1

Basic Principles of Self-Adaptation and Conceptual Model

COPYRIGHTED MATERIAL

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.
© 2021 John Wiley & Sons Ltd. Published 2021 by John Wiley & Sons Ltd.

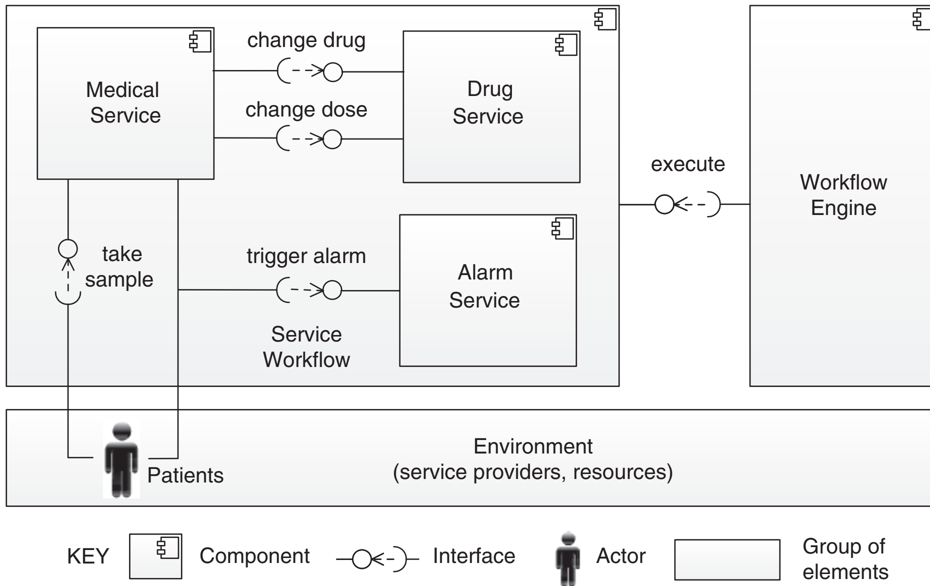

Figure 1.1 Architecture of a simple service-based health assistance system

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

