbeys of West-Vleteren, Westmalle, Tongerlo, and Orval for their hospitality during my stays when working on this book.

Finally, I express my gratitude to Wiley for their support with the publication of this manuscript.

Danny Weyns

Acronyms

24/7	24 hours a day, seven days a week: all the time
A-LTL	Adapt operator-extended Linear Temporal Logic
ActivFORMS	Active FOrmal Models for Self-adaptation
Amazon EC2	Amazon Elastic Compute Cloud
AMOCS-MA	Automated Multi-objective Control of Software with Multiple Actuators
AP	Atomic Propositions
C	Coulomb
CD-ROM	Compact Disk Read Only Memory
CPU	Central Processing Unit
CTL	Computation Tree Logic
dB	deciBel
DCRG	Dynamic Condition Response Graph
DeltaIoT.v2	Advanced version of DeltaIoT
DeltaIoT	IoT application for building security monitoring
DiVA	Dynamic Variability in complex Adaptive systems
DTMC	Discrete-Time Markov Chain
ENTRUST	ENgineering of TRUstworthy Self-adaptive sofTware
EUREMA	ExecUtable RuntimE MegAmodels
F1-score	Score that combines precision and recall to evaluate a classifier
FLAGS	Fuzzy Live Adaptive Goals for Self-adaptive systems
FORMS	FOrmal Reference Model for Self-adaptation
FQL4KE	Fuzzy Q-Learning for Knowledge Evolution
FUSION	FeatUre-oriented Self-adaptatION
GDPR	General Data Protection Regulation
GORE	Goal-Oriented Requirements Engineering
IBM	International Business Machines Corporation
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet-of-Things
ISO	International Organization for Standardization
KAMI	Keep Alive Models with Implementation
KAOS	Knowledge Acquisition in Automated Specification
LTS	Labeled Transition System
MAPE-K	Monitor-Analyze-Plan-Execute-Knowledge

Introducing self-adaptive systems is challenging given the diversity of research topics, engineering methods, and application domains that are part of this field. To tackle this challenge, this text is based on six pillars.

First, we lay a foundation for *what* constitutes a self-adaptive system by introducing two generally acknowledged, but complementary basic principles. These two principles enable us to characterize self-adaptive systems and distinguish them from other related types of systems. From the basic principles, a conceptual model of a self-adaptive system is derived, which offers a basic vocabulary that we use throughout the text.

Second, the core of the text, which focuses on *how* self-adaptive systems are engineered, is partitioned into convenient chunks driven by research and engineering efforts over time. In particular, the text approaches the engineering of self-adaptive systems in seven waves. These waves put complementary aspects of engineering self-adaptive systems in focus that synergistically have contributed to the current body of knowledge in the field. Each wave highlights a trend of interest in the research community. Some of the earlier waves have stabilized now and resulted in common knowledge in the community. Other more recent waves are still very active and the subject of debate; the knowledge of these waves has not been fully consolidated yet.

Third, throughout the text we use a well-thought-out set of applications to illustrate the material with concrete examples. We use a simple service-based application to illustrate the basic principles and the conceptional model of self-adaptive systems. Before the core part of the text that zooms in on the seven waves of research on engineering self-adaptive systems, we introduce a practical Internet-of-Things application that we use as the main case to illustrate the characteristics of the different waves. In addition, we use a variety of cases from different contemporary domains to illustrate the material, including a client-server system, a mobile service, a geo-localization service, unmanned vehicles, video compression, different Web applications, and a Cloud system.

Fourth, each core chapter of the book starts with a list of learning outcomes at different orders of thinking (from understanding to synthesis) and concludes with a series of exercises. The exercises are defined at four different levels of complexity, characterized by four letters that refer to the expected average time required for solving the exercises. Level *H* requires a basic understanding of the material of the chapter; the exercises should be solvable in a number of person-hours. Level *D* requires in depth understanding of the material of the chapter; these exercises should be solvable within person-days. Level *W* requires the study of some additional material beyond the material in the chapter; these exercises should be solvable within person-weeks. Finally, level *M* requires the development of novel solutions based on the material provided in the corresponding chapter; these exercises require an effort of person-months. The final chapter discusses the maturity of the field and outlines open challenges for research in self-adaptation, which can serve as further inspiration for future research endeavors, for instance as a start point for PhD projects.

Fifth, each chapter concludes with bibliographic notes. These notes point to foundational research papers of the different parts of the chapter. In addition, the notes highlight some characteristic work and provide pointers to background material. The material referred to in the bibliographic notes is advised for further reading.

Sixth, supplementary material is freely available for readers, students, and teachers at the book website: **https://introsas.cs.kuleuven.be/**. The supplementary material includes slides for educational purposes, selected example solutions of exercises, models and code that can be used for the exercises, and complementary material that elaborates on specific material from the book.

As such, this manuscript provides a starting point for students, researchers, and engineers that want to familiarize themselves with the field of self-adaptation. The text aims to offer a solid basis for those who are interested in self-adaptation to obtain the required skill set to understand the fundamental principles and engineering methods of self-adaptive systems.

The principles of self-adaptation have their roots in software architecture, model-based engineering, formal specification languages, and principles of control theory and machine learning. It is expected that readers are familiar with the basics of these topics when starting with our book, although some basic aspects are introduced in the respective chapters.

1

Basic Principles of Self-Adaptation and Conceptual Model

Modern software-intensive systems[1] are expected to operate under uncertain conditions, without interruption. Possible causes of uncertainties include changes in the operational environment, dynamics in the availability of resources, and variations of user goals. Traditionally, it is the task of system operators to deal with such uncertainties. However, such management tasks can be complex, error-prone, and expensive. The aim of self-adaptation is to let the system collect additional data about the uncertainties during operation in order to manage itself based on high-level goals. The system uses the additional data to resolve uncertainties and based on its goals re-configures or adjusts itself to satisfy the changing conditions.

