An Introduction to Self-Adaptive Systems

A Contemporary Software Engineering Perspective

Danny Weyns
Katholieke Universiteit Leuven, Belgium
Linnaeus University Växjö, Sweden

WILEY

IEEE PRESS

Contents

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.

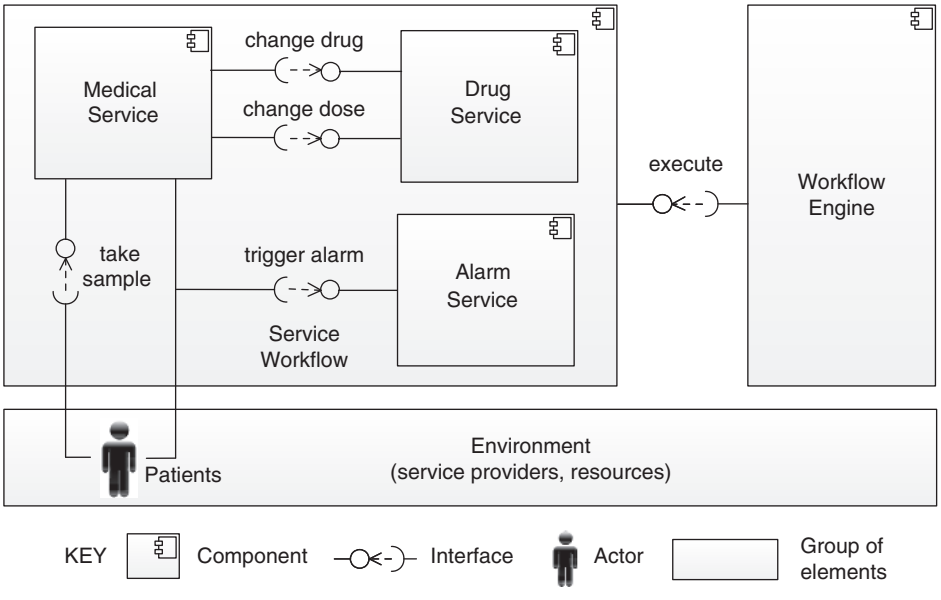

Figure 1.1 Architecture of a simple service-based health assistance system

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.

The first interpretation considers a self-adaptive system as a system that is able to adjust its behavior in response to the perception of changes in the environment and the system itself. The *self* prefix indicates that the system decides autonomously (i.e. without or with minimal human intervention) how to adapt to accommodate changes in its context and environment. Furthermore, a prevalent aspect of this first interpretation is the presence of uncertainty in the environment or the domain in which the software is deployed. To deal with these uncertainties, the self-adaptive system performs tasks that are traditionally done by operators. Hence, the first interpretation takes the stance of the external observer and looks at a self-adaptive system as a black box. Self-adaptation is considered as an observable property of a system that enables it to handle changes in external conditions, availability of resources, workloads, demands, and failures and threats.

The second interpretation contrasts traditional "internal" mechanisms that enable a system to deal with unexpected or unwanted events, such as exceptions in programming languages and fault-tolerant protocols, with "external" mechanisms that are realized by means of a closed feedback loop that monitors and adapts the system behavior at runtime. This interpretation emphasizes a "disciplined split" between two distinct parts of a self-adaptive system: one part that deals with the domain concerns and another part that deals with the adaptation concerns. Domain concerns relate to the goals of the users for which the system is built; adaptation concerns relate to the system itself, i.e. the way the system realizes the user goals under changing conditions. The second interpretation takes the stance of the engineer of the system and looks at self-adaptation from the point of view how the system is conceived.

Hence, we introduce *two complementary basic principles* that determine what a self-adaptive system is:

1. ***External principle:*** A self-adaptive system is a system that can handle changes and uncertainties in its environment, the system itself, and its goals autonomously (i.e. without or with minimal required human intervention).
2. ***Internal principle:*** A self-adaptive system comprises two distinct parts: the first part interacts with the environment and is responsible for the domain concerns – i.e. the concerns of users for which the system is built; the second part consists of a feedback loop that interacts with the first part (and monitors its environment) and is responsible for the adaptation concerns – i.e. concerns about the domain concerns.

Let us illustrate how the two principles of self-adaptation apply to the service-based health assistance system. Self-adaptation would enable the system to deal with dynamics in the types of services that are invoked by the system as well as variations in the failure rates and costs of particular service instances. Such uncertainties may be hard to anticipate before the system is deployed (external principle). To that end, the service-based system could be enhanced with a feedback loop. This feedback loop tracks the paths of services that are invoked in the workflow, as well as the failure rates of service instances and the costs of invoking service instances that are provided by the service providers. Taking this data into account, the feedback loop adapts the selection of service instances by the workflow engine such that a set of adaptation concerns is achieved. For instance, services

are selected that keep the average failure rate below a required threshold, while the cost of using the health assistance system is minimized (internal principle).

1.2 Other Adaptation Approaches

The ability of a software-intensive system to adapt at runtime in order to achieve its goals under changing conditions is not the exclusivity of self-adaptation, but can be realized in other ways.

The field of autonomous systems has a long tradition of studying systems that can change their behavior during operation in response to events that may not have been anticipated fully. A central idea of autonomous systems is to mimic human (or animal) behavior, which has been a source of inspiration for a very long time. The area of cybernetics founded by Norbert Wiener at MIT in the mid twentieth century led to the development of various types of machines that exposed seemingly "intelligent" behavior similar to biological systems. Wiener's work contributed to the foundations of various fields, including feedback control, automation, and robotics. The interest in autonomous systems has expanded significantly in recent years, with high-profile application domains such as autonomous vehicles. While these applications have extreme potential, their successes so far have also been accompanied by some dramatic failures, such as the accidents caused by first generation autonomous cars. The consequences of such failures demonstrate the real technical difficulties associated with realizing truly autonomous systems.

An important sub-field of autonomous systems is multi-agent systems, which studies the coordination of autonomous behavior of agents to solve problems that go beyond the capabilities of single agents. This study involves architectures of autonomous agents, communication and coordination mechanisms, and supporting infrastructure. An important aspect is the representation of knowledge and its use to coordinate autonomous behavior of agents. Self-organizing systems emphasize decentralized control. In a self-organizing system, simple reactive agents apply local rules to adapt their interactions with other agents in response to changing conditions in order to cooperatively realize the system goals. In such systems, the global macroscopic behavior emerges from the local interactions of the agents. However, emergent behavior can also appear as an unwanted side effect, for example in the form of oscillations. Designing decentralized systems that expose the required global behavior while avoiding unwanted emergent phenomena remains a major challenge.

Context-awareness is another traditional field that is related to self-adaptation. Context-awareness puts the emphasis on handling relevant elements in the physical environment as first-class citizens in system design and operation. Context-aware computing systems are concerned with the acquisition of context (e.g. through sensors to perceive a situation), the representation and understanding of context, and the steering of behavior based on the recognized context (e.g. triggering actions based on the actual context). Context-aware systems typically have a layered architecture, where a context manager or dedicated middleware is responsible for sensing and dealing with context changes. Self-aware computing systems contrast with context-aware computing systems in the sense that these systems capture and learn knowledge not only about the environment but also about themselves. This knowledge is encoded in the form of runtime models,

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

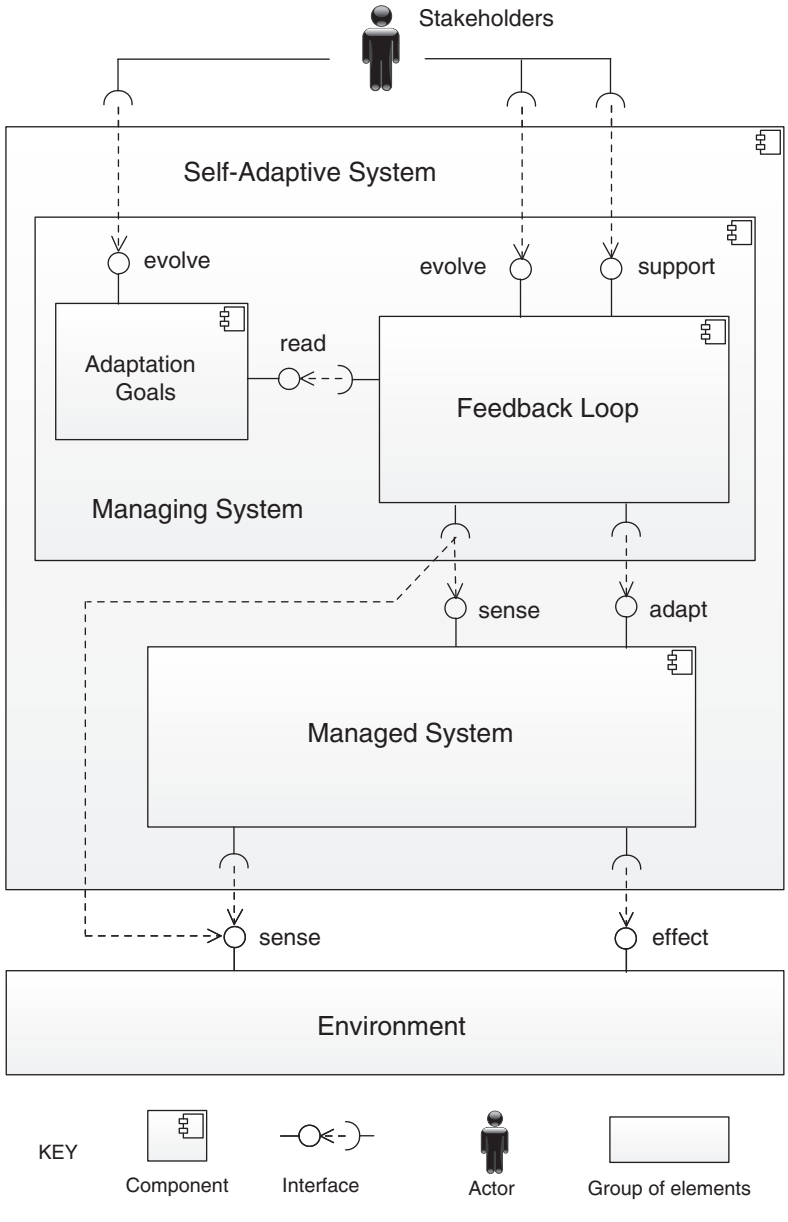

Figure 1.2 Conceptual model of a self-adaptive system

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

of the day may temporarily be excluded from selection to avoid harming the trustworthiness of the system. As we explained above, when a new adaptation goal is added to the system, in order to keep the average latency of invocations of the assistance service below a required threshold, the managing system will need to be updated. For instance, the managing system will need to be updated such that it can make adaptation decisions based on three adaptation goals instead of two.

1.4.5 Conceptual Model Applied

Figure 1.3 summarizes how the the conceptual model maps to the self-adaptive service-based health assistance system. The operator in this particular instance is responsible for supporting the self-adaptive system with handling fail-safe conditions (through the support interface). In this example, we do not consider the evolution of adaptation goals and the managing system.

