This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2017.2757143, IEEE Design and Test

Self-Awareness in Systems on Chip - A Survey

Axel Jantsch, Nikil Dutt, Amir M. Rahmani

Abstract

We describe the emerging paradigm of self-aware computing and give an overview of proposed architectures and applications with focus on SoC solutions.

1 Introduction

In addition to its roots in psychology the notion of self-awareness has been used in computing in a variety of different domains such as autonomic computing, organic computing, adaptive systems, and self-organizing systems, often with different, implicitly given definitions and objectives. Thus, a complete survey with all relevant work is impossible in the limited space of this article and a consistent treatment of this concept across all domains is a challenge. Consequently, our survey is incomplete in that it does not cover all interesting work. Rather it tries to

(a) explain the motivation of researchers and their interest in this topic,

(b) show its benefits,

(c) provide a paradigmatic reference frame, that relates to all key ingredients of self-awareness,

(d) give a cursory historical account, and a representational and fair exposition of the concepts of self-awareness in the various domains with systems on chip as the main focus.

First, we briefly introduce self-awareness and provide a reference definition of the term. Then, in section 1.2 we list benefits and motivate why researchers have so extensively studied and used the concept. In section 2 we introduce a paradigmatic architecture of a self-aware system (figure 2) which in our opinion is complete in the sense that it contains all crucial elements of self-awareness. Since in the literature the term self-awareness is often used to encompass only parts, frequently different parts, of the elements in our paradigmatic architecture, it serves as reference to which previous work can be easily related. In section 3 we discuss the main domains where self-awareness has been more or less extensively used, namely autonomic computing, self-adaptive systems, organic computing, and control theory. In a way, the first three sections 1, 2 and 3 can be considered as introduction to our main topic, self-awareness in SoCs. This lengthy introduction is necessary because of the diverse use of the concept in various domains, but we hope to provide a solid basis and understanding for the discussion of work on on-chip self-awareness and to appreciate the use and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2017.2757143, IEEE Design and Test

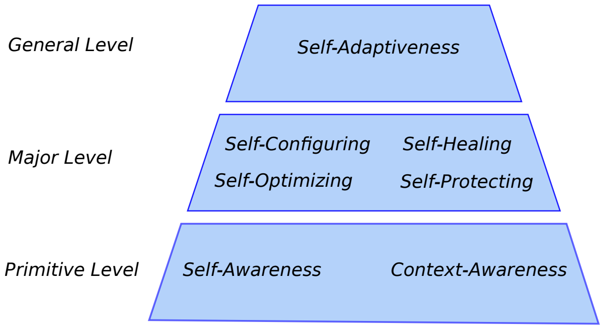

Figure 1: The hierarchy of self-* properties, first proposed in 2001 but cited here from Salehie and Tahvildari in 2009 [71].

utility of the involved concepts. Finally, in section 5 we list and briefly discuss the most important challenges of further research in this field.

1.1 What is self-awareness?

When engineers contemplate a concept they instinctively ask how it can be made useful and the desire to fully understand all the details and implications seem to be less important than to find a way to utilize it for a practical end. This has been the fate of self-awareness as a subject of study in the context of computing during the last 20 years. The pyramid of self-* properties (figure 1), originally proposed in the IBM initiative on autonomic computing in 2003 [48, 71], illustrates this point. Motivated by the objective to make software systems more flexible and truly self-adaptive researchers have identified self-configuration, self-healing and other self-* features as essential properties with self-awareness and context-awareness located at the primitive level.

Although adaptive systems with no self-awareness exist, it has been argued that a sophisticated self-model is a prerequisite for sensible adaptive behaviour when the environment and the system itself are sufficiently complex and there exists a causal relation between self-* properties and high quality in complex software systems [70].

As a consequence of this goal oriented approach in the study of self-awareness in computing systems, the research agenda has been dominated by a quest for utility. While this approach leads more directly to applicable results, it also makes a deeper understanding of the concepts appear less desirable. As an example we quote fully a definition of self-awareness offered by Kounev in 2011 [50] and in modified form also more recently in 2015 by Kounev et al. [51]:

> Self-awareness, in this context, is defined by the combination of three properties that IT systems and services should possess:
>
> 1. *Self-reflective*: i) aware of their software architecture, execution environment and the hardware infrastructure on which they are running, ii) aware of their operational goals in terms of QoS requirements, service-level agreements (SLAs) and cost- and

energy-efficiency targets, iii) aware of dynamic changes in the above during operation,

2. *Self-predictive*: able to predict the effect of dynamic changes (e.g., changing service workloads or QoS requirements) as well as predict the effect of possible adaptation actions (e.g., changing service deployment and/or resource allocations),
3. *Self-adaptive*: proactively adapting as the environment evolves in order to ensure that their QoS requirements and respective SLAs are continuously satisfied while at the same time operating costs and energy-efficiency are optimized.