Consider as an example a simple service-based health assistance system as shown in Figure 1.1. The system takes samples of vital parameters of patients; it also enables patients to invoke a panic button in case of an emergency. The parameters are analyzed by a medical service that may invoke additional services to take actions when needed; for instance, a drug service may need to notify a local pharmacy to deliver new medication to a patient. Each service type can be realized by one of multiple service instances provided by third-party service providers. These service instances are characterized by different quality properties, such as failure rate and cost. Typical examples of uncertainties in this system are the patterns that particular paths in the workflow are invoked by, which are based on the health conditions of the users and their behavior. Other uncertainties are the available service instances, their actual failure rates and the costs to use them. These parameters may change over time, for instance due to the changing workloads or unexpected network failures.

Anticipating such uncertainties during system development, or letting system operators deal with them during operation, is often difficult, inefficient, or too costly. Moreover, since many software-intensive systems today need to be operational 24/7, the uncertainties necessarily need to be resolved at runtime when the missing knowledge becomes available. Self-adaptation is about how a system can mitigate such uncertainties autonomously or with minimum human intervention.

The basic idea of self-adaptation is to let the system collect new data (that was missing before deployment) during operation when it becomes available. The system uses the

1 A software-intensive system is any system where software dominates to a large extent the design, construction, deployment, operation, and evolution of the system. Some examples include mobile embedded systems, unmanned vehicles, web service applications, wireless ad-hoc systems, telecommunications, and Cloud systems.

An Introduction to Self-Adaptive Systems: A Contemporary Software Engineering Perspective,
First Edition. Danny Weyns.

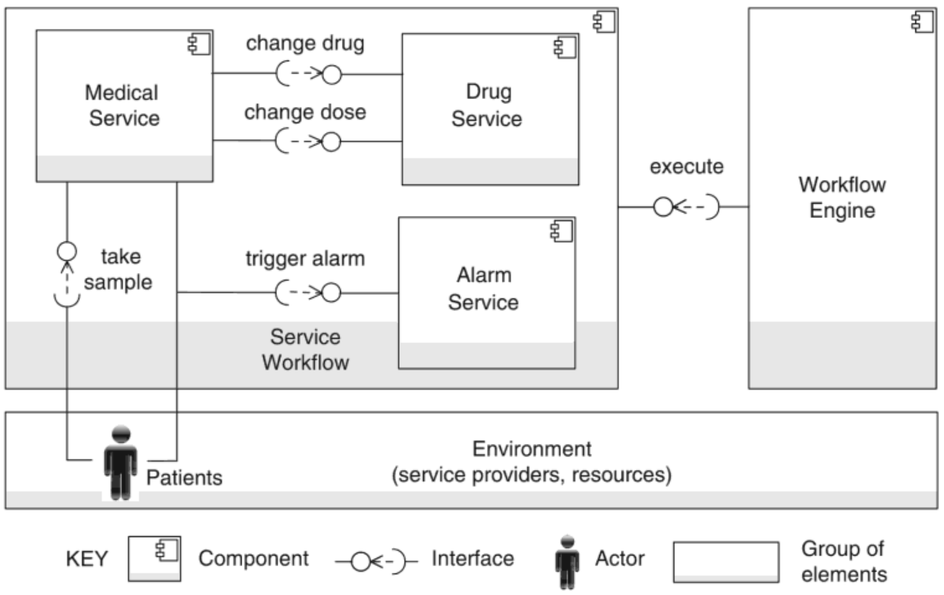

Figure 1.1 Architecture of a simple service-based health assistance system

additional data to resolve uncertainties, to reason about itself, and based on its goals to reconfigure or adjust itself to maintain its quality requirements or, if necessary, to degrade gracefully.

In this chapter, we explain *what* a self-adaptive system is. We define two basic principles that determine the essential characteristics of self-adaptation. These principles allow us to define the boundaries of what we mean by a self-adaptive system in this book, and to contrast self-adaptation with other approaches that deal with changing conditions during operation. From the two principles, we derive a conceptual model of a self-adaptive system that defines the basic elements of such a system. The conceptual model provides a basic vocabulary for the remainder of this book.

LEARNING OUTCOMES

- To explain the basic principles of self-adaptation.
- To understand how self-adaptation relates to other adaptation approaches.
- To describe the conceptual model of a self-adaptive system.
- To explain and illustrate the basic concepts of a self-adaptive system.
- To apply the conceptual model to a concrete self-adaptive application.

1.1 Principles of Self-Adaptation

There is no general agreement on a definition of the notion of *self-adaptation*. However, there are two common interpretations of what constitutes a self-adaptive system.

which a self-aware system uses to reason at runtime, enabling it to act in accordance with higher-level goals.

1.3 Scope of Self-Adaptation

Autonomous systems, multi-agent systems, self-organizing systems, and context-aware systems are families of systems that apply classical approaches to deal with change at runtime. However, these approaches do not align with the combined basic principles of self-adaptation. In particular, none of these approaches comply with the second principle, which makes an explicit distinction between a part of the system that handles domain concerns and a part that handles adaptation concerns. However, the second principle of self-adaptation can be applied to each of these approaches – i.e. these systems can be enhanced with a feedback loop that deals with a set of adaptation concerns. This book is concerned with self-adaptation as a property of a computing system that is compliant with the two basic principles of self-adaptation.

Furthermore, self-adaptation can be applied at different levels of the software stack of computing systems, from the underlying resources and low-level computing infrastructure to middleware services and application software. The challenges of self-adaptation at these different levels are different. For instance, the space of adaptation options of higher-level software entities is often multi-dimensional, and software qualities and adaptation goals usually have a complex interplay. These characteristics are less applicable to the adaptation of lower-level resources, where there is often a more straightforward relation between adaptation actions and software qualities. In this book, we consider self-adaptation applied at different levels of the software stack of computing systems, from virtualized resources up to application software.

1.4 Conceptual Model of a Self-Adaptive System

Starting from the two basic principles of self-adaptation, we define a conceptual model for self-adaptive systems that describes the basic elements of such systems and the relationship between them. The basic elements are intentionally kept abstract and general, but they are compliant with the basic principles of self-adaptation. The conceptual model introduces a basic vocabulary for the field of self-adaptation that we will use throughout this book. Figure 1.2 shows the conceptual model of a self-adaptive system.

The conceptual model comprises *four basic elements:* environment, managed system, feedback loop, and adaptation goals. The feedback loop together with the adaptation goals form the managing system. We discuss the elements one by one and illustrate them for the service-based health assistance application.

