Chapter 2
Self-aware Computing Systems: Related Concepts and Research Areas

Javier Cámara, Kirstie L. Bellman, Jeffrey O. Kephart, Marco Autili, Nelly Bencomo, Ada Diaconescu, Holger Giese, Sebastian Götz, Paola Inverardi, Samuel Kounev and Massimo Tivoli

Abstract Self-aware computing systems exhibit a number of characteristics (e.g., autonomy, social ability, and proactivity) which have already been studied

J. Cámara (✉)
Carnegie Mellon University, Pittsburgh, PA 15213, USA
e-mail: jcmoreno@cs.cmu.edu

K.L. Bellman
The Aerospace Corporation, Los Angeles, CA, USA
e-mail: Kirstie.L.Bellman@aero.org

J.O. Kephart
Thomas J. Watson Research Center, Yorktown Heights, NY, USA
e-mail: kephart@us.ibm.com

M. Autili · P. Inverardi · M. Tivoli
University of L'Aquila, 67100 L'Aquila, Italy
e-mail: marco.autili@univaq.it

P. Inverardi
e-mail: paola.inverardi@univaq.it

M. Tivoli
e-mail: massimo.tivoli@univaq.it

N. Bencomo
Aston University, Birmingham B4 7ET, UK
e-mail: nelly@acm.org

A. Diaconescu
Telecom Paris Tech, 75013 Paris, France
e-mail: ada.diaconescu@telecom-paristech.fr

H. Giese
Hasso-Plattner-Institut, 14482 Potsdam, Germany
e-mail: Holger.Giese@hpi.de

S. Götz
University of Technology Dresden, Dresden, Germany
e-mail: sebastian.goetz@acm.org

S. Kounev
Universität Würzburg, Am Hubland, 97074 Würzburg, Germany
e-mail: skounev@acm.org

S. Kounev et al. (eds.), *Self-Aware Computing Systems*,
DOI 10.1007/978-3-319-47474-8_2

in different research areas, such as artificial intelligence, organic computing, or autonomic and self-adaptive systems. This chapter provides an overview of strongly related concepts and areas of study from the perspective of self-aware computing systems.

2.1 Introduction

The notion of self-aware computing encompasses different aspects which have already been the subject of study in different research areas of computer science. In fact, systems that feature one or several desirable characteristics in a self-aware computing system, such as being able to learn models about itself and its environment, reasoning, planning, or providing explanations, are already a reality. The construction of such systems has been made possible thanks to the research efforts carried out in areas such as artificial intelligence, autonomic computing, self-adaptive and self-organizing systems, or cognitive computing. As it happens, many of these disciplines will foreseeably be strongly intertwined with research in the area of self-aware computing, making it stand on the proverbial *shoulders of giants*.

This chapter presents an overview of concepts and research areas strongly related to self-aware computing. Every section presents a different area of research and explores its relation to self-aware computing systems. Note that there are disciplines that cannot be considered as fully within the scope of computer science (e.g., cybernetics) in which engineers employ ideas that are well aligned with the areas for which we provide an overview in this chapter. However, those areas are not discussed in this chapter due to space limitations.

This chapter starts with an overview of different related forms of control in Sect. 2.2. Next, Sect. 2.3 lays down the foundation for the rest of the chapter by presenting an overview of one of the existing perspectives on artificial intelligence that resonates most closely with self-aware computing systems.

After the introduction of the basics, Sect. 2.4 presents an overview of autonomic computing, which enables the construction of systems able to manage themselves in accordance with a set of high-level objectives specified by administrators or system users. Section 2.5 describes organic computing, which deals with the study of systems that dynamically adapt to changing conditions and exhibit a number of self-* properties, as well as context awareness. Next, Sect. 2.6 introduces service-based systems and cloud computing, including concepts such as location-transparent computation and autonomous services as agents. Section 2.7 provides an overview of self-organizing systems, which are able to organize themselves according to the laws of the environment within which they execute. Then, Sect. 2.8 introduces self-adaptive systems, which are strongly related to autonomic systems and able to adjust their own behavior in response to its perception of the environment and the system.

Section 2.9 introduces reflective computing and the notion of *computational reflection* as the system's ability to reason about its own resources, capabilities, and limitations in the context of its current operational environment. Next, Sect. 2.10

introduces *models at run-time*, that is, abstract self-representations of a system focused on a given aspect that may include its structure, behavior, or goals. This section also explores the relation between models at run-time and the concept of computational reflection presented in Sect. 2.9. Section 2.11 presents situation-aware and context-aware systems, in which the emphasis is made on building human–machine systems that observe, evaluate, and act within diverse situations that include a comprehensive set of factors that correspond to people, location, and events, as well as other environmental factors. Section 2.12 presents symbiotic cognitive computing, which are multi-agent systems that comprise both human and software agents that collectively perform cognitive tasks such as decision-making better than human or software agents can by themselves. Then, Sect. 2.13 covers auto-tuning, which deals with the automation of performance tuning, mostly for scientific applications.

After presenting an overview of different related areas and concepts, Sect. 2.14 provides a constructive definition of self-aware computing system that makes some considerations concerning the different factors influencing feasibility, capabilities, and ultimately determine under which conditions it is possible to actually develop a self-aware computing system, and how.

2.2 Control

In control theory, several advanced forms of control and adaptive control have been developed that involve learning, reasoning, and acting as well as models employed online as outlined for self-aware computing systems as introduced in Chap. 1. To compare self-aware computing systems with adaptive control architectures applied to software, we look at first into *model reference adaptive controllers* (*MRACs*) and *model identification adaptive controllers* (*MIACs*) in the following.

In case of *model reference adaptive controllers* (*MRACs*) [33, 37], a reference model defining desired closed-loop performance is employed to steer the adaptation. Consequently, the scheme is comparable to a prediction model of what is wanted that is used to steer the adaptation of the controller. However, as we have a prediction model of the plant only but not of the controller there is no process like learning involved, as the reference model is given at design time. The reference model is more a form of a given (high level) goal that is employed to steer the adjustments.

The *model identification adaptive controller* (*MIAC*) [37] scheme performs some form of system identification while the system is running, which can be compared to learning a model and then reasoning about the learned model to determine how to adjust the controller. However, we learn only a model only of the plant and not of the controller and therefore, if the plant is the context, we have context awareness only, and if the plant is a part of the system, we have self-awareness. As both cases are required for self-awareness according to Chap. 1, employing the MIAC scheme only leads to a self-aware computing system if the software and the environment are somehow subject to system identification.

Model-predictive control (*MPC*) [72] uses a model of the plant and a finite horizon for the predictions of the future output. The predicted outputs are employed to compute optimal set points (steady-state optimization). The optimal set points are then employed to calculate required control inputs to achieve the set points. When self-aware computing systems are compared with model-predictive control, architectures using a predictive model to plan the impact of future control actions such that the given criteria are optimized (according to goals) can be mapped to the reasoning and action. MPC can also be combined with system identification (cf. [40]) similar to MIAC as thus also a learning component is possible. However, MPC employing system identification learns a model only of the plant and not of the controller and therefore, if the plant is the context, we have context awareness only, and if the plant is a part of the system, we have self-awareness. As a link in the case of MIAC, both cases are required for self-awareness according to Chap. 1, employing the MPC with system identification scheme only leads to a self-aware computing system if the software and the environment are somehow subject to system identification.

Overall we can conclude that if the software and the environment are somehow subject to system identification, the system identification in control theory is comparable to the learning of self-aware computing systems. Also the MPC scheme of control theory can be seen as a special case of reasoning and acting (adapting) of self-aware computing systems. Finally, reference models in the MRAC scheme of control theory are a special case of static goals as considered by self-aware computing systems. Consequently, it can be argued that also self-aware computing systems in case they adapt the software behavior like less advanced forms of self-adaptive systems can likely largely benefit from the achievements of control theory. However, as also for the less advanced forms of self-adaptive systems principles and solutions of control can only be applied to software systems in restricted cases and the transfer of applicable control theory results to self-aware computing systems is still in its infancy.

2.3 Artificial Intelligence

There are many different perspectives on artificial intelligence, but the one that resonates most closely with self-aware systems is that adopted by Russell and Norvig in their book "Artificial Intelligence: A Modern Approach" [69], according to which artificial intelligence is fundamentally about designing and building rational agents. Wooldridge and Jennings [85] further define an agent as a *software-based computer system that enjoys the following properties*:

1. *autonomy*: agents operate without the direct intervention of humans or others and have some kind of control over their actions and internal state;
2. *social ability*: agents interact with other agents (and possibly humans) via some kind of agent-communication language;

3. *reactivity*: agents perceive their environment and respond in a timely fashion to changes that occur in it; and
4. *proactiveness*: agents do not simply act in response to their environment, and they are able to exhibit goal-directed behavior by taking initiative.

By emphasizing social ability as an essential property of agents, Wooldridge and Jennings suggest that agents typically exist in environments in which other agents are present, and that they interact with one another via some sort of agent-communication language, thereby forming multi-agent systems.

Self-aware computing systems as defined in this chapter possess the characteristics of autonomy, social ability, reactivity, and proactivity and can therefore be understood as types of agents or multi-agent systems that achieve these characteristics via the specific approach of learning models and using those models to determine how best to satisfy their goals.

2.3.1 Overview of Agents and Multi-agent Systems

A software agent can be defined, very generally, as a software entity that can accomplish tasks on behalf of its user, by acting within its environment [60]. In [69], agents are also referred to as *rational entities*, meaning that they would take the best possible action, considering available information and capabilities, to approach their objectives: "For each percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given evidence provided by the percept sequence and whatever built-in knowledge the agent has."

A wide variety of agent types, with more specific abilities and characteristics, has been defined within this vast area to address the particularities of different domains, based on different approaches. An extensive review of all agent types would be well beyond the scope of this chapter. We merely aim to highlight here the most relevant types that would help us compare multi-agent systems with self-aware computing.