It is an excellent definition, which we will use as reference in this survey. However, there are two interesting observations to note. First, self-awareness has moved up in prominence. It has been at the bottom of the self-* pyramid in 2001 (figure 1) as a supporting feature for more advanced adaptive behaviour. In the 2011 definition the term is used to encompass all relevant self-* properties including self-adaptiveness. In a way the pyramid has been turned upside down because the community has realized that self-awareness is not a simple collection of state variables that describe the state of the system (item (1) in above definition), but it has also to include the operational goals of the system and it has to properly reflect the effects of its own actions and of environmental changes on these state variables. Essentially, it has to include a complete, even though abstracted, model of the static and dynamic system properties. Once the dynamic properties are also properly represented, a prediction, perhaps by simulation, of the system and its interaction with the environment becomes possible and it is self-predictive according to item (2) in Kounev's definition. Based on this capacity of prediction it can proactively adapt its own actions earlier than it would otherwise be possible (item (3)). Hence, rather than being a simple elementary property to be used by a self-adapting system controller, self-awareness, once fully accomplished, makes proactive, adaptive behaviour almost straight forward.

The second point to note about the definition is, that it is purely utilitarian in that it does not try to capture the essence of the rich concept of self-awareness. It assumes that self-awareness as defined, i.e. self-reflective + self-predictive + self-adaptive, is useful for accomplishing the QoS requirements and the service-level agreements of the system, and thus, it should be implemented. It does not address the question, why it deserves to be a separate concept and what it adds to self-reflective, self-predictive and self-adaptive. Although we have no definite answer to these questions it may well worth to ponder them in order to identify additional aspects that are essential for self-awareness but not yet fully accounted for in the state of the art. There is some indication that learning, keeping track of history and dynamic goal management are such essential aspects. For instance, Chandra et al. [12] argue that a system has to acquire a substantial part of the self-model during operation by some kind of learning process to be considered self-aware. Externally built and implanted knowledge does not suffice. However, their argument is not of principal nature but claims that it will be very difficult to develop a sufficiently accurate self-model without a dynamic tuning and optimization process. While this may be debatable it illustrates the complexity of the concept even if the only concern is its usefulness in an engineering endeavor.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2017.2757143, IEEE Design and Test

As we discuss in section 3, autonomic computing is not the only branch of research that has struggled with the concept of self-awareness. Organic computing, bio-inspired computing and self-organization are other prominent lines of research that have approached the topic from different angles and contributed with specific insights and solutions. Before we discuss them in section 3, we summarize the potential benefits (section 1.2) and sketch a self-aware computing paradigm in section 2, that serves as a reference and reflects to some degree all major proposals for self-aware systems and architectures.

1.2 Benefits in Systems on Chip

Researchers of self-awareness generally argue that it allows a system to deal better with complexity. The complexity comes from the system itself (its structure and its state space), from the environment, and from the exceedingly diverse goals and objectives it has to meet.

Hardware state assessment and management: Self-awareness in hardware systems often facilitates the management of temperature [80], power/energy [29, 36, 80] and real-time performance [20, 36, 55]. Also, ageing effects are addressed with increasingly complex models to assess the progress of ageing and select counter measures [25].

Resource allocation: Given the often high number of processor, memory and interconnect resources available, resource allocation is a common target of self-aware enabled management schemes in both hardware and software. Task allocation and MPSoC configuration with higher energy efficiency based on cross-layer self-awareness on chip is proposed by Sarma et al. [78]. Because the approach extends across individual components (cores, routers) and layers (HW, NoC, OS middleware, application), local assessments have to be integrated into a comprehensive system level assessment. Based on a large number of sensors and comprehensive self-assessment of the system load balancing, task allocation, scheduling and migration for MPSoCs result in significant improvements [79,82]. In the SElf-awarE Computing (SEEC) framework dynamic adaptation through a smart interface between platform and application is achieved [36, 73]. The application registers its performance goals and the platform is responsible for meeting those goals by continuously monitoring the system's performance and appropriately adjusting the resource allocation.

An appropriate self-model allows to allocate scarce communication resources efficiently. Happe and Trammel-Keller use flexible protocol stacks [35] to allow for dynamic rearrangements and optimization of the communication protocols based on needs, requirements and constraints.

In general IT systems meeting Quality of service requirements and service-level agreements under energy-cost constraints is a tremendously complex task [50]. This challenge has driven the field of autonomic computing during the last two decades and with the growing size and complexity of the applications and the IT systems, the self-models and the self-awareness concepts have grown in complexity and sophistication as well [48, 51].

Reaction to changes in the environment: Many systems that interact with the physical environment by means of sensors and actuators have to adapt to changing conditions. Adaptive systems, as for example surveyed by Krupitzer and Becker [54] and further elaborated in section 3.2, have been studied in various applications. By providing a comprehensive assessment of the state of the system and its environment self-awareness offers a solid foundation for adaptation decisions and consequently can increase the quality of adaption. Indeed, we expect a direct dependence of the quality of adaption on the quality of self-assessment.