1.4.1 Environment

The environment refers to the part of the external world with which a self-adaptive system interacts and in which the effects of the system will be observed and evaluated. The environment can include users as well as physical and virtual elements. The distinction between

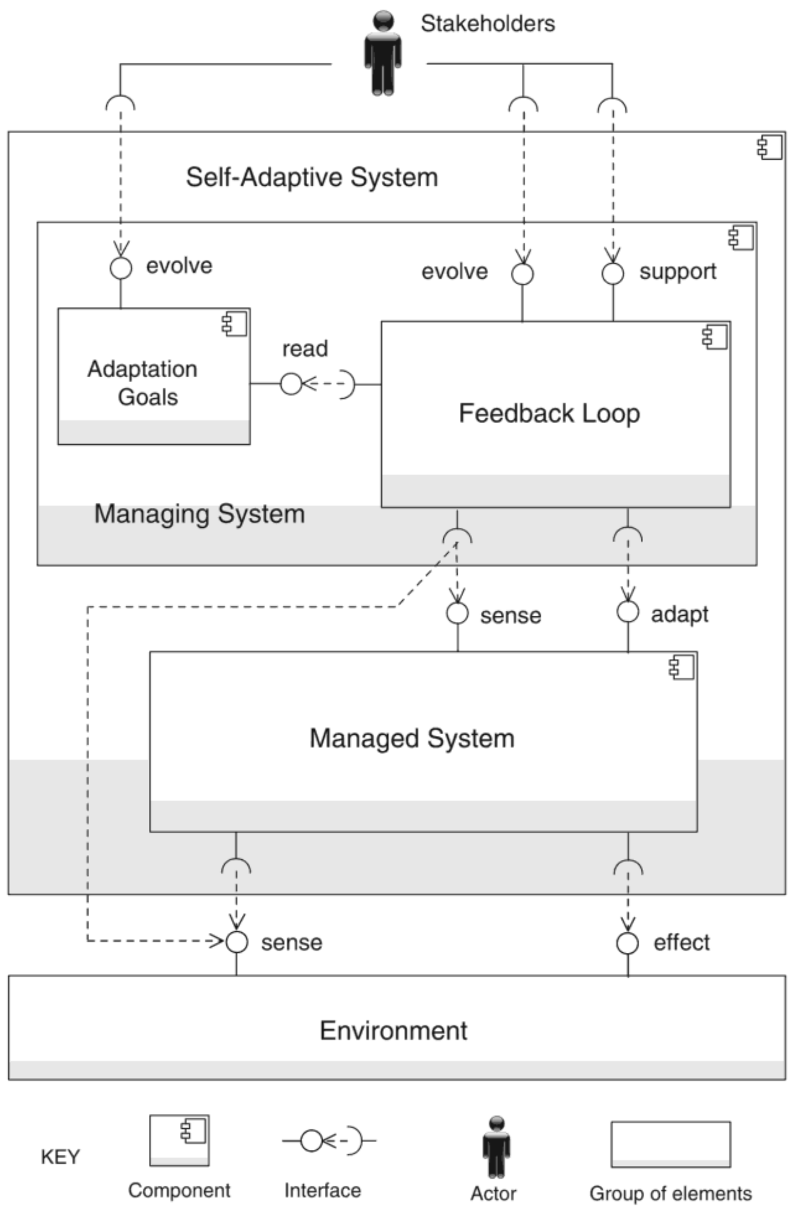

Figure 1.2 Conceptual model of a self-adaptive system

the environment and the self-adaptive system is made based on the extent of control. The environment can be sensed and effected through sensors and effectors, respectively. However, as the environment is not under the control of the software engineer of the system, there may be uncertainty in terms of what is sensed by the sensors or what the outcomes will be of the effectors.

Applied to the service-based health assistance system example, the environment includes the patients that make use of the system; the application devices with the sensors that measure vital parameters of patients and the panic buttons; the service providers with the services instances they offer; and the network connections used in the system, which may all affect the quality properties of the system.

1.4.2 Managed System

The managed system comprises the application software that realizes the functions of the system to its users. Hence, the concerns of the managed system are concerns over the domain, i.e. the environment of the system. Different terminology has been used to refer to the managed system, such as managed element, system layer, core function, base-level system, and controllable plant. In this book, we systematically use the term *managed system*. To realize its functions to the users, the managed system senses and effects the environment. To support adaptations, the managed system needs to be equipped with sensors to enable monitoring and effectors (also called actuators) to execute adaptation actions. Safely executing adaptations requires that actions applied to the managed systems do not interfere with the regular system activity. In general, they may affect ongoing activities of the system – for instance, scaling a Cloud system might require bringing down a container and restarting it.

A classic approach to realizing safe adaptations is to apply adaptation actions only when a system (or the parts that are subject to adaptation) is in a *quiescent state*. A quiescent state is a state where no activity is going on in the managed system or the parts of it that are subject to adaptation so that the system can be safely updated. Support for quiescence requires an infrastructure to deal with messages that are invoked during adaptations; this infrastructure also needs to handle the state of the adapted system or the relevant parts of it to ensure its consistency before and after adaptation. Handling such messages and ensuring consistency of state during adaptations are in general difficult problems. However, numerous infrastructures have been developed to support safe adaptations for particular settings. A well-known example is the OSGi (Open Service Gateway Initiative) Java framework, which supports installing, starting, stopping, and updating arbitrary components (bundles in OSGi terminology) dynamically.

The managed system of the service-based health assistance system consists of a service workflow that realizes the system functions. In particular, a medical service receives messages from patients with values of their vital parameters. The service analyzes the data and either invokes a drug service to notify a local pharmacy to deliver new medication to the patient or change the dose of medication, or it invokes an alarm service in case of an emergency to notify medical staff to visit the patient. The alarm service can also be invoked directly by a patient via a panic button. To support adaptation, the workflow infrastructure offers sensors to track the relevant aspects of the system and the characteristics of service instances (failure rate and cost). The infrastructure allows the selection and use of concrete instances of the different types of services that are required by the system. Finally, the workflow infrastructure needs to provide support to change service instances in a consistent manner by ensuring that a service is only removed and replaced when it is no longer involved in any ongoing service invocation of the health assistance system.