We consider several dimensions of comparison, whereby agents can be either *deliberative* or *reactive*; *mobile* or *static*; and feature various combinations of key characteristics, such as *autonomy*, *learning*, and *social interaction*. In the context of self-aware computing, we are mainly concerned with aspects of autonomy, reasoning, learning, and social abilities. Hence, we will focus on discussing these next.

A *deliberative agent* is the "one that possesses an explicitly represented, symbolic model of the world, and in which decisions (e.g., about what actions to perform) are made via symbolic reasoning" [85]. Conversely, *reactive agents* reach their objectives by implementing a stimulus-response (or reflex) behavior, merely reacting to changes in their environment with corresponding actions. Hence, they do *not* posses symbolic representations or reasoning capabilities [15].

Russel and Norvig [69] refine this agent typology further, defining *goal-oriented* and *utility-based* agents. These correspond to deliberative agents that pursue goals in a binary manner—either achieving the goal or not achieving it—or in a more

modulated manner—where goal achievement can be equated to various degrees of utility. Russel and Norvig also refine reflex-based agents into basic and *model-based reflex agents*, which are reactive agents with or without internal state, respectively.

An agent's *autonomy* refers to its capability to operate without requiring human intervention, in order to achieve its objectives, or goals, on behalf of its user. In the context of deliberative agents, *proactiveness* is also considered as a key agent feature, related to its autonomy. It implies that the agent will be "taking the initiative" for reaching its goals, rather than simply reacting passively to its environment [85]. Of course, deliberative agents can also react to environmental changes.

An agent's *learning* ability allows it to adapt its behavior—e.g., via changes in its knowledge and reasoning, or in its reflexes—based on interactions with its environment, in order to increase its performance over time. Finally, an agent's *social ability* refers to its capability to interact with other agents, via some well-defined communication language.

A *multi-agent system (MAS)* consists of multiple agents that are engaged in some sort of interaction in order to accomplish one or several tasks, or goals. MAS is typically employed to address complicated computing problems via a divide-and-conquer technique—i.e., dividing the problem among a set of (specialized) agents, which interact to compose partial results into a global solution. In the case of deliberative agents, this implies that knowledge representation, acquisition, and reasoning processes are also distributed among the agents.

2.3.2 Comparison with Self-aware Computing

Since the concept of *agent* has been used rather broadly across various applications and domains, it has become an umbrella term for a wide variety of computing entities that feature highly different capabilities and characteristics. Therefore, it is quite difficult to provide an exact comparison of multi-agent systems with self-aware computing systems, not at least since these later can also feature different kinds and levels of self-awareness (Chap. 3). Considering these reasons, we only attempt here to provide a general comparison, highlighting the main differences in focus between the two concepts.

The concept of a *self-aware computing system* (as defined in Chap. 1) is mostly compatible with that of a *deliberative agent*, which features autonomy, learning, and social abilities—i.e., a "smart agent" in [60]. Indeed, like deliberative agents, self-aware computing systems can possess models of the world that are explicitly represented and on which they can reason in order to achieve higher-level goals (representing the user). In addition to an agent's world models, self-aware systems must also possess models of themselves and must reason on these to perform actions—e.g., self-adaptation to ensure system autonomy in a changing environment; explaining and reporting their current states (and their probable causes) to users, or to other systems; or suggesting means of rectifying undesirable or suboptimal states. Consequently, the learning capabilities of self-aware systems must apply to both models

representing their environments and themselves. Here, self-aware systems focus on the particular problem of agent autonomy, within a changing environment and/or in the presence of internal faults, rather than on problem-solving in general, as is the case for multi-agent systems.

Like social agents, self-aware computing systems may also interact with other systems, either by direct communication or by indirect influence within a shared environment. The systems that such a self-aware system interacts with may feature various levels of self-awareness, or may be non-self-aware. In case of direct communications, a self-aware system's interactions can be equated to agent communications (and hence represent social skills). A specific feature of self-aware computing systems consists in the extent to which they can be, or become, self-aware of the other systems that they interact with—e.g., acquiring and maintaining models of them. This can also be the case in some agent-oriented approaches, like with game theoretical agents, yet here the agents' awareness of each other is typically provided at design time, then potentially refined during run-time. Another interesting feature here consists in the lack of assumptions on the other systems' self-aware capabilities (i.e., heterogeneity of self-awareness levels across a collective of systems). Again, this can be the case in some multi-agent systems—such as some game theoretical cases—yet the agent's self-awareness levels are typically predefined, depending on their roles.

2.4 Autonomic Computing

The autonomic computing initiative [35] was spurred by a concern that rapid growth in the complexity of IT systems would outstrip the ability of IT administrators to cope with that complexity. The proposed solution was for the system to take upon itself a large portion of the management burden. Just as the autonomic nervous system governs our pulse, our respiration, and the dilation of our pupils, freeing our conscious brain to attend to higher-level cognitive functions, the goal of autonomic computing is to create computing systems that manage themselves in accordance with high-level objectives from administrators or system users. While initially conceived as a paradigm for the future of IT management, over the course of time the principles, objectives, and techniques of autonomic computing have come to be applied more broadly, extending to physical systems such as data centers (and data center robots), the Internet of things, and smart homes.

An early paper that outlined the vision and research challenges of autonomic computing [41] laid out an architecture in which autonomic behavior was exhibited at two levels. Autonomic elements (such as databases, Web servers, or physical servers) were envisioned to use a combination of monitoring, analysis, planning, and execution driven by knowledge (often referred to as the *MAPE-K architecture*

or *MAPE-K loop*) to accomplish their own individual goals.[1] System-level autonomic behavior was to be driven by system-level goals and accomplished through well-designed interactions among multiple interacting autonomic elements whose individual goals might be designed to support the desired system-level behavior. The vision did not specify how the goals of autonomic elements might be derived from system-level goals, nor did it specify how to design the interactions among the autonomic elements; these were cited as difficult and important research challenges.

Comparing the definition and vision of autonomic computing systems to that of self-aware computing reveals several similarities and a few distinctions. Employing Knowledge to support the Monitoring, Analysis, Planning, and Execution functions matches very closely the second clause of the self-aware computing definition, which states that self-aware systems "reason using ... models ... enabling them to act." Contained within the Knowledge component of an autonomic element are one or more models that the Analysis component can use to anticipate the likely consequence of an action or a plan (a sequence of actions) that it is contemplating. The objective of the Planning component is to move the autonomic element (or perhaps the autonomic system in general) from its current state (as assessed by the Monitoring component) to a state that it is deemed more desirable according to the high-level goals, which are also held within the Knowledge component. One common approach to using models and high-level goals to drive the behavior of autonomic elements and systems is utility functions. The state space is described in terms of attributes that the administrator deems important (e.g., response time and power consumption), a utility value is ascribed to each possible state, and the system selects an action that would (according to models) lead to a state with the highest achievable utility value, given the current resource of other constraints. Finally, regardless of the means by which analysis and planning are accomplished, the autonomic element Executes the action or plan deemed most desirable by the Planning component, the state of the autonomic element (or the autonomic system) evolves (either in reaction to the action(s) or an external change such as an increase in workload), and the MAPE-K process continues. The execution step is the one point at which the autonomic computing definition may differ from the *reasoning* clause of the self-aware computing definition. Autonomic computing *requires* execution, while self-aware computing *permits* execution but does not require it. Nonetheless, in practice the field of autonomic computing embraces work in which the system recommends an action, but allows a human to judge whether or not to take it, viewing this as an important and necessary evolutionary step toward full-fledged autonomic computing, not just as a matter of making incremental technological progress, but also as a means for building user trust.

[1] In actuality, MAPE-K was not strictly an architecture (it was more of a statement about required functionality than it was a statement about how those functions were to be woven together) nor was it necessarily a loop, as the various components might typically be operating in parallel at all times and not running in a strict order.

The first clause of the self-aware computing definition concerns learning. Learning has always been viewed as an important aspect of autonomic computing, and a preferred means by which models are created, but autonomic computing does not strictly require that an element or a system learn to be regarded as autonomic.

To summarize, while autonomic computing was initially proposed as an IT management solution, the current understanding of the term is much broader, and it overlaps strongly with the definition of self-aware computing systems. The main differences are that autonomic systems are not strictly required to learn, and self-aware systems are not strictly required to act.

2.5 Organic Computing

An organic computing (OC) system is "a technical system which adapts dynamically to the current conditions of its environment. It will be self-organizing, self-configuring, self-healing, self-protecting, self-explaining, and context-aware" [58].

From its inception, OC started with a strong industry pull (including Daimler-Crysler, Siemens, and Bosch) because of the shared belief across several industries that we can no longer adequately design very large-scale, complex systems; complex systems need to help us by designing parts of themselves and by managing parts of themselves.

As part of this was the strong recognition by OC that complex systems have emergence. That is, they have unplanned and unexpected side effects and emergent properties at different levels because of the interactions among large numbers of components under different operational conditions. The OC attitude is "How can we take advantage of the fact that complex systems have emergence?" How can systems use emergence as a source of controlled variation? How can we shape emergence to go in desired directions?

Hence, from its inception, OC emphasized the importance of having systems that could not only observe and adapt to the changing and demanding external world, but also could observe and adapt their own goals, plans, resources, and behaviors as necessary to correctly map to new contexts and requirements. Moreover, in OC approaches, one will take advantage of this self-awareness to adapt to not only changing conditions and requirements, but even to new, emergent properties in the system and its environment.

Although OC has different approaches to meeting the challenges of creating self-adaptive and self-aware systems, the observer/controller architecture is an especially important contribution to mention here because of its clear relationship and similarity to several of the architectures in this book (see more in Chaps. 6 and 8). An early description of the observer/controller architecture is depicted in Fig. 2.1.