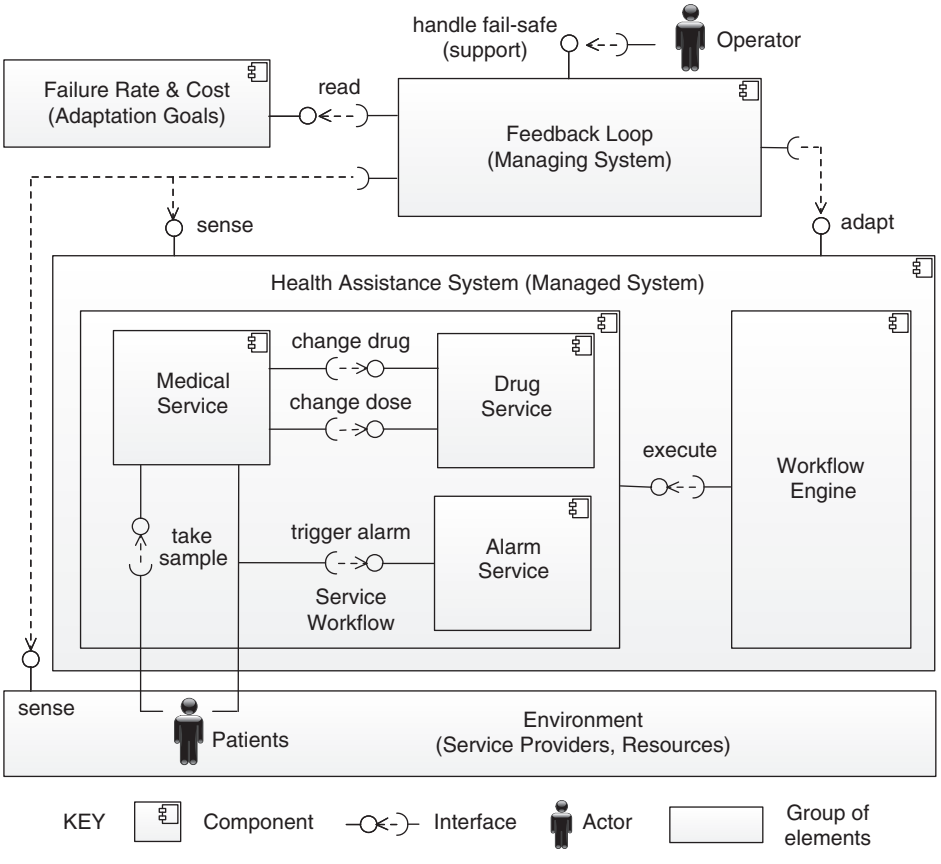

Figure 1.3 Conceptual model applied to a self-adaptive service-based health assistance system

1.5 A Note on Model Abstractions

It is important to note that the conceptual model for self-adaptive systems abstracts away from distribution – i.e. the deployment of the software to hardware that is connected via a network. Whereas a distributed self-adaptive system consists of multiple software components that are deployed on multiple nodes connected via some network, from a conceptual point of view such system can be represented as one managed system (that deals with the domain concerns) and one managing system (that deals with adaptation concerns of the managed system). The conceptual model also abstracts away from how adaptation decisions in a self-adaptive system are made and potentially coordinated among different components. In particular, the conceptual model is invariant to self-adaptive systems where the adaptation functions are made by a single centralized entity or by multiple coordinating entities in a decentralized way. In a concrete setting, the composition of the components of a self-adaptive system, the concrete deployment of these components to hardware elements, and the degree of decentralization of the decision making of adaptation will have a deep impact on how such self-adaptive systems are engineered.

1.6 Summary

Dealing with uncertainties in the operating conditions of a software-intensive system that are difficult to predict is an important challenge for software engineers. Self-adaptation is about how a system can mitigate such uncertainties.

There are two common interpretations of what constitutes a self-adaptive system. The first interpretation considers a self-adaptive system as a system that is able to adjust its behavior in response to changes in the environment or the system itself. The second interpretation contrasts traditional internal mechanisms that enable a system to deal with unexpected or unwanted events with external mechanisms that are realized by means of feedback loops.

These interpretations lead to two complementary basic principles that determine what is a self-adaptive system. The external principle states that a self-adaptive system can handle change and uncertainties autonomously (or with minimal human intervention). The internal principle states that a self-adaptive system consists of two distinct parts: one part that interacts with the environment and deals with the domain concerns and a second part that interacts with the first part and deals with the adaptation concerns.

Other traditional approaches to deal with change at runtime include autonomous systems, multi-agent systems, self-organizing systems, and context-aware systems. These approaches differ from self-adaptation, in particular with respect to the second basic principle. However, the second principle can be applied to these approaches through adding a managing system realizing self-adaptation.

Conceptually, a self-adaptive system consists of four basic elements: environment, managed system, adaptation goals, and feedback loop. The environment is external to the system; it defines the domain concerns and is not under control of the software engineer. The

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].

Index

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

u

Bibliography

1 T. Abdelzaher, Y. Diao, J. Hellerstein, C. Lu, and X. Zhu. Introduction to Control Theory And Its Application to Computing Systems. In Zn Liu and C. Xia, editors, *Performance Modeling and Engineering*, pages 185–215. Springer, 2008. ISBN 978-0-387-79361-0. doi: 10.1007/978-0-387-79361-0_7. URL https://doi.org/10.1007/978-0-387-79361-0_7.

2 OSGI Alliance. *OSGI Service Platform, Release 3*. IOS Press, Inc., 2003. ISBN 978-1-58603-311-8. URL https://www.iospress.nl/book/osgi-service-platform-release-3/.

3 R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994. ISSN 0304-3975. doi: https://doi.org/10.1016/0304-3975(94)90010-8. URL http://www.sciencedirect.com/science/article/pii/0304397594900108.

4 S. Aminikhanghahi and D. Cook. A Survey of Methods for Time Series Change Point Detection. *Knowledge Information Systems*, 51(2):339–367, May 2017. ISSN 0219-1377. doi: 10.1007/s10115-016-0987-z. URL https://doi.org/10.1007/s10115-016-0987-z.

5 J. Andersson, R. de Lemos, S. Malek, and D. Weyns. Modeling Dimensions of Self-Adaptive Software Systems. In B. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, pages 27–47. Springer, 2009. ISBN 978-3-642-02161-9. doi: 10.1007/978-3-642-02161-9_2. URL http://dx.doi.org/10.1007/978-3-642-02161-9_2.

6 J. Andersson, L. Baresi, N. Bencomo, R. de Lemos, A. Gorla, P. Inverardi, and T. Vogel. Software Engineering Processes for Self-Adaptive Systems. In R. de Lemos, H. Giese, H. Müller, and M. Shaw, editors, *Software Engineering for Self-Adaptive Systems II*, pages 51–75. Springer, 2013. ISBN 978-3-642-35813-5. doi: 10.1007/978-3-642-35813-5_3. URL https://doi.org/10.1007/978-3-642-35813-5_3.

7 K. Angelopoulos, A. Papadopoulos, V. Silva Souza, and J. Mylopoulos. Engineering Self-Adaptive Software Systems: From Requirements to Model Predictive Control. *Transactions on Autonomous and Adaptive Systems*, 13(1):1:1–1:27, April 2018. ISSN 1556-4665. doi: 10.1145/3105748. URL http://doi.acm.org/10.1145/3105748.

8 K. Astrom and R. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 11/2019. ISBN 978-0-691-13576-2.

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

URL http://www.cds.caltech.edu/~murray/amwiki/index.php/Second_Edition.

9 M. Autili, P. Inverardi, R. Spalazzese, M. Tivoli, and F. Mignosi. Automated Synthesis of Application-layer Connectors from Automata-based Specifications. *Journal of Computer and System Sciences*, 104:17–40, September 2019. doi: 10.1016/j.jcss.2019.03.001. URL https://www.sciencedirect.com/science/article/pii/S0022000019300248.

10 C. Baier and J.P. Katoen. *Principles of Model Checking*. The MIT Press. Cambridge, Massachusetts, 2008. ISBN 9780262026499. URL https://mitpress.mit.edu/books/principles-model-checking.

11 D. Barbosa, R. de Moura Lima, P. Maia, and E. Junior. Lotus@Runtime: A Tool for Runtime Monitoring and Verification of Self-adaptive Systems. In *12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 24–30, Buenos Aires, Argentina, 2017. IEEE. ISBN 978-1-5386-1550-8. doi: 10.1109/SEAMS.2017.18. URL https://doi.org/10.1109/SEAMS.2017.18.

12 D. Barbosa, R. de Moura Lima, P. Maia, and E. Junior. Lotus@Runtime Artifact, November 2019. URL https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/lotusruntime/.

13 D. Barbosa, R. de Moura Lima, P. Maia, and E. Junior. *Lotus@Runtime Github*, November 2019. URL https://github.com/davimonteiro.

14 L. Baresi and C. Ghezzi. The Disappearing Boundary between Development-time and Run-time. In *Future of Software Engineering Research*, pages 17–22, Santa Fe, New Mexico, USA, 2010. doi: 10.1145/1882362.1882367. URL https://doi.org/10.1145/1882362.1882367.

15 L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini. Validation of Web Service Compositions. *IET Software*, 1(6):219–232, December 2007. ISSN 1751-8814. doi: 10.1049/iet-sen:20070027. URL https://ieeexplore.ieee.org/document/4435102.

16 L. Baresi, L. Pasquale, and P. Spoletini. Fuzzy Goals for Requirements-Driven Adaptation. In *18th IEEE International Requirements Engineering Conference*, pages 125–134, Banff, Alberta, Canada, 2010. IEEE. ISBN 978-0-7695-4162-4. doi: 10.1109/RE.2010.25. URL http://dx.doi.org/10.1109/RE.2010.25.

17 C. Barna, H. Ghanbari, M. Litoiu, and M. Shtern. Hogna: A Platform for Self-adaptive Applications in Cloud Environments. In *10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 83–87, Florence, Italy, 2015. IEEE. doi: 10.1109/SEAMS.2015.26. URL http://dl.acm.org/citation.cfm?id=2821357.2821372.

18 C. Barna, H. Ghanbari, M. Litoiu, and M. Shtern. Hogna Artifact, November 2019. URL https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/hogna/.

19 J. Beal, M. Viroli, D. Pianini, and F. Damiani. Self-Adaptation to Device Distribution in the Internet of Things. *ACM Transactions on Autonomous and Adaptive Systems*, 12(3):12:1–12:29, September 2017. ISSN 1556-4665. doi: 10.1145/3105758. URL http://doi.acm.org/10.1145/3105758.