1.4.3 Adaptation Goals

Adaptation goals represent concerns of the managing system over the managed system; adaptation goals relate to quality properties of the managed system. In general, four principal types of high-level adaptation goals can be distinguished: self-configuration (i.e. systems that configure themselves automatically), self-optimization (systems that continually seek ways to improve their performance or reduce their cost), self-healing (systems that detect, diagnose, and repair problems resulting from bugs or failures), and self-protection (systems that defend themselves from malicious attacks or cascading failures).

Since the system uses the adaptation goals to reason about itself during operation, the goals need to be represented in a machine-readable format. Adaptation goals are often expressed in terms of the uncertainty they have to deal with. Example approaches are the specification of quality of service goals using probabilistic temporal logics that allow for probabilistic quantification of properties, the specification of fuzzy goals whose satisfaction is represented through fuzzy constraints, and a declarative specification of goals (in contrast to enumeration) allowing the introduction of flexibility in the specification of goals. Adaptation goals can be subject to change themselves, which is represented in Figure 1.2 by means of the *evolve* interface. Adding new goals or removing goals during operation will require updates of the managing system, and often also require updates of probes and effectors.

In the health assistance application, the system dynamically selects service instances under changing conditions to keep the failure rate over a given period below a required threshold (self-healing goal), while the cost is minimized (optimization goal). Stakeholders may change the threshold value for the failure rate during operation, which may require just a simple update of the corresponding threshold value. On the other hand, adding a new adaptation goal, for instance to keep the average response time of invocations of the assistance service below a required threshold, would be more invasive and would require an evolution of the adaptation goals and the managing system.

1.4.4 Feedback Loop

The adaptation of the managed system is realized by the managing system. Different terms are used in the literature for the concept of managing system, such as autonomic manager, adaptation engine, reflective system, and controller. Conceptually, the managing system realizes a feedback loop that manages the managed system. The feedback loop comprises the adaptation logic that deals with one or more adaptation goals. To realize the adaptation goals, the feedback loop monitors the environment and the managed system and adapts the latter when necessary to realize the adaptation goals. With a reactive policy, the feedback loop responds to a violation of the adaptation goals by adapting the managed system to a new configuration that complies with the adaptation goals. With a proactive policy, the feedback loop tracks the behavior of the managed system and adapts the system to anticipate a possible violation of the adaptation goals.

An important requirement of a managing system is ensuring that fail-safe operating modes are always satisfied. When such an operating mode is detected, the managing system can switch to a fall-back or degraded mode during operation. An example of an operating mode that may require the managing system to switch to a fail-safe configuration is the inability to find a new configuration to adapt the managed system to that achieves the adaptation goals within the time window that is available to make an adaptation decision. Note that instead of falling back to a fail-safe configuration in the event that the goals cannot be achieved, the managing system may also offer a stakeholder the possibility to decide on the action to take.

The managing system may consist of a single level that conceptually consists of one feedback loop with a set of adaptation goals, as shown in Figure 1.2. However, the managing system may also have a layered structure, where each layer conceptually consists of a feedback loop with its own goals. In this case, each layer manages the layer beneath – i.e. layer *n* manages layer *n*-1, and layer 1 manages the managed system. In practice, most self-adaptive systems have a managing system that consists of just one layer. In systems where additional layers are applied, the number of additional layers is usually limited to one or two. For instance, a managing system may have two layers: the bottom layer may react quickly to changes and adapts the managed system when needed, while the top layer may reason over long term strategies and adapt the underlying layer accordingly.

The managing system can operate completely automatically without intervention of stakeholders, or stakeholders may be involved in support for certain functions realized by the feedback loop; this is shown in Figure 1.2 by means of the generic *support* interface. We already gave an example above where a stakeholder could support the system with handling a fail-safe situation. Another example is a managing system that detects a possible threat to the system. Before activating a possible reconfiguration to mitigate the threat, the managing system may check with a stakeholder whether the adaptation should be applied or not.

The managing system can be subject to change itself, which is represented in Figure 1.2 with the *evolve* interface. On-the-fly changes of the managing systems are important for two main reasons: (i) to update a feedback loop to resolve a problem or a bug (e.g. add or replace some functionality), and (ii) to support changing adaptation goals, i.e. change or remove an existing goal or add a new goal. The need for evolving the feedback loop model is triggered by stakeholders either based on observations obtained from the executing system or because stakeholders want to change the adaptation goals.

The managing system of the service-based health assistance system comprises a feedback loop that is added to the service workflow. The task of the feedback loop is to ensure that the adaptation goals are realized. To that end, the feedback loop monitors the system behavior and the quality properties of service instances, and tracks that the system is not violating the adaptation goals. For a reactive policy, the feedback loop will select alternative service instances that ensure the adaptation goals are met in the event that goal violations are detected. If no configuration can be found that complies with the adaptation goals within a given time (fail-safe operating mode), the managing system may involve a stakeholder to decide on the adaptation action to take. The feedback loop that adapts the service instances to ensure that the adaptation goals are realized may be extended with an extra level that adapts the underlying method that makes the adaptation decisions. For instance, this extra level may track the quality properties of service instances over time and identify patterns. The second layer can then use this knowledge to instruct the underlying feedback loop to give preference to selecting particular service instances or to avoid the selection of certain instances. For instance, services that expose a high level of failures during particular periods managed system comprises the application software that realizes the domain concerns for the users. To support adaptation, the managed system needs to provide probes and effectors and support safe adaptations. The adaptation goals represent concerns over the managed system, which refer to qualities of the system. The feedback loop realizes the adaptation goals by monitoring and adapting the managed system. The feedback loop with the adaptation goals form the managing system. The managing system can be subject to on-the-fly evolution, either to update some functionality of the adaptation logic or to change the adaptation goals.

1.7 Exercises

1.1 Conceptual model pipe and filter system: level H
Consider a pipe and filter system that has to perform a series of tasks for a user. Different instances of the filters are offered by third parties. These filter instances provide different quality of service in terms of processing time and service cost that may change over time. Explain how you would make this a self-adaptive system that ensures that the average throughput of tasks remains under a given threshold while the cost is minimized. Draw the conceptual model that shows your solution to this adaptation problem.