A key emphasis in OC is that complex systems need to have self-control and self-adaptation abilities while always retaining important human-in-the-loop capabilities so that humans can suitably monitor and control when necessary the results of interacting and relatively autonomous computing systems. Hence, the observer/controller

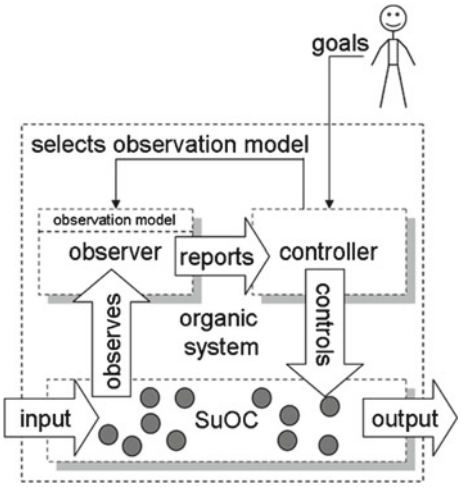

Fig. 2.1 Early observer/controller architecture

architecture is comprised of two top-level concepts: the organic system and a human user, where the organic system adheres to the basic input/compute/output principle of computing. The human user is seen as imposing goals and constraints at times on the organic system, while reviewing the system status based on the OC system's self-reporting capabilities and whatever special human interfaces to system instrumentation have been added.

The organic system is further decomposed into three major components: the system under observation and control (SuOC), the observer, and the controller. All human interaction is relayed by the controller. Notably, the input/compute/output principle is realized by the SuOC. Observer and controller impose a feedback loop onto the SuOC, where the first observes the SuOC and reports to the controller, which in turn controls the SuOC.

An important characteristic of the SuOC in organic computing is that it is comprised of agents, i.e., autonomous entities. In other words, the SuOC is already a set of self-organizing systems. The observer and controller enhance this system to achieve controlled self-organization.

As can be seen by this very brief description, there can be multiple observer–controller layers in a given system. Furthermore, different kinds of self-awareness capabilities, as discussed in the rest of this book, can contribute at many points in this architecture; they will certainly occur in the observational and reasoning capabilities of the *observer*, as well as potentially in the adaptive behaviors directed by the *controller*.

2.6 Service-Based Systems and Cloud Computing

In this section, we first introduce some basic concepts related to service-oriented computing, followed by an overview of the area of cloud computing, emphasizing concepts relevant to self-aware computing systems, such as location-transparent computation and the notion of autonomous services as agents.

2.6.1 Service-Based Systems

Service-oriented computing (SOC) and service-oriented architecture (SOA) are now largely accepted as well-founded reference paradigm and reference architecture for Internet computing [62]. Under SOC, networked devices and their hosted applications are abstracted as autonomous loosely coupled services that, while playing the roles of service providers, consumers (aka clients), and registries, they also interact by following the service-oriented interaction pattern (see Fig. 2.2).

According to this pattern, a service has to define an interface publishable on the Internet, researchable, and callable independently from a particular language or platform. In order to obtain these requirements, a SOA application has to define roles (not all required) as shown in Fig. 2.2.

- *Service Consumer*: the entity that uses the service; it can be an application module or another service;
- *Service Provider*: the entity that provides the service and exposes the interface;
- *Service Contract*: defines the format for the request of a service and the related response;
- *Service Registry*: Directory on the Internet that contains the services.

Despite the remarkable progress of the SOC paradigm and supporting technologies in the last ten years, substantial challenges have been set through the evolution of the Internet. Over the years, the Internet has become the most important networking infrastructure, enabling all to create, contribute, share, use, and integrate information and extract knowledge. As a result, the Internet is changing at a fast pace and is called to evolve into the Future Internet, i.e., a federation of self-aware services

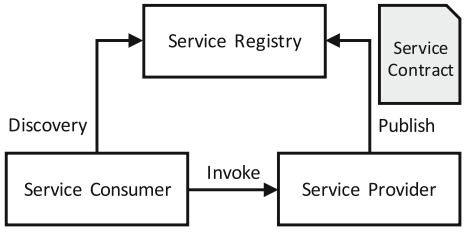

Fig. 2.2 Service-oriented interaction pattern

and networks that provide built-in and integrated capabilities such as service support, contextualization, mobility, security, reliability, robustness, and self-* abilities of communication resources and services [28, 38].

In this wide spectrum, a SBS can be meaningfully seen as a composition of service providers and consumers that interact by providing/requiring functionalities to/from each other. A SBS is often opportunistically built for the purpose of achieving a given goal. The goal typically expresses functional and non-functional high-level requirements that the resulting composition has to fulfill. The former class captures the qualitative behavior of a SBS, its functional specification. The latter defines the SBS's quantitative attributes such as performance, reliability, and security.

From a software engineering perspective, goal changes are always done to meet the new requirements; e.g., users and involved business organizations may change their functional needs and non-functional preferences. Moreover, it can be that the services currently involved in the composition no longer perform as expected. On the practical side, the source of this type of run-time changes can be, e.g., changing conditions of the network through which services communicate, degrading computational resources of the execution environments where services are deployed, upgrading the version of the middleware on top of which services run, and remote service substitution.

The knowledge that service consumers have depend on the contract (often expressed by means of service behavioral models) exposed by the service providers they want to interact with (interface only, interface plus interaction protocol, interface plus interaction protocol plus non-functional attributes, etc.). As a consequence, also the kind of reasoning that enables a SBS to act based on its knowledge depends on the kind of models and notations used to describe service contracts.

Last but not least, since a SBS can be seen as a composition of services, the way the system can act to enable self-awareness is constrained by the structure and behavior of the adopted composition means. In particular, two forms of composition to build SBSs can be distinguished, one centralized, i.e., service orchestration, and one distributed, i.e., service choreography [5].

2.6.2 Cloud Computing

Cloud computing refers to the on-demand delivery of IT resources and applications via the Internet, possibly with a pay-as-you-go pricing. By referring to the NIST definition of cloud computing [53], "cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service-provider interaction." In other words, cloud computing is essentially about moving services, computation, and data off-site to a location-transparent entity. Cloud computing distinguishes three service models, as described below:

- *Software as a Service* (*SaaS*): WAN-enabled application services (e.g., Google Apps, Salesforce.com, and WebEx). The capability provided to the consumer is to use the providers' applications running on a cloud infrastructure. The applications are accessible from various client devices through either a client interface, such as a Web browser (e.g., Web-based email), or a program interface.
- *Platform as a Service* (*PaaS*): Foundational elements to develop new applications (e.g., Coghead and Google Application Engine). The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider.
- *Infrastructure as a Service* (*IaaS*): Providing computational and storage infrastructure in a centralized, location-transparent service (e.g., Amazon). The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources, where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications.

Some of the main characteristics of cloud computing are concerned with (i) elasticity, in which it requires on-demand capabilities of resources; (ii) broad network access, in which access to the cloud can be done using any computer-based device; (c) resource pooling, in which data can be used and added in the cloud at any time; (d) measured services, in which consumers only pay for the resources they use from the cloud; (e) energy efficiency, in which the energy consumption of cloud data centers are optimized; and (f) virtualization, in which the infrastructure is divided and seen as separated logic components.

The above characteristics of cloud computing require a degree of self-awareness of the technology. For example, it is necessary for the system to be aware of the need of new resources and to be able to free the not-used resources at a certain moment of time. However, it is not possible to say that cloud computing technologies have the necessary level of self-awareness, as per the definition given in Chap. 1.

2.6.3 Comparison with Self-aware Computing

Since the vision of Weiser [81] was published almost 25 years ago, pervasive systems have almost become reality. Computers have become ubiquitous and are available in areas nobody would have expected them 20 years ago such as cars, parks, or even pot plants at home. Nevertheless, these computers are often far from being self-aware. In many cases, these computers act as simple sensors merely storing the sensed environment on a local memory or transmit it to a central server. The two main points in the self-aware computing definition are often not fulfilled. Pervasive systems only in some cases *learn* about their environment but they only rarely reason about this knowledge.

Nevertheless, there are novel areas of research within the pervasive computing community, such as the *smart environment* community. Here the devices try to learn

behavioral patterns about the user in order to anticipate certain actions, requirements, or desires of the user. This anticipation is often founded on predefined rules and only allows very limited flexibility with respect to defining new goals for the individual device or the entire system. Furthermore, the interplay between the individual devices and the impact of their actions on each other is often hardwired within the individual devices. This limits the capabilities to include new devices during run-time without explicit setup of the system. While implementing individual autonomous computing agents within the different devices and using a service-based approach introduce higher flexibility and robustness to the pervasive system, the higher, system-wide goals are still not considered when actions of the individual devices are performed.

2.7 Self-organizing Systems

As their name suggests, *self-organizing systems* are systems that are able to organize themselves adaptively and without external control. *Organization* is at the core of this definition and generally comprises the relations, interconnections, conditionality or dependencies between the system's components, or variables. Hence, organization relates heavily to the system's *structure*, defining its main components and their interrelations.

In the remainder of this section, we first present a general overview of self-organizing systems, followed by a discussion of cross-pollination opportunities with self-aware computing.

2.7.1 Overview of Self-organizing Systems

From a general perspective, if a system (or general "machine" [4]) is viewed as a set of states S, with a set of inputs I and a function f that maps $I x S$ into S—i.e., determining the system's future state based on the current state and inputs—then the system's organization represents the manner in which its variables are interrelated via the mapping function f. A self-organizing system here implies that the system is able to change its own mapping function. This raises some controversies around the system's boundary definition, since it implies the extension of the initial system with a controller that monitors and updates its organization [4].

However, most often, self-organization is understood as a dynamic adaptive and autonomous process that results from the inherent behavior of each system component and of the "laws" of the environment within which they execute [4, 20]; and which results in a progressive increase in system structure [84]. Examples of natural self-organization include the spontaneous assembly of protons, neutrons, and electrons into atoms; of different atoms into organic molecules; and the evolution of living organisms adapted to their environments. Examples in artificial systems include the adaptive formation of ad hoc mobile networks, of robot swarms, and of

component- and service-based software system assemblies. Ashby prefers referring to this type of self-organization as the "spontaneous generation of organization."