20 N. Bencomo and A. Belaggoun. Supporting Decision-Making for Self-Adaptive Systems: From Goal Models to Dynamic Decision Networks. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, Essen, Germany, 2013. Springer. ISBN 978-3-642-37422-7. doi: 10.1007/978-3-642-37422-7_16. URL https://link.springer.com/chapter/10.1007/978-3-642-37422-7_16.

21 N. Bencomo and L. Hernán García Paucar. RaM: Causally-Connected and Requirements-Aware Runtime Models using Bayesian Learning. In *22nd International Conference on Model Driven Engineering Languages and Systems*, pages 216–226, Munich, Germany, 2019. IEEE. doi: 10.1109/MODELS.2019.00005. URL https://doi.org/10.1109/MODELS.2019.00005.

22 N. Bencomo, J. Whittle, P. Sawyer, A. Finkelstein, and E. Letier. Requirements Reflection: Requirements as Runtime Entities. In *32nd International Conference on Software Engineering*, Cape Town, South Africa, 2010. IEEE. doi: 10.1145/1810295.1810329. URL https://ieeexplore.ieee.org/document/6062159.

23 N. Bencomo, R. France, B. Cheng, and U. Assmann. *Models@Run.Time: Foundations, Applications, and Roadmaps*, volume 8378 of *Lecture Notes in Computer Science*. Springer, 2014. ISBN 978-3-319-08915-7. doi: 10.1007/978-3-319-08915-7. URL https://www.springer.com/gp/book/9783319089140.

24 N. Bencomo, S. Götz, and H. Song. Models@Run.Time: A Guided Tour of the State of the Art and Research Challenges. *Software & Systems Modeling*, 18(5):3049–3082, October 2019. doi: 10.1007/s10270-018-00712-x. URL https://doi.org/10.1007/s10270-018-00712-x.

25 A. Bennaceur, C. McCormick, J. Galán, C. Perera, A. Smith, A. Zisman, and B. Nuseibeh. Feed Me, Feed Me: An Exemplar for Engineering Adaptive Software. In *11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 89–95, Austin, Texas, 2016. ACM. ISBN 978-1-4503-4187-5. doi: 10.1145/2897053.2897071. URL http://doi.acm.org/10.1145/2897053.2897071.

26 A. Bennaceur, C. McCormick, J. Galán, C. Perera, A. Smith, Andrea Z., and B. Nuseibeh. Feed Me, Feed Me Artifact, November 2019. URL https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/feed-me-feed-me/.

27 A. Bennaceur, C. McCormick, J. Galán, C. Perera, A. Smith, Andrea Z., and B. Nuseibeh. Feed Me, Feed Me Artifact Website, November 2019. URL http://sead1.open.ac.uk/fmfm/.

28 K. Bennett and V. Rajlich. Software Maintenance and Evolution: A Roadmap. In *The Future of Software Engineering*, pages 73–87, Limerick, Ireland, 2000. ACM. ISBN 1-58113-253-0. doi: 10.1145/336512.336534. URL http://doi.acm.org/10.1145/336512.336534.

29 J. Bernardo and A. Smith. *Bayesian Theory, 2nd Edition*. Wiley, 2007. ISBN 978-0470028735. URL https://www.wiley.com/en-us/Bayesian+Theory-p-9780471924166.

30 D. Berry, B. Cheng, and J. Zhang. The Four Levels of Requirements Engineering for and in Dynamic Adaptive Systems. In *International Workshop on the Design and Evolution of Autonomic Application Aoftware*, Saint Louis, MO, USA, 2005. doi:

10.1109/ICSE.2005.1553669. URL https://ieeexplore.ieee.org/document/1553669.

31 G. Blair, N. Bencomo, and R. France. Models@Run.Time. *IEEE Computer*, 42(10):22–27, October 2009. ISSN 0018-9162. doi: 10.1109/MC.2009.326. URL https://dl.acm.org/doi/10.1109/MC.2009.326.

32 L. Bojke, K. Claxton, M. Sculpher, and Palmer S. Characterizing Structural Uncertainty in Decision Analytic Models: A Review and Application of Methods. *Value Health*, 12(5):739–749, July-August 2009. doi: 10.1111/j.1524-4733.2008.00502.x. URL https://onlinelibrary.wiley.com/doi/full/10.1111/j.1524-4733.2008.00502.x.

33 V. Braberman, N. D'Ippolito, N. Piterman, D. Sykes, and S. Uchitel. Controller Synthesis: From Modelling to Enactment. In *35th International Conference on Software Engineering*, pages 1347–1350, San Francisco, CA, USA, 2013. IEEE. ISBN 978-1-4673-3076-3. doi: 10.1109/ICSE.2013.6606714. URL http://dl.acm.org/citation.cfm?id=2486788.2487002.

34 Y. Brun. Improving Impact of Self-adaptation and Self-management Research Through Evaluation Methodology. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, Cape Town, South Africa, 2010. ACM. doi: 10.1145/1808984.1808985. URL http://doi.acm.org/10.1145/1808984.1808985.

35 Y. Brun, G. Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw. Engineering Self-Adaptive Systems Through Feedback Loops. In B. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, pages 48–70. Springer, 2009. ISBN 978-3-642-02160-2. doi: 10.1007/978-3-642-02161-9_3. URL http://dx.doi.org/10.1007/978-3-642-02161-9_3.

36 Y. Brun, R. Desmarais, K. Geihs, M. Litoiu, A. Lopes, M. Shaw, and M. Smit. A Design Space for Self-Adaptive Systems. In R. de Lemos, H. Giese, H. Muller, and M. Shaw, editors, *Software Engineering for Self-Adaptive Systems II*, pages 33–50. Springer, 2013. ISBN 978-3-642-35813-5. doi: 10.1007/978-3-642-35813-5_2. URL http://dx.doi.org/10.1007/978-3-642-35813-5_2.

37 T. Bures, F. Plasil, M. Kit, P. Tuma, and N. Hoch. Software Abstractions for Component Interaction in the Internet of Things. *Computer*, 49(12): 50–59, December 2016. ISSN 1558-0814. doi: 10.1109/MC.2016.377. URL https://ieeexplore.ieee.org/document/7756271.

38 C. Cachin. Architecture of the Hyperledger Blockchain Fabric. In *Distributed Cryptocurrencies and Consensus Ledgers*, Chicago, Illinois, USA, 2016. IBM Research. URL https://www.zurich.ibm.com/dccl/papers/cachin_dccl.pdf.

39 R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic QoS Management and Optimization in Service-Based Systems. *IEEE Transactions on Software Engineering*, 37 (3):387–409, May 2011. ISSN 0098-5589. doi: 10.1109/TSE.2010.92. URL http://dx.doi.org/10.1109/TSE.2010.92.

40 R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola. Self-adaptive Software Needs Quantitative Verification at Runtime. *Communications of the ACM*, 55(9):69–77,

2012. doi: 10.1145/2330667.2330686. URL https://doi.org/10.1145/2330667.2330686.

41 R. Calinescu, S. Gerasimou, and A. Banks. Self-adaptive Software with Decentralised Control Loops. In *International Conference on Fundamental Approaches to Software Engineering*, London, UK, 2015. Springer. ISBN 978-3-662-46675-9. doi: 10.1007/978-3-662-46675-9_16. URL https://doi.org/10.1007/978-3-662-46675-9_16.

42 R. Calinescu, D. Weyns, S. Gerasimou, M. U. Iftikhar, I. Habli, and T. Kelly. Engineering Trustworthy Self-Adaptive Software with Dynamic Assurance Cases. *IEEE Transactions on Software Engineering*, 44(11):1039–1069, November 2018. ISSN 2326-3881. doi: 10.1109/TSE.2017.2738640. URL https://ieeexplore.ieee.org/document/8008800.

43 J. Cámara, P. Correia, R. de Lemos, D. Garlan, P. Gomes, B. Schmerl, and R. Ventura. Evolving an Adaptive Industrial Software System to Use Architecture-based Self-adaptation. In *8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 13–22, San Francisco, CA, USA, 2013. doi: 10.1109/SEAMS.2013.6595488. URL https://doi.org/10.1109/SEAMS.2013.6595488.

44 J. Camara, G. Moreno, and D. Garlan. Reasoning about Human Participation in Self-Adaptive Systems. In *10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 146–156, Florence, Italy, 2015. ACM. doi: 10.1109/SEAMS.2015.14. URL https://ieeexplore.ieee.org/abstract/document/7194669.

45 M. Caporuscio, V. Grassi, M. Marzolla, and R. Mirandola. GoPrime: A Fully Decentralized Middleware for Utility-Aware Service Assembly. *IEEE Transactions on Software Engineering*, 42(2):136–152, February 2016. doi: 10.1109/TSE.2015.2476797. URL https://ieeexplore.ieee.org/document/7243346.

46 V. Cardellini, E. Casalicchio, V. Grassi, and F. Lo Presti. Adaptive Management of Composite Services under Percentile-Based Service Level Agreements. In P. Maglio, M. Weske, J. Yang, and M. Fantinato, editors, *International Conference on Service-Oriented Computing*, pages 381–395, San Francisco, CA, USA, 2010. Springer. ISBN 978-3-642-17358-5. doi: 10.1007/978-3-642-17358-5_26. URL https://link.springer.com/chapter/10.1007/978-3-642-17358-5_26.

47 C. Cetina, P. Giner, J. Fons, and V. Pelechano. Autonomic Computing through Reuse of Variability Models at Runtime: The Case of Smart Homes. *Computer*, 42(10):37–43, October 2009. ISSN 1558-0814. doi: 10.1109/MC.2009.309. URL https://dl.acm.org/doi/10.1109/MC.2009.309.

48 V. Chandola, A. Banerjee, and V. Kumar. Anomaly Detection: A Survey. *ACM Computing Surveys*, 41(3):15:1–15:58, July 2009. ISSN 0360-0300. doi: 10.1145/1541880.1541882. URL http://doi.acm.org/10.1145/1541880.1541882.

49 B. Cheng and J. Atlee. Research Directions in Requirements Engineering. In *Future of Software Engineering*, pages 285–303, Minneapolis, MN, USA, 2007. IEEE. ISBN 0-7695-2829-5. doi: 10.1109/FOSE.2007.17. URL https://doi.org/10.1109/FOSE.2007.17.

50 B. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Di Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In B. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, pages 1–26. Springer, 2009. ISBN 978-3-642-02161-9. doi: 10.1007/978-3-642-02161-9_1. URL https://doi.org/10.1007/978-3-642-02161-9_1.

51 B. Cheng, P. Sawyer, N. Bencomo, and J. Whittle. A Goal-Based Modeling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty. In *12th International Conference on Model Driven Engineering Languages and Systems*, pages 468–483, Denver, CO, 2009. Springer. ISBN 978-3-642-04424-3. doi: 10.1007/978-3-642-04425-0_36. URL http://dx.doi.org/10.1007/978-3-642-04425-0_36.