1.2 Conceptual model Znn.com news service: level H
Setting. Consider Znn.com, a news service that serves multimedia news content to customers. Architecturally, Znn.com is set up as a Web-based client-server system that serves clients from a pool of servers. Customers of Znn.com expect a reasonable response time, while the system owner wants to keep the cost of the server pool within a certain operating budget. In normal operating circumstances, the appropriate trade-offs can be made at design-time. However, from time to time, due to highly popular events, Znn.com experiences spikes in news requests that are not within the originally designed parameters. This means that the clients will not receive content in a timely manner. To the clients, the site will appear to be down, so they may not use the service anymore, resulting in lost revenue. The challenge for self-adaptation is to enable the system to still provide content at peak times. There are several ways to deal with this, such as serving reduced content, increasing the number of servers serving content, and choosing to prioritize serving paying customers.
Task. Enhance Znn.com with self-adaptation to deal with the challenge of the news service. Identify the basic concepts of the self-adaptive system (environment, managed system, feedback loop, adaptation goals) and describe the responsibilities of each element. Draw the conceptual model that shows your solution to this adaptation problem.
Additional material. See the Znn artifact website [53].

1.3 Conceptual model video encoder: level H

Setting. Consider a video encoder that takes a stream of video frames (for instance from an mp4 video) and compresses the frames such that the video stream fits a given communication channel. While compressing frames, the encoder should maintain a required quality of the manipulated frames compared to the original frames, which is expressed as a similarity index. To achieve these conflicting goals, the encoder can change three parameters for each frame: the quality of the encoding and the setting of a sharpening filter and the setting of a noise reduction filter that are both applied to the image. The quality parameter that relates to a compression factor for the image has a value between 1 and 100, where 100 preserves all frame details and 1 produces the highest compression. However, the relationship between quality and size depends on the frame content, which is difficult to predict upfront. The sharpening filter and the noise reduction filter modify certain pixels of the imagine, for instance to remove elements that appear after compressing the original frame. The sharpening filter has a parameter with a value that ranges between 0 and 5, where 0 indicates no sharpening and 5 maximum sharpening. The noise reduction filter has a parameter that specifies the size of the applied noise reduction filter, which also varies between 0 and 5.
Task. Enhance the video encoder with self-adaptation capabilities to deal with the conflicting goals of compressing frames and ensuring a required level of quality. Identify the basic concepts of the self-adaptive system (environment, managed system, feedback loop, adaptation goals) and describe the responsibilities of each element. Draw the conceptual model that shows your solution to this adaptation problem.
Additional material. See the Self-Adaptive Video Encoder artifact website [136].

1.4 Implementation feedback loop Tele-Assistance System: level D
Setting. TAS, short for Tele-Assistance System, is a Java-based artifact that supports research and experimentation on self-adaptation. TAS simulates a health assistance service for elderly and chronically sick people, similar to the health assistance service used in this chapter. TAS uses a combination of sensors embedded in a wearable device and remote third-party services from medical analysis, pharmacy and emergency service providers. The TAS workflow periodically takes measurements of the vital parameters of a patient and employs a medical service for their analysis. The result of an analysis may trigger the invocation of a pharmacy service to deliver new medication to the patient or to change their dose of medication, or, in a critical situation, the invocation of an alarm service that will send a medical assistance team to the patient. The same alarm service can be invoked directly by the patient by using a panic button on the wearable device. In practice, the TAS service will be subject to a variety of uncertainties: services may fail, service response times may vary, or new services may become available. Different types of adaptations can be applied to deal with these uncertainties, such as switching to equivalent services, simultaneously invoking several services for equivalent operations, or changing the workflow architecture.
Task. Download the source code of TAS. Read the developers guide that is part of the artifact distribution, and prepare Eclipse to work with the artifact. Execute the TAS artifact and get familiar with it. Now design a feedback loop that deals with service failures. The first adaptation goal is a threshold goal that requires that the average

number of service failures should not exceed 10% of the invocations over 100 service invocations. The second adaptation goal is to minimize the cost for service invocations over 100 service invocations. Implement your design and test it. Evaluate your solution and assess.

Additional material. For the TAS artifact, see [201]. The latest version of TAS can be downloaded from the TAS website [212]. For background information about TAS, see [200].

1.8 Bibliographic Notes

The external principle of self-adaptation is grounded in the description of what constitutes a self-adaptive system provided in a roadmap paper on engineering self-adaptive system [50]. Y. Brun et al. complemented this description and motivated the "self" prefix indicating that the system decides autonomously [35]. The internal principle of self-adaptation is grounded in the pioneering work of P. Oreizy et al. that stressed the need for a systematic approach to deal with software modification at runtime (as opposed to ad-hoc "patches") [150]. In their seminal work on Rainbow, D. Garlan et al. contrasted internal mechanisms to adapt a system (for instance using exceptions) with external mechanisms that enhance a system with an external feedback loop that is responsible for handling adaptation [81].

Back in 1948, N. Wiener published a book that coined the term "cybernetics" to refer to self-regulating mechanisms. This work laid the theoretical foundation for several fields in autonomous systems. M. Wooldridge provided a comprehensive and readable introduction to the theory and practice of the field of multi-agent systems [215]. F. Heylighen reviewed the most important concepts and principles of self-organization [97]. Based on these principles, V. Dyke Parunak et al. demonstrated how digital pheromones enable robust coordination between unmanned vehicles [190]. T. De Wolf and T. Holvoet contrast self-organization with emergent behavior [60].

B. Schilit et al. defined the notion of context-aware computing and described different categories of context-aware applications [172]. In the context of autonomic systems, Hinchey and Sterritt referred to self-awareness as the capability of a system to be aware of its states and behaviors [98]. M. Parashar and S. Hariri referred to self-awareness as the ability of a system to be aware of its operational environment [153]. P. Gandodhar et al. reported the results of a survey on context-awarenss [79], and C. Perera et al. surveyed context-aware computing in the area of the Internet-of-Things [154]. S. Kounev et al. defined self-aware computing systems and outlined a taxonomy for these types of systems [119].