This is also the typical understanding of self-organization in the computing systems domain, notably in research communities such as the self-adaptive and self-organizing (SASO) systems—as reflected for instance in the proceeding series of the International Conference on SASO Systems.[2] Here, self-organization is interesting because of the advantageous properties that it features in general (e.g., resilience and adaptability to a wide range of environmental changes; robustness in the face of internal failures; and scalability with the number of components and adaptation frequency). In the SASO community, self-organization is also seen as a bottom–up alternative to achieving self-adaptation, which was originally designed as top–down.

The main challenge here is: How to design self-organizing systems that also meet desirable goals? Indeed, in natural systems, most instances of self-organization have no other obvious purpose than their own existence and survival within their environment. In more "interesting" cases (from a goal-oriented system perspective), different organisms self-organize into more-or-less temporary formations in order to achieve via collective action a common goal that none of them could have achieved individually (e.g., swarms, flocks, herds, teams, and societies). Yet, when building artificial systems, determining which component behaviors and interaction laws will lead to the self-organization of systems that meet the designer's goals within targeted environments is a difficult task, subject to active research. These challenges differ from those highlighted by self-aware computing, where the research focus is placed on the system's knowledge acquisition and the way in which usage of this knowledge can serve the system's achievement of goals.

2.7.2 Cross-pollination Opportunities with Self-aware Computing

In self-organizing systems, any knowledge available is decentralized and distributed across the participating system components, or agents. An exception may occur if global knowledge were encoded within the environment shared by the system's components. This aspect will be interesting to study within the context of decentralized (or self-organized) self-aware systems.

Conversely, it will be interesting to explore how self-awareness could help a system's components self-organize in order to achieve a shared goal. Here, the hard-coded elementary behaviors and "laws" of the environment that fuel self-organization could be adapted dynamically by the system components, as they become aware of their shared goals (e.g., already the case in social organizations). Also, components that become aware of their own characteristics (e.g., range of behaviors and properties they can exhibit), of the characteristics of other components, and of the key theoretical principles of self-organization (still to be produced by the corresponding research

[2] SASO history in 2016: http://saso2016.informatik.uni-augsburg.de/history.html.

fields) may be better able to select the components with which they connect in order to have a better chance of achieving their goals.

2.8 Self-adaptive Systems

In the self-adaptive software community, self-* properties are organized into levels where *self-adaptiveness* is at the top (or general level), while self-awareness is considered only a primitive level like context awareness and the typical autonomic computing like self-* properties such as self-configuring, self-healing, self-optimizing, or self-protecting are considered major level properties in between the other two levels (cf. [70]). Furthermore, in the self-adaptive software community most approaches emphasis an architectural perspective (cf. [70]) where besides the control of parameters changes of the architecture may matter.

In the rest of this section, we first present a general overview of self-adaptive systems, followed by a discussion of anticipatory self-adaptive systems, which are those that exhibit a specific set of characteristics which are strongly related to self-aware computing systems, such as the ability to predict, or self-adapt proactively.

2.8.1 Overview of Basic Self-adaptive Systems

Like in autonomic computing for self-adaptive systems, control loops are often considered one of the core objects of the design efforts [16, 74] and it is advocated that in order to achieve real self-management capabilities besides a direct layer for change management also a goal management is required (cf. reference architecture [44]). However, besides some specific approaches that emphasize architectural models or goals in contrast to the notion for self-aware computing systems of Definition 1.1 for the basic efforts for self-adaptive systems hold that neither the learning of models nor the capability to reason based on this models to realize the adaptation loop has been emphasized so far.

In a series of Dagstuhl seminars, the community has identified mainly modeling dimensions, requirements, engineering through feedback loops, assurances, the design space, processes, decentralized control, and practical run-time verification and validation as the main issues that have to be addressed (see two research road maps [18, 21]. However, again neither the employed knowledge nor the capability to reason based on this knowledge as advocated by the notion for self-aware computing systems of Definition 1.1 plays a prominent role.

The notion for self-aware computing systems of Definition 1.1 is overlapping with the notion of self-adaptive software as it also covers systems where no self-adaptation happens. As advocated in [31], the limitation to only fully automatic adaptation is probably too limited and instead, it would be better to also consider related manual activities such as change management and their coordination with

automated adaptation steps. Therefore, to include also mixed forms where people supervise the adaptation or the self-awareness helps with manual adaptation in the notion for self-aware computing systems of Definition 1.1 seems somehow beneficial to better cover the real needs and the real design options.

For the subset of self-aware computing systems that realize some self-adaptation behavior, however, we can conclude that they describe a subset of the self-adaptive software, where in addition to the existence of the feedback loop we also learn models capturing knowledge and reason about these models allowing them to act according to internal and external conditions in accordance with higher-level goals. While several suggestions go in a similar direction as Definition 1.1 (cf. [44]), the community will likely benefit from the suggested notion for self-aware computing systems of Definition 1.1 that clearly separate lower-level solutions without explicit knowledge capturing and reasoning from approaches that have these capabilities based on learning models and reasoning based on the models.

2.8.2 Anticipatory Self-adaptive Systems

What distinguishes a *self-adaptive system* from any other system is its ability to adjust its behavior in response to its perception of the environment and the system itself [18]. Self-adaptive systems typically operate employing a knowledge base that can incorporate an explicit representation of the system's structure, goals, and assumptions about its environment. However, there is an ample variation in the level of detail in which the different elements of this knowledge base are described, as well as in the reasoning capabilities that different approaches exhibit [70].

The characteristics of early proposals to self-adaptation [32, 44] tend to be far from the traits of self-aware computing systems listed in Definition 1.1. These approaches tend to be reactive and adapt in response to the changes without anticipating future changes or reasoning about the long-term outcome of adaptation (e.g., a system may adapt to a transient change, only to adapt again and go back to its original configuration moments later). Moreover, these proposals tend to be rather limited in terms of learning capabilities. In contrast, recent approaches to self-adaptation [17, 19, 34] are better aligned with the description of self-aware computing system given in Chap. 1. The general trend among these proposals is a paradigm shift from reactive to proactive adaption, incorporating the ability to learn, predict, and systematically exploit knowledge to improve the operation of the system.

These approaches fit well into the category of *anticipatory self-adaptive system*, defined as "able to anticipate to the extent possible, its needs and behaviors and those of its context, and able to manage itself proactively" [63]. Based on this definition, we can identify the main criteria that anticipatory self-adaptive systems should ideally satisfy:

1. *Predictive*. The system can likely determine *ahead of time* if a condition that requires adaptation will take place. Predictions can be exploited to avoid unnec-

essary adaptations or improve the overall choice of adaptation (e.g., by factoring in information about future resource availability or workload and other environment conditions into the decision-making process [19, 55]). Predictions can also help to enforce safety properties when reachability of a potential safety violation from the current state of the system is detected [48].

2. *Proactive.* The system can enact adaptation before a deviation from its functionality or qualities takes place. A representative proactive approach to self-adaptation in cyber-security is Moving Target Defense (MTD) [86]. MTD assumes that a system that remains static with the same configuration over long periods of time gives potential attackers time for reconnaissance and exploitation of system weaknesses. Hence, the idea behind MTD is adapting to change the configuration of the system periodically, thus reducing the chance of an attacker of finding and exploiting a weakness. Another example of proactivity is latency-aware proactive self-adaptation (PLA) [55], which anticipates changes in environment conditions and triggers adaptations with enough lead time to deal with them in a timely fashion, based on information about the execution time required to complete adaptations and achieve their effects in the controlled system (i.e., their latency). In the area of service-based systems, PROSA [34] is an approach that carries out tests at run-time to detect potential problems before they happen in real transactions, triggering adaptations when tests fail.
3. *Learning*. The system can generate and incorporate new knowledge (typically derived from observations of the system and its environment at run-time), and use it to improve subsequent adaptions. Simple forms of learning can also be found in reactive approaches. To select adaptations, Rainbow [32] employs information about the actual outcome of past adaptations to derive probabilities that represent the likelihood of possible outcomes of future adaptations. Proactive approaches can employ more sophisticated learning techniques to leverage its prediction capabilities (e.g., employing Bayesian learning to estimate the future behavior of the environment [17, 27]).

Table 2.1 categorizes some anticipatory approaches to self-adaptation. It is worth noticing that although a proactive self-adaptive system can benefit significantly from predictions, proactive approaches are not necessarily predictive. One example is MTD. In the simplest form of MTD, the system's configuration is changed proactively with a fixed frequency, without any reasoning involving a model of the envi-

Table 2.1 Anticipatory self-adaptation approaches

Approach	Learning	Predictive	Proactive
KAMI [27]	✓	✓	✓
QoSMOS [17]	✓	✓	✓
Cheng et al. [19]		✓	✓
PLA [55]		✓	✓
Li et al. [48]		✓	✓
PROSA [34]			✓
MTD [86]			✓

ronment or predictions about its future behavior. Moreover, we can observe that in terms of learning, anticipatory self-adaptive approaches are still far from the ideal of self-aware computing systems. In particular, learning capabilities are employed only in approaches that involve relatively simple adaptations (e.g., parameter optimization [17, 27]), but not combined with adaptations that entail complex changes to a system's architecture.

2.9 Reflective Computing

In 1987, Maes [51] defined and implemented "computational reflection" as "the process of reasoning about and/or acting upon oneself." Computational reflection is an engineered system's ability to reason about its own resources, capabilities, and limitations in the context of its current operational environment. Reflection capabilities can range from simple, straightforward adjustments of another program's parameters or behaviors (e.g., altering the step size on a numerical process or the application of rules governing which models are used at different stages in a design process) to sophisticated analyses of the system's own reasoning, planning, and decision processes (e.g., noticing when one's approach to a problem is not working and revising a plan).