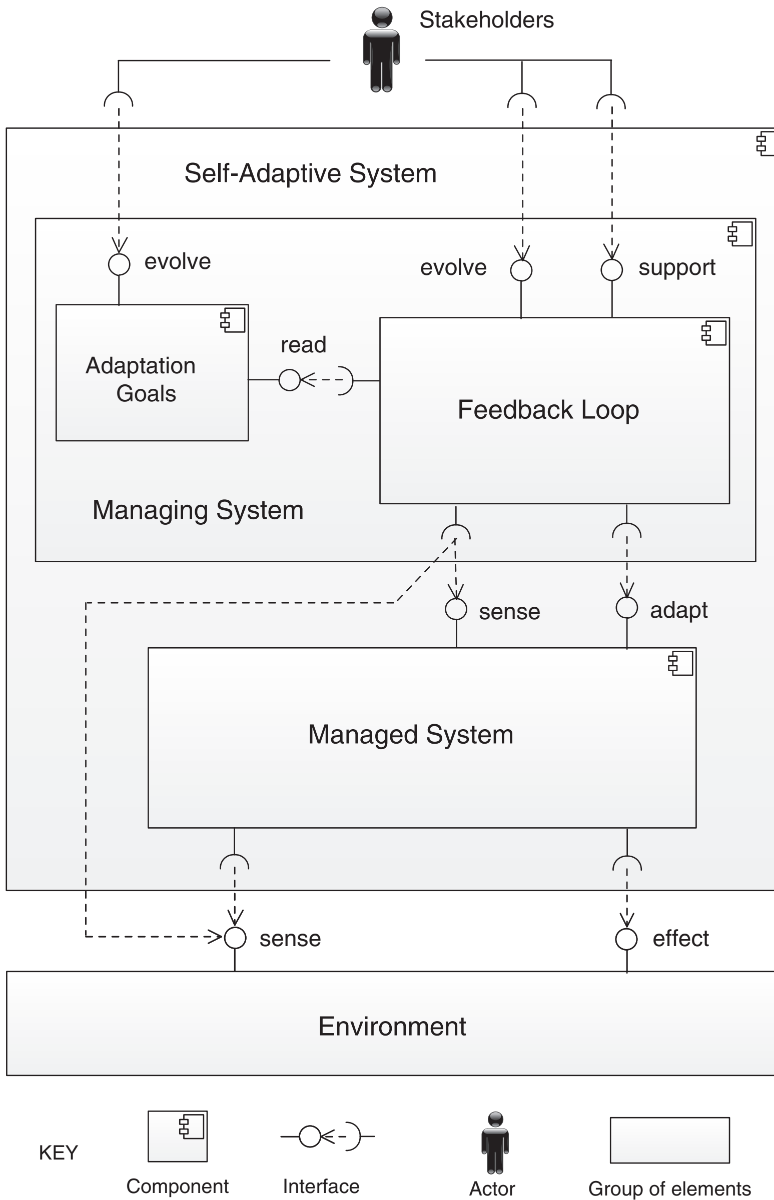

Figure 1.2 Conceptual model of a self-adaptive system

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.

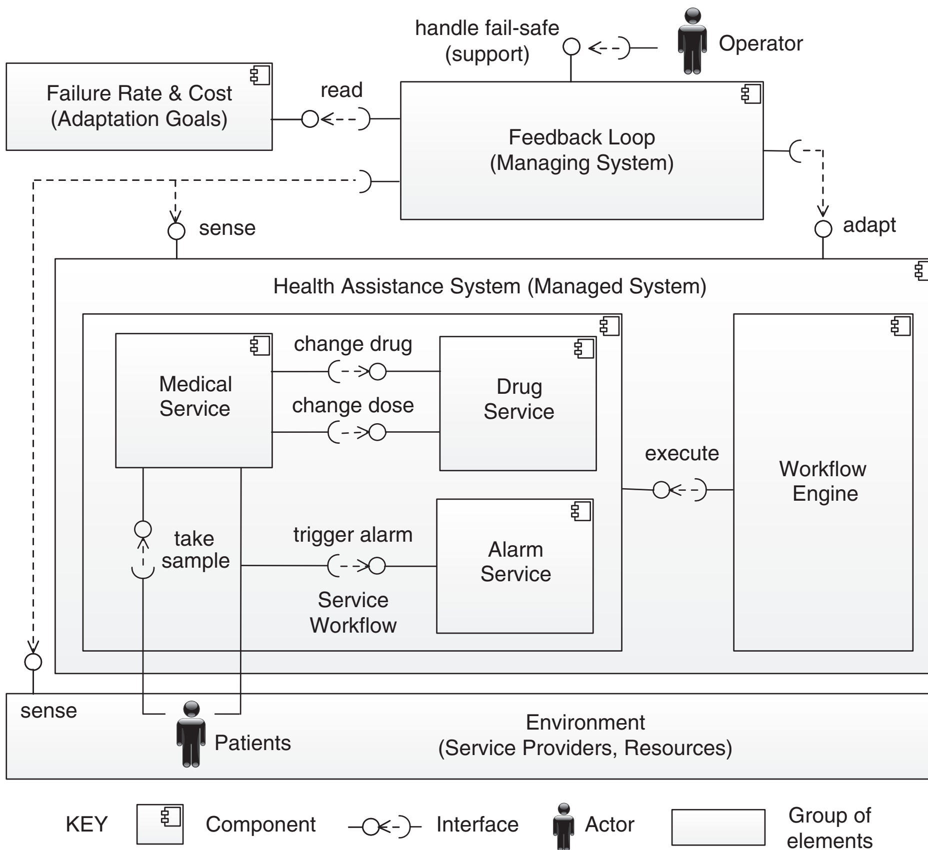

Figure 1.3 Conceptual model applied to a self-adaptive service-based health assistance system

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].