Several authors have provided arguments for why engineering self-adaptation at different levels of the technology stack poses different challenges. Among these are the growing complexity of the adaptation space from lower-level resources up to higher-level software [5, 36], and the increasingly complex interplay between system qualities on the one hand and adaptation options at higher levels of the software stack on the other hand [72].

M. Jackson contrasted the notion of environment, which is not under the control of a designer, and the system, which is controllable [106]. J. Kramer and J. Magee introduced the notion of quiescence [120]. A quiescent state of a software element is a state where no activity is going on in the element so that it can be safely updated. Such a state may be reached spontaneously or it may need to be enforced. J. Zhang and B. Cheng created the A-LTL specification language to specify the semantics of adaptive programs [218], underpinning safe adaptations. The OSGi framework [2] offers a modular service platform for Java that implements a dynamic component model that allows components (so called bundles) to be installed, started, stopped, updated, and uninstalled without requiring a reboot.

J. Kephart and D. Chess identified the primary types of higher-level adaptation goals [112]: self-configuration, self-optimization, self-healing, and self-protection.

M. Salehie and L. Tahvildari referred to self-adaptive software as software that embodies a closed-loop mechanism in the form of an adaptation loop [170]. Similarly, Dobson et al. referred to an autonomic control loop, which includes processes to collect and analyze data, and decide and act upon the system [65]. Y. Brun et al. argued for making feedback loops first-class entities in the design and operation of self-adaptive systems [35].

J. Camara et al. elaborated on involving humans in the feedback loop to support different self-adaptation functions, including the decision-making process [44]. Weyns et al. presented a set of architectural patterns for decentralizing control in self-adaptive systems [209].

The service-based health assistance system used in this book is based on the Tele-Assistance System (TAS) exemplar [200]. TAS offers a prototypical application that can be used to evaluate and compare new methods, techniques, and tools for research on self-adaptation in the domain of service-based systems. The service-based health assistance system was originally introduced in [15].

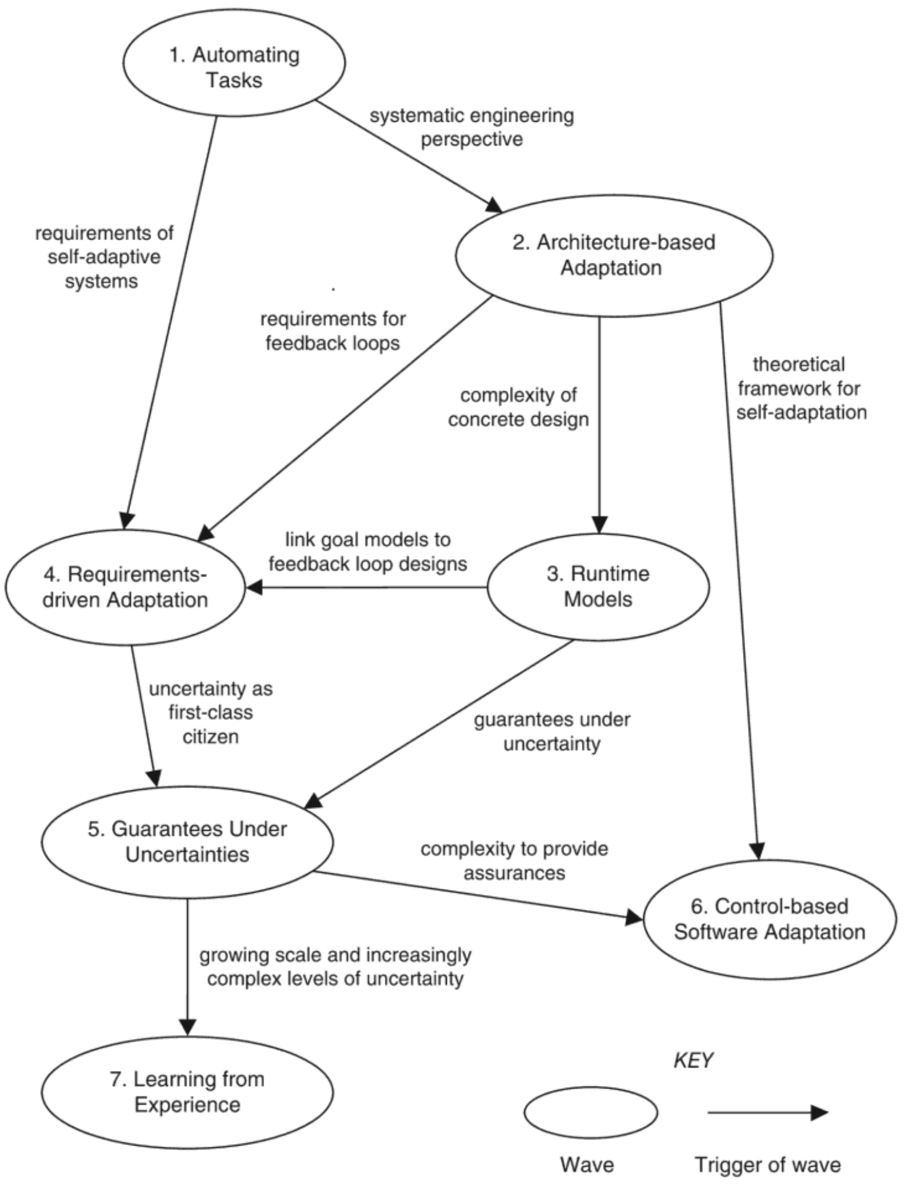
1. Automating Tasks
systematic engineering perspective
2. Architecture-based Adaptation
requirements of self-adaptive systems
requirements for feedback loops
complexity of concrete design
theoretical framework for self-adaptation
link goal models to feedback loop designs
4. Requirements-driven Adaptation
3. Runtime Models
uncertainty as first-class citizen
guarantees under uncertainty
5. Guarantees Under Uncertainties
complexity to provide assurances
6. Control-based Software Adaptation
growing scale and increasingly complex levels of uncertainty
7. Learning from Experience
KEY
Wave
Trigger of wave

Figure 2.1 Seven waves of research on engineering self-adaptive systems

The fourth wave, *Requirements-driven Adaptation*, is triggered by the need to consider requirements of self-adaptive systems as first-class citizens (from waves one and two) and link the requirements to feedback loop designs (from wave three). The fourth wave puts the emphasis on the requirements that need to be solved by the managing system and how these requirements drive the design of the managing system. A distinction can be made between requirements that accommodate the adaptation concerns (and need to be translated to operational adaptation goals) and requirements about the functionality of the managing system, in particular its correct realization.