Reflection processes must include more than the sensing of data, monitoring of an event, or perception of a pattern; they must also have some type of capability to reason about this information and to act upon this reasoning. However, although reflection is more than monitoring, it does not imply that the system is "conscious." Many animals demonstrate self-awareness; not only do they sense their environment, but they are also able to reason about their capabilities within that environment. For example, when a startled lizard scurries into a crevice, rarely does it try to fit into a hole that is too small for its body. If it is injured or tired, it changes the distance that it attempts to run or leap. This adaptive behavior reveals the ability of the animal system to somehow take into account the current constraints of the environment and its own body within that environment [9, 10].

In order to bring out the ways in which the self-awareness processes and architectures enhance and further develop reflective architectures, we will quickly overview one approach to implementing computational reflection and the building of reflection processes in a robotic-car example (also see Chap. 9 for additional discussion of self-modeling issues in this test bed).

The Wrappings' approach uses both explicit meta-knowledge and recursively applied algorithms to recruit and configure resources dynamically to "problems posed" to the system by users, external systems, or the system's own internal processing. The problem manager (PM) algorithms use the Wrappings to choreograph seven major functions: discover, select, assemble, integrate, adapt, explain, and evaluate. "Discover" programs (or as called in the Wrappings, "resources") identify new resources that can be inserted into the system for a problem. "Selection" resources decide which resource(s) should be applied to this problem in this context.

"Assembly" is syntactic integration and these resources help set up selected resources so that they can pass information or share services. "Integration" is semantic, including constraints on when and why resources should be assembled. "Adaptation" resources help to adjust or set up a resource for different operational conditions. "Explanation" resources are more than a simple event history because they provide information on why and what was not selected. "Evaluate" includes the impact or effectiveness of the given use of this resource in the current problem. The meta-knowledge for a Wrapping is always for the USE of a resource within a particular context and for a specific posed problem. It includes assumptions and constraints, the required services and input, the resulting services and output, and the best practices for using this resource in this situation.

The Wrappings' "problem-posing" has many benefits, including separating problems from solution methods and keeping an explicit, analyzable trace of what problems were used to evoke and configure resources. Because all of the resources are wrapped, even the resources that support the Wrappings' processing, the system is computationally reflective—it can reason about the use of all of its resources [12].

Wrappings [45, 46] provide an implementation strategy for computational reflection that provides control over the level of self-awareness available in the system and the levels of self-awareness to be used at any given time. The mechanism that allows this flexibility is the Problem-Posing Programming Paradigm, which strictly and completely separates the information service requests (the problems) from the information service providers (the resources) and reconnects problems in context with resources using explicit interpretable rules collected into Wrappings' Knowledge Bases. Moreover, the processes that perform the connection (called PMs, or problem managers) are also resources and are also Wrapped, so they can be swapped out as easily as any other resources. We emphasize that the designers have control over the level of detail of decomposition of the processes in the system, and of the rules by which resources are used for particular problems. There is no inherent limit on that level of detail (some implementations go down to the individual hardware instruction, but most go to the typical software component/module level). More detail on the implementation architecture is given in Chap. 8.

The flexibility of the Wrappings' approach provides multiple entry points for the reflective processes. A reflective resource has the general form: Given a goal, purpose, or function, a reflective process uses the sources of information to do some action, decision or to create data that is used by other processes. The goal or function for that reflective process could be built in during design time or assigned dynamically to that reflective process by other programs. It may be in continual use or it may be recruited or evoked only when certain resources are active or conditions exist. The sources of information can be, e.g., data sets, sensor output, or monitors. The reasoning process for reflection can be done with an algorithm, decision process, rulebase, cognitive model, or planner. The resulting actions are myriad, but include sending messages, setting program or context parameters, recruiting new components, initiating new processes, or instigating a replan or undo process.

Although the Wrappings' approach and reflective architectures approach briefly outlined here have proven its value for resource management and dynamic inte-

gration among large numbers of resources, the original approach was in practice limited to largely the management and adaptation of single large distributed systems. Although, the benefits of reflection were clear for interactions among systems (e.g., the self-knowledge could be made available to other systems for coordination [11] and external viewpoints by other systems could help a system identify its own problems or state and learn better [8, 47], in fact, the new work in self-aware systems as seen in this volume will help greatly by expanding new ideas for how collections of self-aware systems could interact and organize.

2.10 Models@run.time and Reflection

A model at run-time (models@run.time) [14] is defined as an abstract self-representation of a system that is focused on a given aspect of the running system. Such aspects include its structure, behavior, and goals. The run-time model exists in tandem with the given system during the actual execution time of that system. As in the case of traditional model-driven engineering (MDE) [30], a self-representation of the system in the form of run-time models can also be used as the basis for software synthesis, but in this case the generation can be done at run-time [57, 82].

Before describing the role of models@run.time in the area of self-aware computing, it is useful to briefly introduce the relationship between models@run.time and reflection (the topic reflection is more extensively covered in Sect. 2.9) and other aspects. Computational reflection focuses on the representations of an underlying system that are both self-representations and causally connected [14]. The causally connected representations of aspects of the system are constantly mirroring the running system and its current state and behavior. Causal connection implies that if the system changes, the self-representations of the system (i.e., the models) should also change, and vice versa.

Even if closely related, models@run.time and reflection are not the same. Reflection deals with models that are linked to the computation model and therefore tend to be focused on the solution space and in many cases at a rather low level of abstraction. The research area models@run.time deals with models that are defined at a much higher level of abstraction. Further, run-time models more frequently relate to the problem space. Examples of applications using run-time models are self-adaptation [57] or generation of mediators to support interoperability [12].

Traditionally, the structure of a run-time model has been conceived at design time (e.g., architecture models [57]). However, they can also be learned at run-time. In [12], the authors show how using learning methods the required knowledge of the context and environment can be captured and distilled to be formulated and made explicitly available as a run-time model and therefore support reasoning. Another example of techniques to be used to learn run-time models are shown in [87].

Models@run.time are at the core of self-aware systems. They are relevant to support self-awareness as defined in Chap. 1. (i) The run-time models correspond to the learned models which capture knowledge about the system itself and their envi-

ronment. Specifically, the run-time models support learning to capture the needed knowledge about the system itself (e.g., its own goals and requirements [71, 82]) or its perception of the environment [77, 87]. (ii) The run-time models when consulted should provide up-to-date information about the system and therefore support reasoning (e.g., predict, analyze, and plan) enabling the system to act based on their knowledge. As the run-time model is causally connected, actions taken based on the reasoning can be made at the model level rather than at the system level [56].

We argue that the definition of self-awareness requires a self-representation (i.e., run-time model) of the subject of awareness. For example, if the system is aware of its own architecture the system would need a representation of its architecture (a architecture run-time model). Other examples are awareness of its own requirements or any other aspect about itself. If the object of the awareness is part of the environment of the system (i.e., it is outside the system), it should be considered a self-representation as well as the representation includes the perspective of the system. Two different systems will usually have different representations (or models) of their perception of the same object of awareness.

2.11 Situation-Aware Systems and Context Awareness

Situation awareness (SA) is an ongoing body of research with many conferences, workshops, and papers which develops theory and applications in building human–machine systems that observe, evaluate, and act within diverse situations. Here we are using the term "situation" in the technical sense [6, 23, 50] where a situation includes at least the elements of the situation, e.g., objects, events, people, systems, and environmental factors, and their current states, e.g., locations and actions.

Fracker [29] described SA as the combining of new information with existing information for the purpose of developing a "composite picture of the situation along with the projections of future status and subsequent decisions as to appropriate courses of action to take." Dominguez et al. [23] added to this view an emphasis on the "continuous extraction of environmental information" with the explicit feedback loop that would use the developed perceptions and understanding to direct the next collection of data.

Credited with seminal work in this field, Endsley [24] argues that "it is important to distinguish the term situation awareness, as a state of knowledge, from the processes used to achieve that state. These processes, which may vary widely among individuals and contexts, will be referred to as situational assessment or the process of achieving, acquiring, or maintaining SA." Thus, in brief, situation awareness is viewed as "a state of knowledge," and situational assessment as "the processes" used to achieve that knowledge. Endsley's model illustrates three stages or steps of SA formation: perception, comprehension, and projection. Perception is considered Level 1 of a SA system. "The first step in achieving SA is to perceive the status, attributes, and dynamics of the relevant elements in the environment. Thus, Level 1 SA, the most basic level of SA, involves the processes of monitoring, cue detection, and simple

recognition, which lead to an awareness of multiple situational elements (objects, events, people, systems, and environmental factors) and their current states (locations, conditions, modes, and actions)."

By this framework, Level 2 in a SA is comprehension and is a synthesis of the Level 1 SA elements through the "processes of pattern recognition, interpretation, and evaluation. Level 2 SA requires integrating this information to understand how it will impact upon the individual's goals and objectives. This includes developing a comprehensive picture of the world, or of that portion of the world of concern to the individual." The highest level of SA, Level 3, is "projection" or the ability to predict the future actions of elements in the environment. "Level 3 SA is achieved through knowledge of the status and dynamics of the elements and comprehension of the situation (Levels 1 and 2 SA), and then extrapolating this information forward in time to determine how it will affect future states of the operational environment" [24]. With SA, one does not guarantee successful decision-making, but does provide some of the necessary inputs, it is argued, for successful decision-making with cue recognition, situation assessment, and prediction. As in self-aware systems, goals play a key role in SA. Both multiple goals, the fact of competing goals, and goal prioritization are emphasized in SA. However, it appears that for most SA systems these goals are "given" to it and predesigned, whereas in self-awareness (as shown in Chap. 3) although there are certainly goals given to a self-aware system, it is also expected that the self-aware system will alter and adapt even high-level goals and possibly generate low-level goals.