52 S. Cheng and D. Garlan. Stitch: A Language for Architecture-based Self-adaptation. *Journal of Systems and Software*, 85(12):2860–2875, December 2012. ISSN 0164-1212. doi: 10.1016/j.jss.2012.02.060. URL http://dx.doi.org/10.1016/j.jss.2012.02.060.

53 S. Cheng and B. Schmerl. ZNN Artifact, November 2019. URL https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/model-problem-znn-com/.

54 Z. Coker, D. Garlan, and C. Le Goues. SASS: Self-adaptation Using Stochastic Search. In *10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 168–174, Florence, Italy, 2015. IEEE. doi: 10.5555/2821357.2821386. URL http://dl.acm.org/citation.cfm?id=2821357.2821386.

55 D. Cooray, S. Malek, R. Roshandel, and D. Kilgore. RESISTing Reliability Degradation Through Proactive Reconfiguration. In *International Conference on Automated Software Engineering*, pages 83–92, Antwerp, Belgium, 2010. ACM. ISBN 978-1-4503-0116-9. doi: 10.1145/1858996.1859011. URL http://doi.acm.org/10.1145/1858996.1859011.

56 C. E. da Silva, J. D. S. da Silva, C. Paterson, and R. Calinescu. Self-Adaptive Role-Based Access Control for Business Processes. In *12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 193–203, Buenos Aires, Argentina, 2017. doi: 10.1109/SEAMS.2017.13. URL https://doi.org/10.1109/SEAMS.2017.13.

57 A. David, K. Larsen, and A. Legay et al. Uppaal SMC Tutorial. *International Journal on Software Tools for Technology Transfer*, 17(4): 397–415, 2015. ISSN 1433-2787. doi: 10.1007/s10009-014-0361-y. URL https://doi.org/10.1007/s10009-014-0361-y.

58 R. de Lemos, H. Giese, H. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. Villegas, T. Vogel, D. Weyns, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cukic, R. Desmarais, S. Dustdar, G. Engels, K. Geihs, K. Göschka, A. Gorla, V. Grassi, P. Inverardi, G. Karsai, J. Kramer, A. Lopes, J. Magee, S. Malek, S. Mankovski,

R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. Pezzè, C. Prehofer, W. Schäfer, R. Schlichting, D. Smith, J. Pedro Sousa, L. Tahvildari, K. Wong, and J. Wuttke. Software Engineering for Self-Adaptive Systems: A Second Research Roadmap. In R. de Lemos, H. Giese, H. Muller, and M. Shaw, editors, *Software Engineering for Self-Adaptive Systems II*, pages 1–32. Springer, 2010. doi: 10.1007/978-3-642-35813-5_1. URL https://doi.org/10.1007/978-3-642-35813-5_1.

59 R. de Lemos, D. Garlan, C. Ghezzi, H. Giese, J. Andersson, M. Litoiu, B. Schmerl, D. Weyns, L. Baresi, N. Bencomo, Y. Brun, J. Camara, R. Calinescu, M. Cohen, A. Gorla, V. Grassi, L. Grunske, P. Inverardi, J. Jezequel, S. Malek, R. Mirandola, M. Mori, H. Müller, R. Rouvoy, C. Rubira, E. Rutten, M. Shaw, G. Tamburrelli, G. Tamura, N. Villegas, T. Vogel, and F. Zambonelli. Software Engineering for Self-Adaptive Systems: Research Challenges in the Provision of Assurances. In R. de Lemos, D. Garlan, C. Ghezzi, and H. Giese, editors, *Software Engineering for Self-Adaptive Systems III. Assurances*, pages 3–30. Springer, 2017. ISBN 978-3-319-74183-3. doi: 10.1007/978-3-319-74183-3_1. URL https://link.springer.com/chapter/10.1007/978-3-319-74183-3_1.

60 T. De Wolf and T. Holvoet. Emergence Versus Self-Organisation: Different Concepts but Promising When Combined. In *Engineering Self-Organising Systems: Methodologies and Applications*, pages 1–15, Utrecht, The Netherlands, 2005. Springer. ISBN 978-3-540-31901-6. doi: 10.1007/11494676_1. URL http://dx.doi.org/10.1007/11494676_1.

61 A. Dey. Context-Aware Computing. In J. Krumm, editor, *Ubiquitous Computing Fundamentals*, pages 321–352. Chapman & Hall/CRC, 2009. ISBN 1420093606. URL https://dl.acm.org/doi/book/10.5555/1803789.

62 A. Diaconescu, J. McCann, and P. Lalanda. *Autonomic Computing: Principles, Design and Implementation*. Springer, 2013. ISBN 978-1-4471-5007-7. doi: 10.1007/978-1-4471-5007-7. URL https://www.springer.com/gp/book/9781447150060.

63 N. D'Ippolito, V. Braberman, J. Kramer, J. Magee, D. Sykes, and S. Uchitel. Hope for the Best, Prepare for the Worst: Multi-tier Control for Adaptive Systems. In *36th International Conference on Software Engineering*, pages 688–699. ACM, 2014. ISBN 978-1-4503-2756-5. doi: 10.1145/2568225.2568264. URL http://doi.acm.org/10.1145/2568225.2568264.

64 DistriNet. PacketWorld Test Bed, November 2019. URL https://sourceforge.net/projects/packet-world/.

65 S. Dobson, S. Denazis, D. Fernández, A. and Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, Ni. Schmidt, and F. Zambonelli. A Survey of Autonomic Communications. *ACM Transactions on Autonomous and Adaptive Systems*, 1(2):223–259, December 2006. ISSN 1556-4665. doi: 10.1145/1186778.1186782. URL http://doi.acm.org/10.1145/1186778.1186782.

66 R. Edwards and N. Bencomo. DeSiRE: Further Understanding Nuances of Degrees of Satisfaction of Non-functional Requirements Trade-off. In *13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 12–18, Gothenburg, Sweden, 2018. ACM. ISBN 978-1-4503-5715-9. doi:

10.1145/3194133.3194142. URL http://doi.acm.org/10.1145/3194133.3194142.

67 A. Elkhodary, N. Esfahani, and S. Malek. FUSION: A Framework for Engineering Self-tuning Self-adaptive Software Systems. In *18th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 7–16, Santa Fe, New Mexico, USA, 2010. ACM. ISBN 978-1-60558-791-2. doi: 10.1145/1882291.1882296. URL http://doi.acm.org/10.1145/1882291.1882296.

68 I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Model Evolution by Run-time Parameter Adaptation. In *31st International Conference on Software Engineering*, pages 111–121, Vancouver, British Columbia, Canada, 2009. IEEE. ISBN 978-1-4244-3453-4. doi: 10.1109/ICSE.2009.5070513. URL http://dx.doi.org/10.1109/ICSE.2009.5070513.

69 N. Esfahani and S. Malek. Uncertainty in Self-Adaptive Software Systems. In R. de Lemos, H. Giese, H. Müller, and M. Shaw, editors, *Software Engineering for Self-Adaptive Systems II*, pages 214–238. Springer, 2013. ISBN 978-3-642-35813-5. doi: 10.1007/978-3-642-35813-5_9. URL https://doi.org/10.1007/978-3-642-35813-5_9.

70 N. Esfahani, E. Kouroshfar, and S. Malek. Taming Uncertainty in Self-adaptive Software. In *19th SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, pages 234–244, Szeged, Hungary, 2011. ACM. ISBN 978-1-4503-0443-6. doi: 10.1145/2025113.2025147. URL http://doi.acm.org/10.1145/2025113.2025147.

71 A. Filieri, H. Hoffmann, and M. Maggio. Automated Design of Self-adaptive Software with Control-theoretical Formal Guarantees. In *36th International Conference on Software Engineering*, pages 299–310, Hyderabad, India, 2014. ACM. ISBN 978-1-4503-2756-5. doi: 10.1145/2568225.2568272. URL http://doi.acm.org/10.1145/2568225.2568272.

72 A. Filieri, H. Hoffmann, and M. Maggio. Automated Multi-objective Control for Self-adaptive Software Design. In *10th Joint Meeting on Foundations of Software Engineering*, pages 13–24, Bergamo, Italy, 2015. ACM. ISBN 978-1-4503-3675-8. doi: 10.1145/2786805.2786833. URL http://doi.acm.org/10.1145/2786805.2786833.

73 A. Filieri, M. Maggio, K. Angelopoulos, N. D'Ippolito, I. Gerostathopoulos, A. Hempel, H. Hoffmann, P. Jamshidi, E. Kalyvianaki, C. Klein, F. Krikava, S. Misailovic, A. Papadopoulos, S. Ray, A. Sharifloo, S. Shevtsov, M. Ujma, and T. Vogel. Software Engineering Meets Control Theory. In *10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 71–82, Florence, Italy, 2015. IEEE. doi: 10.1109/SEAMS.2015.12. URL http://dl.acm.org/citation.cfm?id=2821357.2821370.

74 A. Filieri, M. Maggio, K. Angelopoulos, N. D'ippolito, I. Gerostathopoulos, A. Hempel, H. Hoffmann, P. Jamshidi, E. Kalyvianaki, C. Klein, F. Krikava, S. Misailovic, A. Papadopoulos, S. Ray, A. Sharifloo, S. Shevtsov, M. Ujma, and T. Vogel. Control Strategies for Self-Adaptive Software Systems. *ACM Transactions on Autonomous and Adaptive Systems*, 11(4):24:1–24:31, 2017. ISSN 1556-4665. doi: 10.1145/3024188. URL http://doi.acm.org/10.1145/3024188.

75 P. Flach. *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*. Cambridge University Press, 2012. ISBN 9781107422223. URL https://www.cambridge.org/be/academic/subjects/computer-science/pattern-recognition-and-machine-learning/.

76 World Economic Forum. How to Prevent Discriminatory Outcomes in Machine Learning, November 2019. URL https://www.weforum.org/whitepapers/how-to-prevent-discriminatory-outcomes-in-machine-learning/.

77 E. Fosler-Lussier. Markov Models and Hidden Markov Models: A Brief Tutorial. Berkeley Technical Report TR-98-041, November 2019. URL http://www.icsi.berkeley.edu/ftp/global/pub/techreports/1998/tr-98-041.pdf.

78 E. Fredericks, B. DeVries, and B. Cheng. AutoRELAX: Automatically RELAXing a Goal Model to Address Uncertainty. *Empirical Software Engineering*, 19(5):1466–1501, 2014. doi: 10.1007/s10664-014-9305-0. URL https://doi.org/10.1007/s10664-014-9305-0.

79 P. Gandodhar and S. Chaware. Context Aware Computing Systems: A Survey. In *2nd International Conference on IoT in Social, Mobile, Analytics and Cloud*, pages 605–608, Tamil Nadu, India, 2018. doi: 10.1109/I-SMAC.2018.8653786. URL https://ieeexplore.ieee.org/document/8653786.