The fifth wave, *Guarantees Under Uncertainty*, is triggered by the need to deal with uncertainty as a first-class citizen when engineering self-adaptive systems (from wave four) and how to mitigate this uncertainty and guarantee the adaptation goals (from wave three). The fifth wave is concerned with providing trustworthiness for self-adaptive systems that need to operate under uncertainty. An important aspect of this wave is the collection of evidence that a self-adaptive system has realized its adaptation goals. This evidence can be provided offline (i.e. not directly controlled by the running system) and complemented online (i.e. under control of the running system).

The sixth wave, *Control-based Software Adaptation*, is triggered by the complexity of providing assurances (from wave five) and the need for a theoretical framework for self-adaptation (from wave two). The sixth wave is concerned with exploiting the mathematical basis of control theory for designing self-adaptive systems and analyzing and guaranteeing key properties. Central aspects of this wave are the definition of adaptation goals, the selection of a controller, and the identification of a model of the managed system that is used by the controller. A particular challenge is understanding the relationship between quality requirements, adaptation goals, and traditional controller properties, which is important for providing guarantees for self-adaptive systems.

Wave seven, *Learning from Experience*, is triggered by the growing scale of systems and increasingly complex levels of uncertainty (from wave five). The seventh wave is concerned with exploiting machine learning techniques to support different functions that need to be realized by a managing system. Examples are the use of learning techniques to keep a runtime model up-to-date, to reduce very large search spaces of adaptation options, learning to determine the impact of adaptation decisions on the goals of systems, and sorting adaptation options by predicting the values of their quality properties in order to support efficient decision-making for adaptation.

2.2 Contributions Enabled by the Waves

Table 2.1 summarizes the relevant aspects of the state-of-the-art before each wave with a motivation for the wave, the topic that is studied in each wave, and the contributions that are enabled by each of the waves.

The table summarizes how the subsequent waves have triggered each other, contributing complementary knowledge on the engineering of self-adaptive systems. At the time of writing, waves W1 to W4 are relatively stable and have contributed a substantial body of knowledge to the field. Wave W5 is in an active stage, but this wave has already produced substantial consolidated knowledge. Waves W6 and W7 are relatively new, and the knowledge consolidated in these waves is still limited.

2.3 Waves Over Time with Selected Work

Figure 2.2 gives a schematic overview of when each of the seven waves of research in the field of self-adaptive systems emerged over time. The time window per wave represents

Table 2.1 Summary of the state-of-the-art before each wave with motivation, topic of each wave, and the contributions enabled by each wave (or *expected to be enabled* for active waves).

Wave	State of the art before wave and motivation for the wave	Topic of the wave	Contributions (to be) enabled by the wave
W1	System management done by human operators is a complex and error prone process	Automation of management tasks	Systems manage themselves based on high-level objectives; understanding of the basic functions of self-adaptation
W2	Motivation for self-adaptation acknowledged; need for a principled engineering perspective to define self-adaptive systems and reason about adaptation at runtime	Architecture perspective on self-adaptation	Architecture as driver for the engineering of self-adaptive systems; central role of architectural models to reason about adaptation at runtime
W3	Architecture principles of self-adaptive systems understood; concrete realization is complex	Model-driven approach extended to runtime to realize self-adaptation	Different types of runtime models as key elements to define feedback loops and support runtime decision-making
W4	Design of feedback loops well understood; requirements problem they intend to solve is implicit	Requirements for self-adaptive systems	Languages and formalisms to specify requirements for self-adaptive systems and their operationalization
W5	Mature solutions for engineering self-adaptive systems, but uncertainty handled in ad-hoc manner	The role of uncertainty in self-adaptive systems and how to mitigate it	Use of formal techniques (at runtime) to guarantee adaptation goals under uncertainty
W6	Engineering of MAPE-based self-adaption well understood, but solutions are often complex	Applying principles from control theory to realize self-adaptation	Control theory as a basis for the design and formal analysis of self-adaptive systems
W7	Growing scale of systems and increasingly complex levels of uncertainty they face	The use of learning techniques to support self-adaptation	Learning techniques used to support the effectiveness of different adaptation functions