Many researchers have discussed the limitations of the current SA approaches, noting especially that the most widely cited models of SA lack support from the cognitive sciences (Banbury and Tremblay, [6]) and that there is also important mathematical and logical work to be done in defining these terms computationally (M. Kokar, [59]). In terms of self-awareness processes, we would say that SA has not yet incorporated the same sophistication (e.g., in learning, model-building) to its internal models that it applies to its external models of the situation. That is, as clearly seen from SA research, although SA certainly includes cognitive processes such as "mental models," attention, and decision-making, there has historically in SA been less of an emphasis on any reflection processes or self-models as used in this volume. Although models are emphasized for use by the cognitive/intelligent processes for situation awareness, these models are not explicit models of the system itself, its reasoning and learning capabilities, or its limitations, but rather focus on the objects and the situations to be perceived. It appears from Endsley and other SA-leading researchers that they are making some assumptions about what is useful in terms of their class of problems. While in self-aware systems, we are recognizing the need for both short-term and longer-term processes, it appears that SA is focused more on immediate and fast responses, proceeding from pattern recognition of key factors in the environment—"The speed of operations in activities such as sports, driving, flying, and air traffic control practically prohibits such conscious deliberation in the majority of cases, but rather reserves it for the exceptions." From Endsley [26], it would appear that SA views some of the cognitive processes that build models as largely "backward focused," forming reasons for past events, while

situation awareness is typically forward looking, projecting what is likely to happen in order to inform effective decision processes. In self-awareness, we see the benefits for learning, understanding, and model-building processes as leading to more adaptive behavior in the long-term certainly, and even leading to better behavior at run-time in accordance with the real-time requirements.

Related to SA is the area of research called "sensemaking." Klein, Moon, and Hoffman [43] distinguish between situation awareness and sensemaking as follows: "Situation awareness" is about the knowledge state that is achieved—either knowledge of current data elements, or inferences drawn from these data, or predictions that can be made using these inferences (Endsley, [24]). In contrast, sensemaking is about the process of achieving these kinds of outcomes, the strategies, and the barriers encountered (p. 71). Hence, sensemaking is viewed more as "a motivated, continuous effort to understand connections (which can be among people, places, and events) in order to anticipate their trajectories and act effectively" (Klein et al. [43], p. 71) rather than the state of knowledge underlying situation awareness. Although Endsley [26] points out that sensemaking is actually considering a subset of the processes used to maintain situation awareness, as noted above it is unclear how such longer-term processes such as understanding, self-awareness and self-aware models, and "sensemaking" fit into the current concepts of SA.

There has been an emphasis on SA on comparing the models of experts and novices, noting how the available data in a complex environment can overwhelm the novice's ability to efficiently process those data (Endsley, [25]) and how "experts" in contrast often have very efficient ways to notice and integrate a large amount of data. Interestingly, although this result is in line with the experience in early Artificial Intelligence with building "expert systems," the focus of many SA studies appeared to be on cues in the environment to activate these mental models rather than internal knowledge bases or rulesets that could become the basis for self-models [73].

In the future, it will be interesting for the field of self-awareness to pull from SA some very interesting research that they have been developing on how teams of situationally aware human and robotic agents best work together. Team SA is defined as "the degree to which every team member possesses the SA required for his or her responsibilities" (Endsley [26], p. 39). The success or failure of a team depends on the success or failure of each of its team members. If any one of the team members has poor SA, it can lead to a critical error in performance that can undermine the success of the entire team. By this definition, each team member needs to have a high level of SA on those factors that are relevant for his or her job. It is not sufficient for one member of the team to be aware of critical information if the team member who needs that information is not aware.

Shared situation awareness can be defined as "the degree to which team members possess the same SA on shared SA requirements" (Endsley and Jones [25], p. 47). As implied by this definition, there are information requirements that are relevant to multiple team members. A major part of teamwork involves the area where these SA requirements overlap—the shared SA requirements that exist as a function of the essential interdependency of the team members. In a poorly functioning team, two or more members may have different assessments on these shared SA requirements

and thus behave in an uncoordinated or even counter-productive fashion. Yet in a smoothly functioning team, each team member shares a common understanding of what is happening on those SA elements that are common.

2.12 Symbiotic Cognitive Computing

Symbiotic cognitive systems (SCS) [42] are multi-agent systems comprising both human and software agents that collectively perform cognitive tasks such as decision-making better than humans or software agents can by themselves. A driving principle of symbiotic cognitive systems is that humans and intelligent agents each have their respective cognitive strengths and weaknesses. The goal is not to surpass humans at challenging intellectual tasks such as chess or Jeopardy!, but rather to create agents that both support and rely upon humans in accomplishing cognitive tasks. This philosophy traces its lineage back to the vision espoused by Licklider in his essay on Man-Computer Symbiosis [49] and is today experiencing a revival among researchers in academia and industry who are pursuing aspects of the symbiotic cognitive systems research agenda from a variety of perspectives. One realm in which the principles and technologies of SCS are being applied is robotics, exemplified in the work of Rosenthal, Veloso and colleagues at Carnegie Mellon University on the Co-Bot [66, 68]. One also finds aspects of symbiotic cognitive computing in cognitive assistants such as Apple's Siri and IPsoft's Amelia (designed for help desks and related applications), and in the cognitive boardroom being developed by IBM Research [42], in which a multi-agent system interacts with humans via speech and gesture to provide seamless access to information and support for high-stakes decision-making.

One aspect of the challenge of creating SCS is that of developing algorithms (and the agents in which they are embodied) that are at least as competent as humans at the cognitive task for which they are designed. This task is made somewhat easier by focussing efforts on those aspects of cognition for which human biases, irrationality, and other deficiencies are well documented [3, 39, 78], and for which machines seem inherently better suited. A second general class of challenges for symbiotic systems is related to making human–agent interactions as seamless as possible. These include:

- Developing multi-modal forms of interaction that combine speech, gesture, touch, facial expression, and perhaps other manifestations of emotion [75];
- Learning mental models of other agents and humans, including their intent, to form a basis for adapting behavior so as to improve the speed and likelihood of accomplishing a task that the collective is trying to solve [68, 79]; and
- Storing, maintaining, and retrieving mental models of the environment, the task, and the other agents and human participants in the task to provide a shared context that can be used for communication among humans and agents [52, 67].

Kephart [42] discussed correspondences between autonomic computing systems and symbiotic computing systems, including the need for a means by which humans

can effectively communicate objectives to the system and the fact that the natural architecture for both is a multi-agent system, and hence, issues of inter-agent communication and interaction are very important. Moreover, self-management in all of its usual forms (self-optimization, self-healing, self-configuration, etc.) is essential for cognitive applications and the cognitive services from which they are built. A key difference is that, in SCS, humans are not just regarded as providers of high-level goals, but are expected to collaborate deeply with symbiotic cognitive systems, interacting with them constantly.

Given the strong overlap between autonomic computing systems and self-aware systems (detailed in Sect. 2.4), there is also a strong relationship between SCS and self-aware computing systems. A three-way comparison among AC, SCS, and self-aware systems is instructive. Like self-aware computing systems, but unlike autonomic computing systems, SCS do not require that agents take action. The reason that some software agents within an SCS may be self-aware without being autonomic is that they are not expected to perform all cognitive tasks by themselves, but instead to work collaboratively with humans. As a result, they may propose actions to humans, who can then use their judgment to decide whether or not to follow the agent's recommendations. Another connection between SCS systems and self-aware computing systems is that, while it is not a strict requirement, SCS are expected to learn models of intent and likely behavior by other participants (including both software agents and humans). In the case of SCS, there is a slight twist—the models may be used not just to manage resources wisely according to fixed goals, but the goal itself (the intent of the human users of the system) may not be revealed fully at the outset, so behavior models may be used to predict future goals and actions—thereby enabling the system to configure itself appropriately in anticipation of what it may be asked to do.

2.13 Auto-tuning

Auto-tuning covers techniques from high-performance computing (HPC), which automate the process of performance tuning for scientific applications (e.g., weather forecasts and genome expression analysis). Various approaches have been developed throughout the past decades [13, 54, 61, 64, 65, 76, 80, 83].

The motivation for auto-tuning in HPC is the problem that the frequency of new hardware increases, but the required time to manually tune high-performance code for this new hardware remains unchanged. Hence, approaches to automate the performance tuning for new hardware are needed.

The common way of performance tuning in HPC relies on source code transformations. Thus, the goal of auto-tuning approaches is to find those source code transformations, which improve performance. A basic prerequisite of most auto-tuning approaches is the existence of a kernel library. Such a library contains kernel (i.e., core) algorithms, which are used by scientific applications. Auto-tuning is applied to those kernel libraries instead of the applications themselves. This adheres to the stan-

dard principles in HPC, where manually optimized kernel libraries are commonly used. The application of auto-tuning enhances these libraries with code transformations, which adjust the libraries' algorithms to the given hardware architecture.

In general, there is a distinction between static and dynamic approaches, depending on when decision-making takes place. This is either at compilation time, denoting static auto-tuning, or at run-time, denoting dynamic auto-tuning.

Auto-tuning approaches are closely related to self-adaptive systems (SAS) in that they realize feedback loops. For example, the CADA loop [22] is realized in the following way: (1) Information about the available hardware is *collected*; (2) this information is *analyzed* with respect to its effect on the kernel algorithms; (3) a *decision* selecting code transformations improving (or optimizing) the performance of the kernel algorithms is made; and (4) the code transformations are applied (*act*).

Thus, auto-tuning can be seen as a special kind of SAS, which operates on source code level with a restricted focus on scientific applications (i.e., HPC). Notably, approaches of the SAS community usually realize the feedback loop on higher levels of abstraction. Commonly, the elements of variation are components, features, or classes, whereas auto-tuning works on source code statements. Auto-tuning approaches mainly apply techniques known from compiler optimization like loop unrolling to identify code variants that optimally utilize the underlying hardware (e.g., by not exceeding the number of available registers or available memory).