80 D. Garlan and B. Schmerl. Model-based Adaptation for Self-healing Systems. In *First Workshop on Self-healing Systems*, pages 27–32, Charleston, South Carolina, 2002. ACM. ISBN 1-58113-609-9. doi: 10.1145/582128.582134. URL http://doi.acm.org/10.1145/582128.582134.

81 D. Garlan, S. Cheng, A. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. *IEEE Computer*, 37(10):46–54, October 2004. ISSN 0018-9162. doi: 10.1109/MC.2004.175. URL https://doi.org/10.1109/MC.2004.175.

82 E. Gat. Three-layer Architectures. In D. Kortenkamp, P. Bonasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, pages 195–210. MIT Press, 1998. ISBN 0-262-61137-6. doi: 10.1.1.43.9376. URL http://dl.acm.org/citation.cfm?id=292092.292130.

83 D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Language Systems*, 7(1):80–112, January 1985. ISSN 0164-0925. doi: 10.1145/2363.2433. URL http://doi.acm.org/10.1145/2363.2433.

84 S. Gerasimou, R. Calinescu, S. Shevtsov, and D. Weyns. UNDERSEA: An Exemplar for Engineering Self-Adaptive Unmanned Underwater Vehicles. In *12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 83–89, Buenos Aires, Argentina, 2017. doi: 10.1109/SEAMS.2017.19. URL https://doi.org/10.1109/SEAMS.2017.19.

85 S. Gerasimou, R. Calinescu, S. Shevtsov, and D. Weyns. UNDERSEA Artifact, November 2019. URL https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/undersea/.

86 S. Gerasimou, R. Calinescu, S. Shevtsov, and D. Weyns. UNDERSEA Website, November 2019. URL https://www-users.cs.york.ac.uk/simos/UNDERSEA/.

87 D. Ghosh, R. Sharman, Raghav R., and S. Upadhyaya. Self-healing Systems – Survey and Synthesis. *Decision Support Systems*, 42(4):2164–2185, January 2007. ISSN

0167-9236. doi: 10.1016/j.dss.2006.06.011. URL http://dx.doi.org/10.1016/j.dss.2006.06.011.

88 T. Glazier, B. Schmerl, J. Camara, and D. Garlan. Utility Theory for Self-Adaptive Systems. Carnegie Mellon University Technical Report CMU-ISR-17-119, 2017. URL http://acme.able.cs.cmu.edu/pubs/uploads/pdf/CMU-ISR-17-119.pdf.

89 M. Gorlick and R. Razouk. Using Weaves for Software Construction and Analysis. In *13th International Conference on Software Engineering*, pages 23–34, Austin, Texas, USA, 1991. IEEE. ISBN 0-89791-391-4. doi: 10.1109/ICSE.1991.130620. URL http://dl.acm.org/citation.cfm?id=256664.256677.

90 E. Grua, I. Malavolta, and P. Lago. Self-adaptation in Mobile Apps: A Systematic Literature Study. In *14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 51–62, Montreal, Quebec, Canada, 2019. IEEE. doi: 10.1109/SEAMS.2019.00016. URL https://doi.org/10.1109/SEAMS.2019.00016.

91 M. Harman. The Current State and Future of Search Based Software Engineering. In *Future of Software Engineering*, pages 342–357, Minneapolis, MN, USA, 2007. IEEE. ISBN 0-7695-2829-5. doi: 10.1109/FOSE.2007.29. URL https://doi.org/10.1109/FOSE.2007.29.

92 M. Harman. The Role of Artificial Intelligence in Software Engineering. In *Realizing AI Synergies in Software Engineering*, Zurich, Switzerland, 2012. IEEE. doi: 10.1109/RAISE.2012.6227961. URL https://ieeexplore.ieee.org/document/6227961.

93 M. Harman, S. Mansouri, and Y. Zhang. Search-based Software Engineering: Trends, Techniques and Applications. *ACM Computing Surveys*, 45(1):11:1–11:61, December 2012. ISSN 0360-0300. doi: 10.1145/2379776.2379787. URL http://doi.acm.org/10.1145/2379776.2379787.

94 J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury. *Feedback Control of Computing Systems*. John Wiley Sons, Inc., USA, 2004. ISBN 9780471266372. doi: 10.1002/047166880X. URL https://onlinelibrary.wiley.com/doi/book/10.1002/047166880X.

95 M. Herlihy. Blockchains from a Distributed Computing Perspective. *Communications of the ACM*, 62(2):78–85, January 2019. ISSN 0001-0782. doi: 10.1145/3209623. URL https://doi.org/10.1145/3209623.

96 H. Hermes and J. Lasalle. *Functional Analysis and Time Optimal Control, Volume 56*. Elsevier, 1969. ISBN 9780080955650. URL https://www.elsevier.com/books/functional-analysis-and-time-optimal-control/hermes/978-0-12-342650-5.

97 F. Heylighen. The Science of Self-organization and Adaptivity. In L. D. Kiel, editor, *Knowledge Management, Organizational Intelligence and Learning, and Complexity: Volume III*. EOLSS Publishers Co Ltd, 2002. ISBN 978-1-84826-913-2. URL https://www.eolss.net/.

98 M. Hinchey and R. Sterritt. Self-Managing Software. *Computer*, 39(2): 107–109, February 2006. ISSN 0018-9162. doi: 10.1109/MC.2006.69. URL https://doi.org/10.1109/MC.2006.69.

99 P. Horn. Autonomic Computing: IBM's Perspective on the State of Information Technology. IBM Technical Report, November 2019. URL https://www.semanticscholar.org/paper/Autonomic-Computing~3A-IBM~27s-Perspective-on-the-State-Horn/1ad1c619a9b3ba5a3ac597f51c8d15011a83423b.

100 IBM. An Architectural Blueprint for Autonomic Computing, November 2019. URL https://www-03.ibm.com/autonomic/pdfs/ACBlueprintWhitePaperV7.pdf.

101 U. Iftikhar and D. Weyns. ActivFORMS: Active Formal Models for Self-adaptation. In *9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 125–134, Hyderabad, India, 2014. ACM. ISBN 978-1-4503-2864-7. doi: 10.1145/2593929.2593944. URL http://doi.acm.org/10.1145/2593929.2593944.

102 U. Iftikhar, G. Ramachandran, P. Bollansée, D. Weyns, and D. Hughes. DeltaIoT: A Self-adaptive Internet of Things Exemplar. In *12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 76–82, Buenos Aires, Argentina, 2017. IEEE. ISBN 978-1-5386-1550-8. doi: 10.1109/SEAMS.2017.21. URL https://doi.org/10.1109/SEAMS.2017.21.

103 U. Iftikhar, G. Sankar Ramachandran, P. Bollansee, D. Weyns, and D. Hughes. DeltaIoT Artifact, November 2019. URL https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/deltaiot/.

104 D. Iglesia and D. Weyns. MAPE-K Formal Templates to Rigorously Design Behaviors for Self-Adaptive Systems. *Transactions on Autonomous and Adaptive Systems*, 10(3):15:1–15:31, September 2015. ISSN 1556-4665. doi: 10.1145/2724719. URL http://doi.acm.org/10.1145/2724719.

105 ISO/IEC25010. 25010 Standard, November 2019. URL https://www.iso.org/standard/35733.html.

106 M. Jackson. The Meaning of Requirements. *Annals of Software Engineering*, 3(1): 5–21, 1997. ISSN 1573-7489. doi: 10.1023/A:1018990005598. URL https://doi.org/10.1023/A:1018990005598.

107 P. Jamshidi, A. Sharifloo, C. Pahl, H. Arabnejad, A. Metzger, and G. Estrada. Fuzzy self-learning controllers for elasticity management in dynamic cloud architectures. In *12th International ACM SIGSOFT Conference on Quality of Software Architectures*, pages 70–79, Venice, Italy, 2016. doi: 10.1109/QoSA.2016.13. URL https://ieeexplore.ieee.org/document/7515437.

108 P. Jamshidi, J. Camara, B. Schmerl, C. Käestner, and D. Garlan. Machine Learning Meets Quantitative Planning: Enabling Self-Adaptation in Autonomous Robots. In *14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 39–50, Montreal, Quebec Canada, 2019. doi: 10.1109/SEAMS.2019.00015. URL https://dl.acm.org/doi/10.1109/SEAMS.2019.00015.

109 P. Janert. *Feedback Control for Computer Systems: Introducing Control Theory to Enterprise Programmers*. O'Reilly, 2013. ISBN 9781449361693. URL http://shop.oreilly.com/product/0636920028970.do.

110 N. Jennings, P. Faratin, A. Lomuscio, S. Parsons, M. Wooldridge, and C. Sierra. Automated Negotiation: Prospects, Methods and Challenges. *Group Decision and Negotiation*, 10(2):199–215, March 2001. ISSN 1572-9907. doi: 10.1023/A:1008746126376. URL https://doi.org/10.1023/A:1008746126376.

111 I. Jureta, J. Mylopoulos, and S. Faulkner. Revisiting the Core Ontology and Problem in Requirements Engineering. In *2008 16th IEEE International Requirements Engineering Conference*, pages 71–80, Catalunya, Spain, 2008. ISBN 978-0-7695-3309-4. doi: 10.1109/RE.2008.13. URL https://ieeexplore.ieee.org/document/4685655.

112 J. Kephart and D. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36 (1):41–50, January 2003. ISSN 0018-9162. doi: 10.1109/MC.2003.1160055. URL http://dx.doi.org/10.1109/MC.2003.1160055.

113 J. Kephart and W. Walsh. An Artificial Intelligence Perspective on Autonomic Computing Policies. In *5th IEEE International Workshop on Policies for Distributed Systems and Networks*, Yorktown Heights, NY, USA, 2004. doi: 10.1109/POLICY.2004.1309145. URL https://ieeexplore.ieee.org/document/1309145.

114 N. Khakpour, C. Skandylas, G. S. Nariman, and D. Weyns. Towards Secure Architecture-Based Adaptations. In *14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 114–125, Montreal, Canada, 2019. doi: 10.1109/SEAMS.2019.00023. URL https://dl.acm.org/doi/10.1109/SEAMS.2019.00023.

115 G. Kiczales, J. des Rivieres, and Bobrow. D. *The Art of the Metaobject Protocol*. The MIT Press. Cambridge, Massachusetts, 1991. ISBN 9780262111584. URL https://mitpress.mit.edu/books/art-metaobject-protocol.

116 G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, Je. Loingtier, and J. Irwin. Aspect-oriented Programming. In *European Conference on Object-Oriented Programming*, pages 220–242, Jyvaskyla, Finland,, 1997. Springer. ISBN 978-3-540-69127-3. doi: 10.1007/BFb0053381. URL https://link.springer.com/chapter/10.1007/BFb0053381.

117 C. Kinneer, Z. Coker, J. Wang, D. Garlan, and C. Le Goues. Managing Uncertainty in Self-adaptive Systems with Plan Reuse and Stochastic Search. In *13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 40–50, Gothenburg, Sweden, 2018. ACM. ISBN 978-1-4503-5715-9. doi: 10.1145/3194133.3194145. URL http://doi.acm.org/10.1145/3194133.3194145.