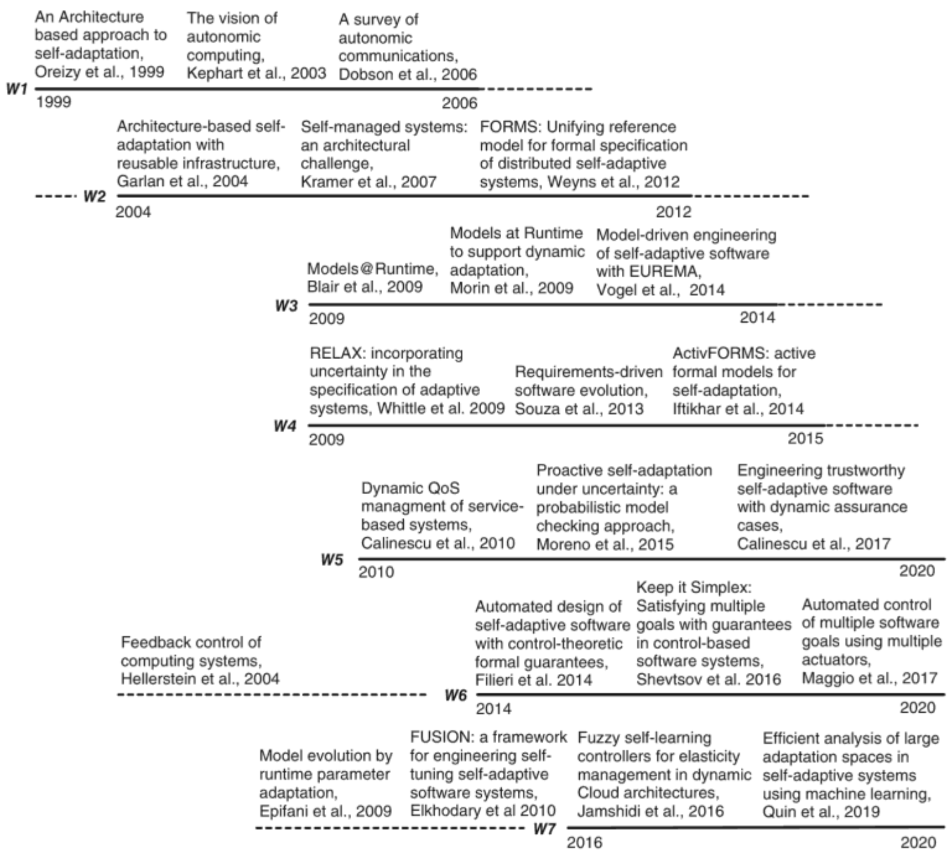
W1
1999
An Architecture based approach to self-adaptation, Oreizy et al., 1999
The vision of autonomic computing, Kephart et al., 2003
A survey of autonomic communications, Dobson et al., 2006
2006
W2
2004
Architecture-based self-adaptation with reusable infrastructure, Garlan et al., 2004
Self-managed systems: an architectural challenge, Kramer et al., 2007
FORMS: Unifying reference model for formal specification of distributed self-adaptive systems, Weyns et al., 2012
2012
W3
2009
Models@Runtime, Blair et al., 2009
Models at Runtime to support dynamic adaptation, Morin et al., 2009
Model-driven engineering of self-adaptive software with EUREMA, Vogel et al., 2014
2014
W4
2009
RELAX: incorporating uncertainty in the specification of adaptive systems, Whittle et al. 2009
Requirements-driven software evolution, Souza et al., 2013
ActivFORMS: active formal models for self-adaptation, Iftikhar et al., 2014
2015
W5
2010
Dynamic QoS managment of service-based systems, Calinescu et al., 2010
Proactive self-adaptation under uncertainty: a probabilistic model checking approach, Moreno et al., 2015
Engineering trustworthy self-adaptive software with dynamic assurance cases, Calinescu et al., 2017
2020
W6
Feedback control of computing systems, Hellerstein et al., 2004
2014
Automated design of self-adaptive software with control-theoretic formal guarantees, Filieri et al. 2014
Keep it Simplex: Satisfying multiple goals with guarantees in control-based software systems, Shevtsov et al. 2016
Automated control of multiple software goals using multiple actuators, Maggio et al., 2017
2020
W7
Model evolution by runtime parameter adaptation, Epifani et al., 2009
FUSION: a framework for engineering self-tuning self-adaptive software systems, Elkhodary et al 2010
2016
Fuzzy self-learning controllers for elasticity management in dynamic Cloud architectures, Jamshidi et al., 2016
Efficient analysis of large adaptation spaces in self-adaptive systems using machine learning, Quin et al., 2019
2020

Figure 2.2 Main periods of activity of each wave over time with representative research papers.

the main period of research activity during a wave. The papers associated with each wave are either a key paper that triggered the research of the wave or papers that characterize the research for that wave. For Waves 6 and 7, which are still in an early stage, there were important precursor papers a few years before the research activities of these waves effectively took off.

2.4 Summary

Over the past two decades, the research in the field of self-adaptation went through seven waves. These waves put complementary aspects of engineering self-adaptive systems into focus. The waves highlight trends of interest in the community that together have produced the current body of knowledge in the field.

The root wave, *Automating Tasks*, deals with delegating complex management tasks of software-intensive systems from human operators to machines. *Architecture-based Adaptation* laid the basis for a systematic engineering approach for self-adaptive systems and the

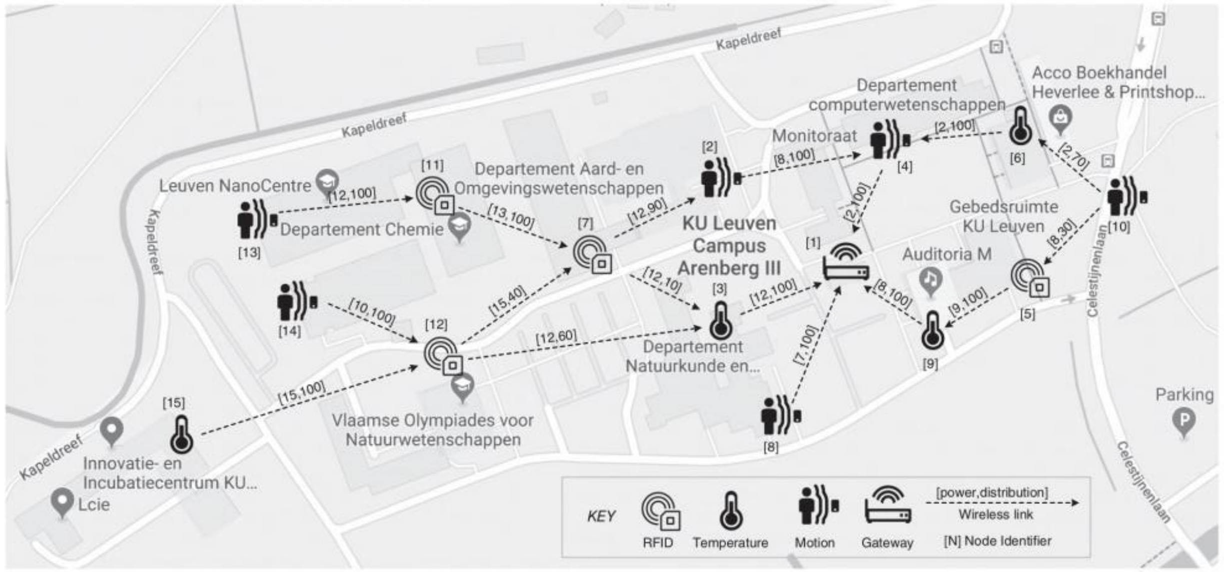

Figure 3.1 Geographical deployment of the DeltaIoT network