2.14 Constructive Definition

According to our definition, self-aware systems are complex systems that while in operation might need to access and analyze pieces of information about themselves and about the execution environment. In large part, this information is made available/created and managed during the various phases of development, e.g., design, architectural structure, code structure, execution machine structure, and deployment information. Thus, it is recognized nowadays that developing self-* systems requires some activities that traditionally occur at development time to be moved to run-time [2, 7, 36]. Activities here refer to the usual development process activities extended with execution monitoring activities. Responsibilities for these activities shift from developers to the system itself, the self-part, causing the traditional boundary between development time and run-time to blur. If a system needs to adapt in order to better respond to an increased and unexpected load of service requests, it might decide to change its configuration, e.g., by substituting one of its components by a more efficient one. In practice, this means being able to detect the situation by monitoring and analyzing the execution environment and its own behavior and also to carry on reconfiguration activities at run-time in a correct and time-efficient way.

The discriminating factor for deciding whether an activity has to be performed at development time or at run-time is cost. Cost can be explained in terms of resources needed to take responsibility of the activity and its achievement. Resources can be

software and hardware capabilities ultimately resulting in time or memory costs that need to be affordable with respect to the system goal and operational requirements.

Service-oriented and cloud computing paradigms permit reconsidering offline activities in a new perspective making it possible for a self-aware system to rely on heavy loaded system infrastructure for self-* system attributes, thus in practice mitigating the traditional cost-driven dichotomy between compile time and run-time.

This consideration leads us to consider also a constructive definition of self-aware computing systems that stresses the fact that the question is not only whether it is possible to make a system or portions of the system self-aware, but also whether it is economically reasonable/sustainable. This requires to focus on the amount of resources, software and hardware, that may be needed in order to support the self-awareness degrees of a system. The cost factor thus becomes the self-enabling factor that may influence design and architectural choices, and coding and execution choices as well as monitoring and analysis system capabilities, and may ultimately determine whether in the given conditions it is actually possible to develop, and how, a self-aware system. This also impacts the complexity of the techniques used to achieve self-awareness that may be more or less advanced depending on whether they are economically justified and sustainable.

So the extent to which a self-aware computing system is able to learn knowledge about itself and/or its environment and reason and adapt to internal and external changes is heavily dependent on development choices (design, architecture, programming languages, coding techniques), and deployment constraints (deployment infrastructure and resource availability). This requires quantitative reasoning capabilities at the process definition level as suggested in [2]. Depending on the system lifetime, these development choices may also be rediscussed; what could have been too costly at a certain stage of maturity of the system and of the technology may become convenient and affordable at a later stage of maturity. This suggests the architecture of the system to be flexible enough to easily accommodate evolutions of system with respect to its self-aware degrees.

2.15 Summary

The area of self-aware computing systems is still incipient, but promising concerning the construction of systems that are required to learn models on an ongoing basis and use them to reason about aspects related to the purpose for which the systems themselves were built.

Self-aware computing systems pose new opportunities and challenges for the research and engineering communities, some of which are related to prior experience in different disciplines.

This chapter has reviewed different concepts and research areas strongly related to self-aware computing. Different sections have explored topics such as AI, autonomic computing, self-organizing systems, or cognitive computing, among others, as well as their relation to self-aware computing systems and potential opportunities

for cross-pollination. Moreover, the landscape outlined in this chapter provided the basis for a constructive definition of self-aware computing system, as well as for some considerations concerning the different factors that influence the feasibility and the capabilities of a self-aware computing system. These considerations serve as a starting point to investigate important questions related to the conditions under which it is possible to actually develop a self-aware computing system, and in what way.

Acknowledgements The authors thank Lukas Esterle, Kurt Geihs, Philippe Lalanda, Peter Lewis, and Andrea Zisman for the useful feedback provided during the elaboration of this chapter.

References

1. *31st International Conference on Software Engineering, ICSE 2009*. IEEE, 2009.
2. Jesper Andersson, Luciano Baresi, Nelly Bencomo, Rogério de Lemos, Alessandra Gorla, Paola Inverardi, and Thomas Vogel. Software engineering processes for self-adaptive systems. In *Software Engineering for Self-Adaptive Systems II - International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*, pages 51–75, 2010.
3. Dan Ariely. *Predictably irrational the hidden forces that shape our decisions*. Harper Collins, New York, 2010.
4. W. Ross Ashby. Principles of the self-organizing system. In *Principles of SelfOrganization: Transactions of the University of Illinois Symposium*.
5. M. Autili, P. Inverardi, and M. Tivoli. Automated synthesis of service choreographies. *Software, IEEE*, 32(1):50–57, Jan 2015.
6. S. Banbury and S. Tremblay. *A cognitive approach to situation awareness: Theory and application*. Aldershot, UK: Ashgate Publishing, 2004.
7. Luciano Baresi and Carlo Ghezzi. The disappearing boundary between development-time and run-time. In *Proceedings of the Workshop on Future of Software Engineering Research, FoSER 2010, at the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010*, pages 17–22, 2010.
8. K.L. Bellman and C. Landauer. A web of reflection processes may help to de-conflict and integrate simultaneous self-optimization. In *SAOS 2014: The 2nd International Workshop on Self-optimisation in Organic and Autonomic Computing Systems*.
9. K.L. Bellman and C. Landauer. Towards an integration science. *Journal of Mathematical Analysis and Applications*, 249(1):3–31, 2000.
10. K.L. Bellman, C. Landauer, and P.R. Nelson. Developing mechanisms for determining "good enough" in sort systems. In *Second IEEE Workshop on Self-Organizing Real Time Systems, 2011*.
11. K.L. Bellman, C. Landauer, and P.R. Nelson. chapter System Engineering for Organic Computing: The Challenge of Shared Design and Control between OC Systems and their Human Engineers, pages 25–80. Understanding Complex Systems Series. Springer, 2008.
12. Nelly Bencomo, Amel Bennaceur, Paul Grace, Gordon S. Blair, and Valérie Issarny. The role of models@run.time in supporting on-the-fly interoperability. *Computing*, 95(3):167–190, 2013.
13. Jeff Bilmes, Krste Asanovicy, Chee-Whye Chinz, and Jim Demmel. Optimizing matrix multiply using PHiPAC: a portable, high-performance, ANSI C coding methodology. In *Proceedings of the 11th International Conference on Super Computing*, pages 340–347. ACM, 1997.
14. G. Blair, N. Bencomo, and R.B. France. Models@ run.time. *Computer*, 42(10):22–27, Oct 2009.
15. Rodney A. Brooks. *Cambrian Intelligence: The Early History of the New AI*. MIT Press, Cambridge, MA, USA, 1999.

16. Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. Engineering Self-Adaptive Systems through Feedback Loops. In *Software Engineering for Self-Adaptive Systems*, 2009.
17. Radu Calinescu, Lars Grunske, Marta Z. Kwiatkowska, Raffaela Mirandola, and Giordano Tamburrelli. Dynamic qos management and optimization in service-based systems. *IEEE Trans. Software Eng.*, 37(3):387–409, 2011.
18. Betty H.C. Cheng et al. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In *Software Engineering for Self-Adaptive Systems*. Springer, 2009.
19. Shang-Wen Cheng, VaheV. Poladian, David Garlan, and Bradley Schmerl. Improving architecture-based self-adaptation through resource prediction. In *Software Engineering for Self-Adaptive Systems*. Springer, 2009.
20. Michel Cotsaftis. *From System Complexity to Emergent Properties*, chapter What Makes a System Complex? - An Approach to Self Organization and Emergence, pages 49–99. Springer, 2009.
21. Rogério de Lemos et al. Software Engineering for Self-Adaptive Systems: A second Research Roadmap. In *Software Engineering for Self-Adaptive Systems II*. Springer, 2013.
22. Simon Dobson, Spyros Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt, and Franco Zambonelli. A survey of autonomic communications. *ACM Transactions on Autonomous and Adaptive Systems*, 1(2):223–259, 2006.
23. Vidulich M. Vogel E. Dominguez, C. and G. McMillan. *Situation awareness: Papers and annotated bibliography*. Armstrong Laboratory, Human System Center, ref. AL/CF-TR-1994-0085, 1994.
24. M.R. Endsley. Toward a theory of situation awareness in dynamic systems. *Human Factors*, 37(1):32–64, 1995.
25. M.R. Endsley. *The role of situation awareness in naturalistic decision making*. 1997.
26. M.R. Endsley. *Situation awareness: Progress and directions*. 2004.
27. Ilenia Epifani, Carlo Ghezzi, Raffaela Mirandola, and Giordano Tamburrelli. Model evolution by run-time parameter adaptation. In *31st International Conference on Software Engineering, ICSE 2009* [1], pages 111–121.
28. European Commission. Digital Agenda for Europe - Future Internet Research and Experimentation (FIRE) initiative, 2015.
29. M.L. Fracker. Measures of situation awareness: Review and future directions (report no. al-tr-1991-0128), 1991b. Wright-Patterson Air Force Base, OH: Armstrong Laboratories.
30. Robert B. France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *International Conference on Software Engineering, ISCE 2007, Workshop on the Future of Software Engineering, FOSE 2007, May 23-25, 2007, Minneapolis, MN, USA*, pages 37–54, 2007.
31. Cristina Gacek, Holger Giese, and Ethan Hadar. Friends or Foes? – A Conceptual Analysis of Self-Adaptation and IT Change Management. In *Proc. of the ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2008)*, pages 121–128. ACM, May 2008.
32. David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley R. Schmerl, and Peter Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *IEEE Computer*, 37(10):46–54, 2004.
33. Chang-chieh Hang and P. Parks. Comparative studies of model reference adaptive control systems. *IEEE Transactions on Automatic Control*, 18(5):419–428, October 1973.
34. Julia Hielscher, Raman Kazhamiakin, Andreas Metzger, and Marco Pistore. A framework for proactive self-adaptation of service-based applications based on online testing. In *Towards a Service-Based Internet*. Springer, 2008.
35. Paul Horn. Autonomic Computing: IBM's Perspective on the State of Information Technology. Technical report, 2001.
36. Paola Inverardi and Massimo Tivoli. The future of software: Adaptation and dependability. In *Software Engineering, International Summer Schools, ISSSE 2006-2008, Salerno, Italy, Revised Tutorial Lectures*, pages 1–31, 2008.