118 B. Kitchenham. Procedures for Performing Systematic Reviews. Keele University Technical Report TR/SE-0401, November 2019. URL http://www.inf.ufsc.br/~aldo.vw/kitchenham.pdf.

119 S. Kounev, J.O. Kephart, A. Milenkoski, and X. Zhu, editors. *Self-Aware Computing Systems*. Springer, 2017. ISBN 978-3-319-47474-8. URL https://www.springer.com/gp/book/9783319474724.

120 J. Kramer and J. Magee. The Evolving Philosophers Problem: Dynamic Change Management. *IEEE Transactions on Software Engineering*, 16 (11):1293–1306, 1990. ISSN 0098-5589. doi: 10.1109/32.60317. URL https://ieeexplore.ieee.org/document/60317.

121 J. Kramer and J. Magee. Self-Managed Systems: An Architectural Challenge. In *Future of Software Engineering. FOSE '07*, pages 259–268, Minneapolis, MN, USA, 2007. doi: 10.1109/FOSE.2007.19. URL https://ieeexplore.ieee.org/document/4221625.

122 D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang. Power and Performance Management of Virtualized Computing Environments via Lookahead Control. *Cluster Computing*, 12(1):1–15, March 2009. ISSN 1386-7857. doi: 10.1007/s10586-008-0070-y. URL http://dx.doi.org/10.1007/s10586-008-0070-y.

123 M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic Symbolic Model Checker. In T. Field, P. Harrison, J. Bradley, and U. Harder, editors, *Computer Performance Evaluation: Modelling Techniques and Tools*, pages 200–204, London, UK, 2002. Springer. ISBN 978-3-540-46029-9. doi: 10.5555/2944225.2944369. URL https://link.springer.com/chapter/10.1007/3-540-46029-2_13.

124 A. Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software*. Wiley, 2009. ISBN 978-0-470-01270-3. URL https://www.wiley.com/en-us/Requirements+Engineering\~3A+From+System+Goals+to+UML+Models+to+Software+Specifications-p-9780470012703.

125 J. Leikas, R. Koivisto, and N. Gotcheva. Ethical Framework for Designing Autonomous Intelligent Systems. *Journal of Open Innovation: Technology, Market, and Complexity*, 5(1), March 2019. doi: 10.3390/joitmc5010018. URL https://doi.org/10.3390/joitmc5010018.

126 A. Leva. An Introduction to Systems and Control Theory for Computer Scientists and Engineers. In *8th ACM/SPEC International Conference on Performance Engineering*, pages 433–436, L'Aquila, Italy, 2017. ACM. ISBN 978-1-4503-4404-3. doi: 10.1145/3030207.3053677. URL http://doi.acm.org/10.1145/3030207.3053677.

127 W. Levine. *The Control Handbook: Control System Applications, Second Edition*. CRC Press, 2010. ISBN 9781420073607. doi: 10.1201/b10384. URL https://doi.org/10.1201/b10384.

128 H. Lim, S. Babu, J. Chase, and S Parekh. Automated Control in Cloud Computing: Challenges and Opportunities. In *Workshop on Automated Control for Datacenters and Clouds*, pages 13–18, Barcelona, Spain, 2009. ACM. ISBN 978-1-60558-585-7. doi: 10.1145/1555271.1555275. URL http://doi.acm.org/10.1145/1555271.1555275.

129 P. Maes. Computational Reflection. In K. Morik, editor, *11th German Workshop on Artificial Intelligence*, pages 251–265, Geseke, Germany, 1987. Springer. ISBN 978-3-642-73005-4. URL https://dl.acm.org/doi/proceedings/10.5555/647607.

130 P. Maes. *Computational Reflection*. Vrije Universiteit Brussel, Belgium, 1987. URL http://soft.vub.ac.be/Publications/1987/vub-arti-phd-87_2.pdf.

131 J. Magee and J. Kramer. Dynamic Structure in Software Architectures. In *4th ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 3–14, San Francisco, California, USA, 1996. ACM. ISBN 0-89791-797-9. doi: 10.1145/239098.239104. URL http://doi.acm.org/10.1145/239098.239104.

132 J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying Distributed Software Architectures. In *5th European Software Engineering Conference*, London, UK, 1995. Springer. ISBN 3-540-60406-5. doi: 10.1007/3-540-60406-5_12. URL http://dl.acm.org/citation.cfm?id=645385.651497.

133 M. Maggio and H. Hoffmann. ARPE: A Tool To Build Equation Models of Computing Systems. In *8th International Workshop on Feedback Computing*, San Jose, CA, USA, 2013. USENIX Association. URL https://www.usenix.org/node/174699.

134 M. Maggio, A. Papadopoulos, A. Filieri, and H. Hoffmann. Automated Control of Multiple Software Goals Using Multiple Actuators. In *11th Joint Meeting on Foundations of Software Engineering*, pages 373–384, Paderborn, Germany, 2017. ACM. ISBN 978-1-4503-5105-8. doi: 10.1145/3106237.3106247. URL http://doi.acm.org/10.1145/3106237.3106247.

135 M. Maggio, A. V. Papadopoulos, A. Filieri, and H. Hoffmann. Self-Adaptive Video Encoder: Comparison of Multiple Adaptation Strategies Made Simple. In *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 123–128, Buenos Aires, Argentina, 2017. doi: 10.1109/SEAMS.2017.16. URL https://ieeexplore.ieee.org/document/7968140.

136 M. Maggio, A. Vittorio Papadopoulos, A. Filieri, and H. Hoffmann. Self-Adaptive Video Encoder Artifact, November 2019. URL https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/self-adaptive-video-encoder/.

137 S. Mahdavi-Hezavehi, P. Avgeriou, and D. Weyns. A Classification Framework of Uncertainty in Architecture-Based Self-Adaptive Systems with Multiple Quality Requirements. In I. Mistrik, N. Ali, R. Kazman, J. Grundy, and B. Schmerl, editors, *Managing Trade-Offs in Adaptable Software Architectures*, pages 45–77. Morgan Kaufmann, 2017. ISBN 978-0-12-802855-1. doi: https://doi.org/10.1016/B978-0-12-802855-1.00003-4. URL http://www.sciencedirect.com/science/article/pii/B9780128028551000034.

138 T. Malone, T. Malone, and K. Crowston. The Interdisciplinary Study of Coordination. *ACM Computing Surveys*, 26(1):87–119, March 1994. ISSN 0360-0300. doi: 10.1145/174666.174668. URL http://doi.acm.org/10.1145/174666.174668.

139 D.H. Mellor. *The Facts of Causation*. Routledge. International Library of Philosophy, 1995. ISBN 0-415-09779-7.

140 G. Mohay, E. Ahmed, S. Bhatia, A. Nadarajan, B. Ravindran, A. Tickle, and R. Vijayasarathy. Detection and Mitigation of High-Rate Flooding Attacks. In S. Raghavan and E. Dawson, editors, *An Investigation into the Detection and Mitigation of Denial of Service (DoS) Attacks*. Springer, 2011. ISBN 978-81-322-0276-9. doi: 10.1007/978-81-322-0277-6_5. URL https://link.springer.com/chapter/10.1007/978-81-322-0277-6_5.

141 S. Moon, K. Lee, and D. Lee. Fuzzy Branching Temporal Logic. *Transactions on Systems, Man and Cybernetics, Part B*, 34(2):1045–1055, April 2004. ISSN 1083-4419. doi: 10.1109/TSMCB.2003.819485. URL http://dx.doi.org/10.1109/TSMCB.2003.819485.

142 G. Moreno, B. Schmerl, and D. Garlan. SWIM: An Exemplar for Evaluation and C@bomparison of Self-adaptation Approaches for Web Applications. In *13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 137–143, Gothenburg, Sweden, 2018. ACM. ISBN 978-1-4503-5715-9. doi: 10.1145/3194133.3194163. URL http://doi.acm.org/10.1145/3194133.3194163.

143 G. Moreno, B. Schmerl, and D. Garlan. SWIM Artifact, November 2019. URL https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/swim/.

144 G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl. Proactive Self-adaptation Under Uncertainty: A Probabilistic Model Checking Approach. In *Foundations of Software Engineering*, pages 1–12, Bergamo, Italy, 2015. ACM. ISBN 978-1-4503-3675-8. doi: 10.1145/2786805.2786853. URL http://doi.acm.org/10.1145/2786805.2786853.

145 B. Morin, O. Barais, J.M. Jezequel, F. Fleurey, and A. Solberg. Models@ Run.time to Support Dynamic Adaptation. *IEEE Computer*, 42 (10):44–51, October 2009. doi: 10.1109/MC.2009.327. URL https://ieeexplore.ieee.org/document/5280651.

146 L. Nahabedian, V. Braberman, N. D'ippolito, J. Kramer, and S. Uchitel. Dynamic Reconfiguration of Business Processes. In *International Conference on Business Process Management*, pages 35–51, Vienna, Austria, 2019. Springer. ISBN 978-3-030-26619-6. doi: 10.1007/978-3-030-26619-6_5. URL https://link.springer.com/chapter/10.1007/978-3-030-26619-6_5.

147 P. Norvig. Artificial Intelligence in the Software Engineering Workflow. In *O?Reilly Artificial Intelligence Conference*, New York, NY, USA, 2017. URL https://www.oreilly.com/radar/artificial-intelligence-in-the-software-engineering-workflow/.

148 B. Nuseibeh and S. Easterbrook. Requirements Engineering: A Roadmap. In *The Future of Software Engineering*, pages 35–46, Limerick, Ireland, 2000. ACM. ISBN 1-58113-253-0. doi: 10.1145/336512.336523. URL http://doi.acm.org/10.1145/336512.336523.

149 IEEE-CS/ACM Joint Task Force on Software Engineering Ethics and Professional Practices. Code of Ethics, November 2019. URL https://www.computer.org/education/code-of-ethics.

150 P. Oreizy, N. Medvidovic, and R. Taylor. Architecture-based Runtime Software Evolution. In *20th International Conference on Software Engineering*, pages 177–186, Kyoto, Japan, 1998. IEEE. ISBN 0-8186-8368-6. URL http://dl.acm.org/citation.cfm?id=302163.302181.

151 P. Oreizy, M. Gorlick, R. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, and A. Wolf. An Architecture-Based Approach to Self-Adaptive Software. *IEEE Intelligent Systems*, 14(3):54–62, May 1999. ISSN 1541-1672. doi: 10.1109/5254.769885. URL http://dx.doi.org/10.1109/5254.769885.

152 P. Padala, K. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and K. Merchant, A. and Salem. Adaptive Control of Virtualized Resources in Utility Computing Environments. In *2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, pages 289–302, Lisbon, Portugal, 2007. ACM. ISBN 978-1-59593-636-3. doi: 10.1145/1272996.1273026. URL http://doi.acm.org/10.1145/1272996.1273026.