37. Rolf Isermann, Karl-Heinz Lachmann, and Drago Matko. *Adaptive Control Systems*. Prentice Hall International series in systems and control engineering. Prentice Hall, New York, 1992. ISBN 0-13-005414-3.
38. Valerie Issarny, Nikolaos Georgantas, Sara Hachem, Apostolos Zarras, Panos Vassiliadist, Marco Autili, MarcoAurlio Gerosa, and AmiraBen Hamida. Service-oriented middleware for the future internet: state of the art and research directions. *Journal of Internet Services and Applications*, 2(1):23–45, 2011.
39. Daniel Kahneman. *Thinking, fast and slow*. Farrar, Straus and Giroux, New York, 2011.
40. Gorazd Karer and Igor Skrjanc, 2012.
41. Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.
42. Jeffrey O. Kephart and Jonathan Lenchner. A symbiotic cognitive computing perspective on autonomic computing. In *2015 IEEE International Conference on Autonomic Computing*, pages 109–114. IEEE Computer Society, 2015.
43. Moon B Klein, G. and R.R. Hoffman. Making sense of sensemaking 1: Alternative perspectives. *IEEE Intelligent Systems*, 21(4):70–73, 2006.
44. Jeff Kramer and Jeff Magee. Self-Managed Systems: an Architectural Challenge. In *FOSE '07: 2007 Future of Software Engineering*, 2007.
45. C. Landauer. Infrastructure for studying infrastructure. In *ESOS 2013: Workshop on Embedded Self-Organizing Systems*.
46. C. Landauer and K.L. Bellman. Generic programming, partial evaluation, and a new programming paradigm.
47. C. Landauer and K.L. Bellman. Self-modeling systems.
48. Wenchao Li, Dorsa Sadigh, S.Shankar Sastry, and SanjitA. Seshia. Synthesis for human-in-the-loop control systems. In *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2014.
49. J. C. R. Licklider. Man-machine symbiosis. *IRE Transactions on Human Factors in Electronics*, HFE-1:4–11, March 1960.
50. Mieczyslaw Kokar M. and M. R. Endsley. Situation awareness and cognitive modeling. *IEEE Intelligent Systems*, 27(3):91–96, 2012.
51. P. Maes and D. Nardi (eds.). *Meta-Level Architectures and Reflection*. 1986.
52. Matthew Marge and Alexander I. Rudnicky. Towards evaluating recovery strategies for situated grounding problems in human-robot dialogue. In *IEEE International Symposium on Robot and Human Interactive Communication, IEEE RO-MAN 2013*, pages 340–341. IEEE, 2013.
53. Peter M. Mell and Timothy Grance. Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, United States, 2011.
54. A. Morajko, P. Caymes-Scutari, T. Margalef, and E. Luque. MATE: Monitoring, analysis and tuning environment for parallel/distributed applications. *Concurrency and Computation: Practice and Experience*, 19(11):1517–1531, 2007.
55. Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley R. Schmerl. Proactive self-adaptation under uncertainty: a probabilistic model checking approach. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015*, pages 1–12, 2015.
56. Brice Morin, Olivier Barais, Grégory Nain, and Jean-Marc Jézéquel. Taming dynamically adaptive systems using models and aspects. In *31st International Conference on Software Engineering, ICSE 2009*[1], pages 122–132.
57. Brice Morin, Franck Fleurey, Nelly Bencomo, Jean-Marc Jézéquel, Arnor Solberg, Vegard Dehlen, and Gordon S. Blair. An aspect-oriented and model-driven approach for managing dynamic variability. In *Model Driven Engineering Languages and Systems, 11th International Conference, MoDELS 2008*, pages 782–796, 2008.
58. Christian Müller-Schloer, Hartmut Schmeck, and Theo Ungerer, editors. *Organic Computing - A Paradigm Shift for Complex Systems*. Springer, 2011.
59. B.E. Ulicny M.M. Kokar and J.J. Moskal. *Ontological structures for higher levels of distributed fusion*. 2012.

60. Hyacinth S. Nwana. Software agents: an overview. *Knowledge Eng. Review*, 11(3):205–244, 1996.
61. Jakob Ostergaard. Discrete optimization of the sparse QR factorization. http://unthought.net/OptimQR/OptimQR/report.html, Oct 1998.
62. Mike P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing: State of the art and research challenges. *IEEE Computer*, 40(11), 2007.
63. Manish Parashar and Salim Hariri. Autonomic computing: An overview. In *Unconventional Programming Paradigms*. Springer, 2005.
64. Markus Püschel, José M. F. Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan Singer, Jianxin Xiong, Franz Franchetti, Aca Gacic, Yevgen Voronenko, Kang Chen, Robert W. Johnson, and Nicholas Rizzolo. SPIRAL: Code generation for DSP transforms. *Proceedings of the IEEE, special issue on "Program Generation, Optimization, and Adaptation"*, 93(2):232–275, 2005.
65. R. Ribler, J. Vetter, H. Simitci, Huseyin Simitci, and Daniel A. Reed. Autopilot: Adaptive control of distributed applications. In *Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing*, pages 172–179, 1998.
66. Stephanie Rosenthal, Joydeep Biswas, and Manuela M. Veloso. An effective personal mobile robot agent through symbiotic human-robot interaction. In Wiebe van der Hoek, Gal A. Kaminka, Yves Lespérance, Michael Luck, and Sandip Sen, editors, *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 915–922. IFAAMAS, 2010.
67. Stephanie Rosenthal, Sarjoun Skaff, Manuela M. Veloso, Dan Bohus, and Eric Horvitz. Execution memory for grounding and coordination. In Hideaki Kuzuoka, Vanessa Evers, Michita Imai, and Jodi Forlizzi, editors, *ACM/IEEE International Conference on Human-Robot Interaction, HRI 2013*, pages 213–214. IEEE/ACM, 2013.
68. Stephanie Rosenthal, Manuela M. Veloso, and Anind K. Dey. Task behavior and interaction planning for a mobile service robot that occasionally requires help. In *Automated Action Planning for Autonomous Mobile Robots, Papers from the 2011 AAAI Workshop*, volume WS-11-09 of *AAAI Workshops*. AAAI, 2011.
69. Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.
70. Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2):1–42, 2009.
71. Peter Sawyer, Nelly Bencomo, Jon Whittle, Emmanuel Letier, and Anthony Finkelstein. Requirements-aware systems: A research agenda for RE for self-adaptive systems. In *RE 2010, 18th IEEE International Requirements Engineering Conference*, pages 95–103, 2010.
72. Dale E. Seborg, Duncan A. Mellichamp, Thomas F. Edgar, and Francis J. Doyle, 2011.
73. MacMillan J. Entin E.E. Serfaty, D. and E.B. Entin. *The decision-making expertise of battle commanders*. 1997.
74. Mary Shaw. Beyond objects: A software design paradigm based on process control. *ACM SIGSOFT Software Engineering Notes*, 20(1):27–38, 1995.
75. Stefanie Tellex, Pratiksha Thaker, Joshua Mason Joseph, and Nicholas Roy. Learning perceptually grounded word meanings from unaligned parallel data. *Machine Learning*, 94(2):151–167, 2014.
76. A. Tiwari and J.K. Hollingsworth. Online adaptive code generation and tuning. In *Proceedings of 2011 International Symposium on Parallel Distributed Processing (IPDPS)*, pages 879–892, 2011.
77. Romina Torres, Nelly Bencomo, and Hernán Astudillo. Market-awareness in service-based systems. In *Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2012*, pages 169–174, 2012.
78. Amos Tversky and Daniel Kahneman. Judgment under Uncertainty: Heuristics and Biases. *Science*, 185(4157):1124–1131, September 1974.
79. Laura Pfeifer Vardoulakis, Lazlo Ring, Barbara Barry, Candace L. Sidner, and Timothy W. Bickmore. Designing relational agents as long term social companions for older adults. In

Intelligent Virtual Agents - 12th International Conference, IVA 2012, volume 7502 of *LNCS*, pages 289–302. Springer, 2012.
80. Michael J. Voss and Rudolf Eigemann. High-level adaptive program optimization with adapt. In *Proceedings of the eighth ACM SIGPLAN symposium on principles and practices of parallel programming*, PPoPP '01, pages 93–102. ACM, 2001.
81. Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, September 1991.
82. Kristopher Welsh, Pete Sawyer, and Nelly Bencomo. Towards requirements aware systems: Run-time resolution of design-time assumptions. In *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 560–563, 2011.
83. R. Clinton Whaley, Antoine Petitet, and Jack Dongarra. Automated empirical optimizations of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, 2001.
84. Tom Wolf and Tom Holvoet. *Engineering Self-Organising Systems: Methodologies and Applications*, chapter Emergence Versus Self-Organisation: Different Concepts but Promising When Combined, pages 1–15. Springer, 2005.
85. Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: theory and practice. *Knowledge Eng. Review*, 10(2):115–152, 1995.
86. Jun Xu, Pinyao Guo, Mingyi Zhao, Robert F. Erbacher, Minghui Zhu, and Peng Liu. Comparing different moving target defense techniques. In *Proceedings of the First ACM Workshop on Moving Target Defense*, MTD '14, pages 97–107. ACM, 2014.
87. Eric Yuan, Naeem Esfahani, and Sam Malek. Automated mining of software component interactions for self-adaptation. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS 2014, pages 27–36. ACM.

http://www.springer.com/978-3-319-47472-4

Self-Aware Computing Systems
Kounev, S.; Kephart, J.O.; Milenkoski, A.; Zhu, X. (Eds.)
2017, XVIII, 722 p. 165 illus., 92 illus. in color.,
Hardcover
ISBN: 978-3-319-47472-4