153 M. Parashar and S. Hariri. Autonomic Computing: An Overview. In J. Banâtre, P. Fradet, J. Giavitto, and O. Michel, editors, *International Workshop on Unconventional Programming Paradigms*, pages 257–269, Le Mont Saint Michel, France, 2005. Springer. ISBN 978-3-540-31482-0. doi: 10.1007/11527800_20. URL https://link.springer.com/chapter/10.1007/11527800_20.

154 C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Context Aware Computing for The Internet of Things: A Survey. *IEEE Communications Surveys Tutorials*, 16(1):414–454, First Quarter 2014. ISSN 2373-745X. doi: 10.1109/SURV.2013.042313.00197. URL https://ieeexplore.ieee.org/document/6512846.

155 D. Perez-Palacin and R. Mirandola. Uncertainties in the Modeling of Self-adaptive Systems: A Taxonomy and an Example of Availability Evaluation. In *5th ACM/SPEC International Conference on Performance Engineering*, pages 3–14, Dublin, Ireland, 2014. ACM. ISBN 978-1-4503-2733-6. doi: 10.1145/2568088.2568095. URL http://doi.acm.org/10.1145/2568088.2568095.

156 PRISM. Probabilistic Symbolic Model Checker, November 2019. URL https://www.prismmodelchecker.org/.

157 N. Privault. *Understanding Markov Chains*. Springer, 2018. ISBN 978-981-13-0658-7. doi: https://doi.org/10.1007/978-981-13-0659-4. URL https://link.springer.com/book/10.1007/978-981-13-0659-4#about.

158 M. Provoost and D. Weyns. DingNet: A Self-adaptive Internet-of-Things Exemplar. In *14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, Montreal, QC, Canada, 2019. doi: 10.1109/SEAMS.2019.00033. URL https://doi.org/10.1109/SEAMS.2019.00033.

159 F. Quin, D. Weyns, T. Bamelis, S. Buttar, and S. Michiels. Efficient Analysis of Large Adaptation Spaces in Self-adaptive Systems Using Machine Learning. In *14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 1–12, Montreal, Quebec, Canada, 2019. IEEE. doi: 10.1109/SEAMS.2019.00011. URL https://doi.org/10.1109/SEAMS.2019.00011.

160 S. V. Raghavan and E. Dawson. *An Investigation into the Detection and Mitigation of Denial of Service (DoS) Attacks: Critical Information Infrastructure Protection*. Springer, 2011. ISBN 978-81-322-0277-6. URL https://www.springer.com/gp/book/9788132202769.

161 V. Rajlich. Software Evolution and Maintenance. In *Future of Software Engineering*, pages 133–144, Hyderabad, India, 2014. ACM. ISBN 978-1-4503-2865-4. doi: 10.1145/2593882.2593893. URL http://doi.acm.org/10.1145/2593882.2593893.

162 G. S. Ramachandran, N. Matthys, W. Daniels, W. Joosen, and D. Hughes. Building Dynamic and Dependable Component-Based Internet-of-Things Applications with

Dawn. In *19th International ACM SIGSOFT Symposium on Component-Based Software Engineering*, pages 97–106, Venice, Italy, 2016. doi: 10.1109/CBSE.2016.18. URL https://ieeexplore.ieee.org/document/7497436.

163 A. Ramirez and B. Cheng. Design Patterns for Developing Dynamically Adaptive Systems. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 49–58, Cape Town, South Africa, 2010. ACM. doi: 10.1145/1808984.1808990. URL https://doi.org/10.1145/1808984.1808990.

164 A. J. Ramirez, A. Jensen, B. Cheng, and D. Knoester. Automatically Exploring how Uncertainty Impacts Behavior of Dynamically Adaptive Systems. In *26th IEEE/ACM International Conference on Automated Software Engineering*, pages 568–571, Lawrence, KS, USA, 2011. doi: 10.1109/ASE.2011.6100127. URL https://ieeexplore.ieee.org/document/6100127.

165 S. Redwine and W. Riddle. Software Technology Maturation. In *8th International Conference on Software Engineering*, London, England, 1985. IEEE. ISBN 0-8186-0620-7. doi: 10.5555/319568.319624. URL http://dl.acm.org/citation.cfm?id=319568.319624.

166 N. Rocco De, L. Michele, P. Rosario, and T. Francesco. A Formal Approach to Autonomic Systems Programming: The SCEL Language. *ACM Transactions on Autonomous and Adaptive Systems*, 9(2):7:1–7:29, July 2014. ISSN 1556-4665. doi: 10.1145/2619998. URL http://doi.acm.org/10.1145/2619998.

167 R. Rouvoy, P. Barone, Y. Ding, F. Eliassen, S. Hallsteinsen, J. Lorenzo, A. Mamelli, and U. Scholz. MUSIC: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments. In B. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, pages 164–182. Springer, 2009. ISBN 978-3-642-02161-9. doi: 10.1007/978-3-642-02161-9_9. URL https://doi.org/10.1007/978-3-642-02161-9_9.

168 RUBiS. Rice University Bidding System, November 2019. URL https://www.rubis.ow2.org.

169 S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2009. ISBN 978-0-13-604259-4. doi: 10.1016/j.artint.2011.01.005. URL http://aima.cs.berkeley.edu/.

170 M. Salehie and L. Tahvildari. Self-adaptive Software: Landscape and Research Challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 4 (2):14:1–14:42, May 2009. ISSN 1556-4665. doi: 10.1145/1516533.1516538. URL http://doi.acm.org/10.1145/1516533.1516538.

171 P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein. Requirements-Aware Systems: A Research Agenda for RE for Self-adaptive Systems. In *18th IEEE International Requirements Engineering Conference*, pages 95–103, Banff, Alberta, Canada, 2010. doi: 10.1109/RE.2010.21. URL https://ieeexplore.ieee.org/document/5636882.

172 B. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. In *1st Workshop on Mobile Computing Systems and Applications*, pages 85–90, Santa Cruz, California, USA, 1994. IEEE. doi: 10.1109/WMCSA.1994.16. URL https://ieeexplore.ieee.org/document/4624429.

173 M. Seidl. *UML @ Classroom: An Introduction to Object-Oriented Modeling*. Springer, 2015. ISBN 978-3-319-12742-2. doi: 10.1007/978-3-319-12742-2. URL https://www.springer.com/gp/book/9783319127415.

174 M. L. Seto, L. Paull, and S. Saeedi. Introduction to Autonomy for Marine Robots. In M. Seto, editor, *Marine Robot Autonomy*, pages 1–46. Springer, 2013. ISBN 978-1-4614-5659-9. doi: 10.1007/978-1-4614-5659-9_1. URL https://doi.org/10.1007/978-1-4614-5659-9_1.

175 M. Shahin, M. Ali Babar, and L. Zhu. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access*, 5:3909–3943, March 2017. ISSN 2169-3536. doi: 10.1109/ACCESS.2017.2685629. URL https://ieeexplore.ieee.org/abstract/document/7884954.

176 S. Shevtsov and D. Weyns. Keep It SIMPLEX: Satisfying Multiple Goals with Guarantees in Control-based Self-adaptive Systems. In *24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 229–241, Seattle, WA, USA, 2016. ACM. ISBN 978-1-4503-4218-6. doi: 10.1145/2950290.2950301. URL http://doi.acm.org/10.1145/2950290.2950301.

177 S. Shevtsov, U. Iftikhar, and D. Weyns. SimCA vs ActivFORMS: Comparing Control- and Architecture-based Adaptation on the TAS Exemplar. In *International Workshop on Control Theory for Software Engineering*, pages 1–8, Bergamo, Italy, 2015. ACM. ISBN 978-1-4503-3814-1. doi: 10.1145/2804337.2804338. URL http://doi.acm.org/10.1145/2804337.2804338.

178 S. Shevtsov, M. Berekmeri, D. Weyns, and M. Maggio. Control-Theoretical Software Adaptation: A Systematic Literature Review. *IEEE Transactions on Software Engineering*, 44(8):784–810, Augustus 2018. ISSN 0098-5589. doi: 10.1109/TSE.2017.2704579. URL https://doi.org/10.1109/TSE.2017.2704579.

179 S. Shevtsov, D. Weyns, and M. Maggio. SimCA*: A Control-theoretic Approach to Handle Uncertainty in Self-adaptive Systems with Guarantees. *ACM Transactions on Autonomous and Adaptive Systems*, 13(4):17:1–17:34, July 2019. ISSN 1556-4665. doi: 10.1145/3328730. URL http://doi.acm.org/10.1145/3328730.

180 S. Shevtsov, D. Weyns., and M. Maggio. Self-Adaptation of Software Using Automatically Generated Control-Theoretical Solutions. In *Engineering Adaptive Software Systems - Communications of NII Shonan Meetings*, pages 35–55. Springer, Kamiyamaguchi, Hayama, Miura District, Kanagawa 240-0198, Japan, 2019. doi: 10.1007/978-981-13-2185-6_2. URL https://doi.org/10.1007/978-981-13-2185-6_2.

181 J. Shortle, J. Thompson, D. Gross, and C. Harris. *Fundamentals of Queueing Theory, Fifth Edition*. Wiley, 2018. ISBN 9781118943526. doi: 10.1002/9781119453765. URL https://onlinelibrary.wiley.com/doi/book/10.1002/9781119453765.

182 M. Sommer, S. Tomforde, J. Hahner, and D. Auer. Learning a Dynamic Re-combination Strategy of Forecast Techniques at Runtime. In *IEEE International Conference on Autonomic Computing*, pages 261–266, Grenoble, France, 2015. doi: 10.1109/ICAC.2015.70. URL https://ieeexplore.ieee.org/document/7266977.

183 V. Souza, A. Lapouchnian, W. Robinson, and J. Mylopoulos. Awareness Requirements for Adaptive Systems. In *6th International Symposium on Software Engineering for*

Adaptive and Self-Managing Systems, pages 60–69, Waikiki, Honolulu, HI, USA, 2011. ACM. ISBN 978-1-4503-0575-4. doi: 10.1145/1988008.1988018. URL http://doi.acm.org/10.1145/1988008.1988018.

184 V. Souza, A. Lapouchnian, K. Angelopoulos, and J. Mylopoulos. Requirements-driven Software Evolution. *Computer Science*, 28(4):311–329, November 2013. ISSN 1865-2034. doi: 10.1007/s00450-012-0232-2. URL http://dx.doi.org/10.1007/s00450-012-0232-2.

185 G. Tallabaci and V. E. Silva Souza. Engineering Adaptation with Zanshin: An Experience Report. In *8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 93–102, San Francisco, CA, USA, 2013. IEEE. doi: 10.1109/SEAMS.2013.6595496. URL https://ieeexplore.ieee.org/document/6595496.

186 G. Tesauro and J. Kephart. Utility Functions in Autonomic Systems. In *First International Conference on Autonomic Computing*, pages 70–77, New York, NY, USA, 2004. IEEE. ISBN 0-7695-2114-2. doi: 10.1109/ICAC.2004.68. URL http://dl.acm.org/citation.cfm?id=1078026.1078411.

187 R. Thayer and M. Dorfman. *Software Requirements Engineering, 2nd Edition*. Wiley-IEEE Computer Society, 1997. ISBN 978-0-818-67738-0. URL https://www.wiley.com/en-aw/Software+Requirements+Engineering,+2nd+Edition-p-9780818677380.

188 H. Tijms. *Stochastic Models, an Algorithmic Approach*. Wiley, 1994. ISBN 978-0471951230.

189 UPPAAL. UPPAAL Tool Suite, November 2019. URL http://www.uppaal.org/.

190 H. Van Dyke Parunak, Sven A. Brueckner, and John Sauter. Digital Pheromones for Coordination of Unmanned Vehicles. In *Environments for Multi-Agent Systems*, pages 246–263, New York, USA, 2005. Springer. ISBN 978-3-540-32259-7. URL https://link.springer.com/chapter/10.1007/978-3-540-32259-7_13.

191 N. Villegas, H. Müller, G. Tamura, L. Duchien, and R. Casallas. A Framework for Evaluating Quality-driven Self-adaptive Software Systems. In *6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 80–89, Waikiki, Honolulu, HI, USA, 2011. ACM. ISBN 978-1-4503-0575-4. doi: 10.1145/1988008.1988020. URL http://doi.acm.org/10.1145/1988008.1988020.

192 N. Villegas, G. Tamura, H. Müller, L. Duchien, and R. Casallas. DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems. In R. de Lemos, H. Giese, H. Müller, and M. Shaw, editors, *Software Engineering for Self-Adaptive Systems II*, pages 265–293. Springer, 2013. ISBN 978-3-642-35813-5. doi: 10.1007/978-3-642-35813-5_11. URL https://doi.org/10.1007/978-3-642-35813-5_11.

193 T. Vogel. mRUBiS: An Exemplar for Model-based Architectural Self-healing and Self-optimization. In *13th International Conference on Software Engineering for Adaptive and Self-Managing Systems*, pages 101–107, Gothenburg, Sweden, 2018. ACM. ISBN 978-1-4503-5715-9. doi: 10.1145/3194133.3194161. URL http://doi.acm.org/10.1145/3194133.3194161.

194 T. Vogel. mRUBIS Artifact, November 2019. URL https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/mrubis/.

195 T. Vogel and H. Giese. Model-Driven Engineering of Self-Adaptive Software with EUREMA. *ACM Transactions on Autonomous and Adaptive Systems*, 8(4):18:1–18:33, January 2014. ISSN 1556-4665. doi: 10.1145/2555612. URL http://doi.acm.org/10.1145/2555612.

196 T. Vogel and H. Giese. Self-Adaptive Systems Artifacts and Model Problems, November 2019. URL https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/.

197 G. Welch and G. Bishop. An Introduction to the Kalman Filter. University of North Carolina at Chapel Hill, NC, USA, 1995. URL https://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf.

198 D. Weyns. Software Engineering of Self-adaptive Systems. In S. Cha, R. Taylor, and K. Kang, editors, *Handbook of Software Engineering*, pages 399–443. Springer, 2019. doi: 10.1007/978-3-030-00262-6_11. URL https://doi.org/10.1007/978-3-030-00262-6_11.

199 D. Weyns and T. Ahmad. *Claims and Evidence for Architecture-Based Self-adaptation: A Systematic Literature Review*, pages 249–265. Springer, Montpellier, France, 2013. doi: 10.1007/978-3-642-39031-9_22. URL https://link.springer.com/chapter/10.1007/978-3-642-39031-9_22.

200 D. Weyns and R. Calinescu. Tele Assistance: A Self-adaptive Service-based System Examplar. In *10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 88–92, Florence, Italy, 2015. IEEE. URL http://dl.acm.org/citation.cfm?id=2821357.2821373.

201 D. Weyns and R. Calinescu. TAS Artifact, November 2019. URL https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/tas/.

202 D. Weyns and U. Iftikhar. ActivFORMS: A Model-Based Approach to Engineer Self-Adaptive Systems. *arXiv 1908.11179, cs.SE*, 2019. URL https://arxiv.org/abs/1908.11179.

203 D. Weyns and M. Provoost. DingNet Website, November 2019. URL https://people.cs.kuleuven.be/~danny.weyns/software/DingNet/index.htm.

204 D. Weyns, A. Helleboogh, and T. Holvoet. The Packet-World: A Test Bed for Investigating Situated Multi-Agent Systems. In R. Unland, M. Calisti, and M. Klusch, editors, *Software Agent-Based Applications, Platforms and Development Kits*, pages 383–408. Birkhäuser, 2005. ISBN 978-3-7643-7348-1. URL https://lirias.kuleuven.be/retrieve/5983.

205 D. Weyns, U. Iftikhar, D. de la Iglesia, and T. Ahmad. A Survey of Formal Methods in Self-adaptive Systems. In *Fifth International C* Conference on Computer Science and Software Engineering*, pages 67–79, Montreal, Quebec, Canada, 2012. ACM. ISBN 978-1-4503-1084-0. doi: 10.1145/2347583.2347592. URL http://doi.acm.org/10.1145/2347583.2347592.

206 D. Weyns, S. Malek, and J. Andersson. FORMS: Unifying Reference Model for Formal Specification of Distributed Self-adaptive Systems. *Transactions on Autonomous and*

Adaptive Systems, 7(1):8:1–8:61, 2012. ISSN 1556-4665. doi: 10.1145/2168260.2168268. URL http://doi.acm.org/10.1145/2168260.2168268.

207 D. Weyns, S. Malek, J. Andersson, and B. Schmerl. Introduction to the Special Issue on State of the Art in Engineering Self-adaptive Systems. *Journal of Systems and Software*, 85(12):2675–2677, 2012. ISSN 0164-1212. doi: 10.1016/j.jss.2012.07.045. URL https://doi.org/10.1016/j.jss.2012.07.045.

208 D. Weyns, U. Iftikhar, and J. Soderland. Do External Feedback Loops Improve the Design of Self-adaptive Systems? A Controlled Experiment. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, San Francisco, CA, USA, 2013. ISBN 978-1-4673-4401-2. doi: 0.1109/SEAMS.2013.6595487. URL http://dl.acm.org/citation.cfm?id=2487336.2487341.

209 D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, and K. Göschka. On Patterns for Decentralized Control in Self-Adaptive Systems. In R. de Lemos, H. Giese, H. Müller, and M. Shaw, editors, *Software Engineering for Self-Adaptive Systems II*, pages 76–107. Springer, 2013. ISBN 978-3-642-35813-5. doi: 10.1007/978-3-642-35813-5_4. URL https://doi.org/10.1007/978-3-642-35813-5_4.

210 D. Weyns, N. Bencomo, R. Calinescu, J. Camara, C. Ghezzi, V. Grassi, L. Grunske, P. Inverardi, J. Jezequel, S. Malek, R. Mirandola, M. Mori, and G. Tamburrelli. Perpetual Assurances for Self-Adaptive Systems. In R. de Lemos, D. Garlan, C. Ghezzi, and H. Giese, editors, *Software Engineering for Self-Adaptive Systems III. Assurances*, pages 31–63. Springer, 2017. ISBN 978-3-319-74183-3. doi: 10.1007/978-3-319-74183-3_2. URL https://link.springer.com/chapter/10.1007/978-3-319-74183-3_2.

211 D. Weyns, U. Iftikhar, D. Hughes, and N. Matthys. Applying Architecture-Based Adaptation to Automate the Management of Internet-of-Things. In *European Conference on Software Architecture*, pages 49–67, Madrid, Spain, 2018. Springer. ISBN 978-3-030-00761-4. doi: 10.1007/978-3-030-00761-4_4. URL https://link.springer.com/chapter/10.1007/978-3-030-00761-4_4.

212 D. Weyns, R. Calinescu, Iftikhar U., and Y. Ruan. TAS Website, November 2019. URL https://people.cs.kuleuven.be/~danny.weyns/software/TAS/.

213 D. Weyns, U. Iftikhar, and G. Sankar Ramachandran. DeltaIoT Website, November 2019. URL https://people.cs.kuleuven.be/~danny.weyns/software/DeltaIoT/.

214 J. Whittle, P. Sawyer, N. Bencomo, B. Cheng, and J. Bruel. RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems. In *17th IEEE International Requirements Engineering Conference*, pages 79–88, Atlanta, Georgia, USA, 2009. IEEE. ISBN 978-0-7695-3761-0. doi: 10.1109/RE.2009.36. URL http://dx.doi.org/10.1109/RE.2009.36.

215 M. Wooldrige. *An Introduction to MultiAgent Systems*. Wiley, USA, 2009. ISBN 978-0-470-51946-2. URL https://www.wiley.com/en-us/An+Introduction+to+MultiAgent+Systems\~2C+2nd+Edition-p-9780470519462.

216 S. Yang. Cybersecurity Threats – Can we Predict Them? In *Research Features Magazine: Engineering and Technology*. Research Publishing International Ltd, 2018. URL https://cdn2.researchfeatures.com/wp-content/uploads/2018/07/Shanchieh-Jay-Yang-1.pdf.

217 E. Yuan, N. Esfahani, and S. Malek. A Systematic Survey of Self-Protecting Software Systems. *ACM Transactions on Autonomous and Adaptive Systems*, 8(4):17:1–17:41, Januari 2014. ISSN 1556-4665. doi: 10.1145/2555611. URL http://doi.acm.org/10.1145/2555611.

218 J. Zhang and B. Cheng. Using Temporal Logic to Specify Adaptive Program Semantics. *Journal of Systems and Software*, 79(10):1361–1369, 2006. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2006.02.062. URL http://www.sciencedirect.com/science/article/pii/S0164121206001397.

219 J. Zhang and B. Cheng. Model-based Development of Dynamically Adaptive Software. In *28th International Conference on Software Engineering*, pages 371–380, Shanghai, China, 2006. ACM. ISBN 1-59593-375-1. doi: 10.1145/1134285.1134337. URL http://doi.acm.org/10.1145/1134285.1134337.

220 T. Zhao, W. Zhang, H. Zhao, and Z. Jin. A Reinforcement Learning-Based Framework for the Generation and Evolution of Adaptation Rules. In *IEEE International Conference on Autonomic Computing*, pages 103–112, Columbus, OH, USA, 2017. doi: 10.1109/ICAC.2017.47. URL https://ieeexplore.ieee.org/document/8005338.

221 F. Zhou, B. Wu, and Z. Li. Deep Meta-Learning: Learning to Learn in the Concept Space. *CoRR*, abs/1802.03596, 2018. URL http://arxiv.org/abs/1802.03596.