In summary we conclude that self-awareness leads to more sensible behaviour based on more detailed and often more explicit representation of the system's goals, its own state (available resources, faults, etc.), and the environment. Moreover, it leads to more efficiency due to better and adequate usage of resources. It can be used to detect aberrations of the system's behavior (faults, aging, malicious attacks, design errors, etc.), and of the environment. However, many of these potential benefits are only superficially studied and it remains to be seen what solutions can be found and how effective they are.

2 Self-Awareness Paradigm

Figure 2 shows a paradigmatic architecture of a self-aware system that allows to cover and relate most of the systems proposed in the literature. However, this is not the only or the best way to illustrate the structure of self-awareness. Being a loosely defined umbrella concept there are many options regarding what to include and what to exclude, what to highlight and what to deemphasize, what to make explicit and what to make implicit. Other researchers have made different choices, e.g. Lewis et al. [56], presumably due to different preferences in their research. For us attention, goal management and a central desirability scale are key elements not found in other architectures of self-awareness. Hence, we use figure 2 as a basis of discussion in this article but in the absence of either theoretical arguments or empirical evidence that clearly favors one architecture over another, we suggest to pragmatically use whatever is more suitable in a given context.

Kounev's definition cited above in section 1.1 [50] is represented in this figure as follows. The *self-reflective* part is located in the static self model (i), the goal management and goal hierarchy (ii), and the dynamic self model (iii). The *self-predictive* part is located in the dynamic self model, and the *self-adaptive* part is located in the decision making, the goal management and the goal hierarchy. More recently Kounev et al. [51] have revised this notion and have formulated the following definition.

> Self-aware computing systems are computing systems that:
>
> 1. *learn models* capturing *knowledge* about themselves and their environment (such as their structure, design, state, possible actions, and run-time behavior) on an ongoing basis and
> 2. *reason* using the models (for example predict, analyze, consider, plan) enabling them to act based on their knowledge and

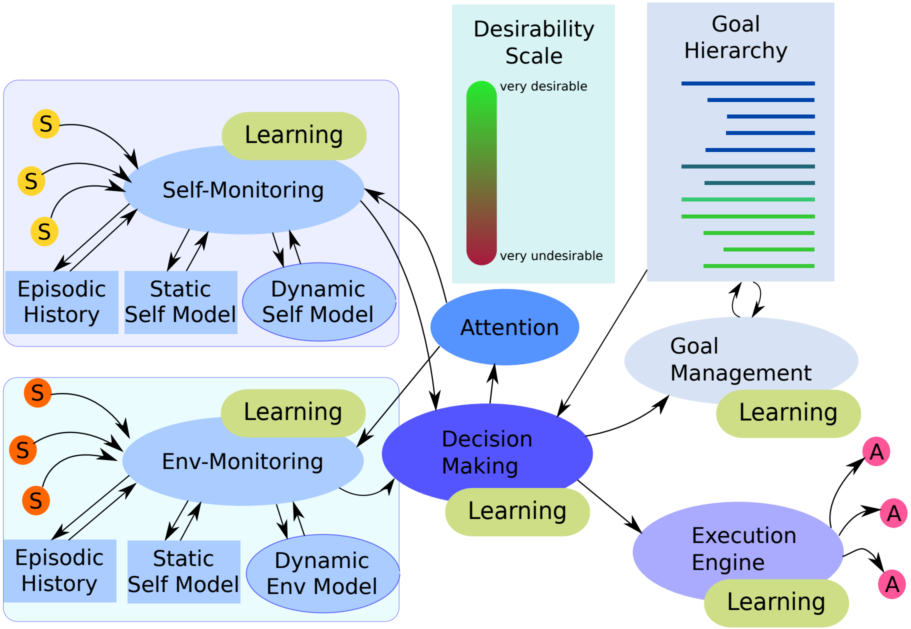

Figure 2: Paradigmatic architecture of a self-aware system. "S" nodes are sensors, "A" nodes are actuators. A proper assessment of the Self and the environment (Self-monitoring and Environmental-monitoring) are the basis for active goal management and effective decision making. The desirability scale is the currency of assessment. All assessments that are considered ”good” or ”bad” are mapped onto this scale. This allows the comparison of otherwise unrelated properties, e.g. the quality of a signal and the load-level of the battery. Machine learning algorithms are useful for all activities. Both monitoring functions can continuously improve to identify the normality and categorize aberrations. Goal management can learn to dynamically prioritize sub-goals in a way to optimize the accomplishments for high level goals. The decision making can optimize its algorithm based on the effect of its decisions on the system's performance. The execution engine can learn to generate control commands for the best possible effect. Note, that many connections are not drawn for the sake of clarity.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2017.2757143, IEEE Design and Test

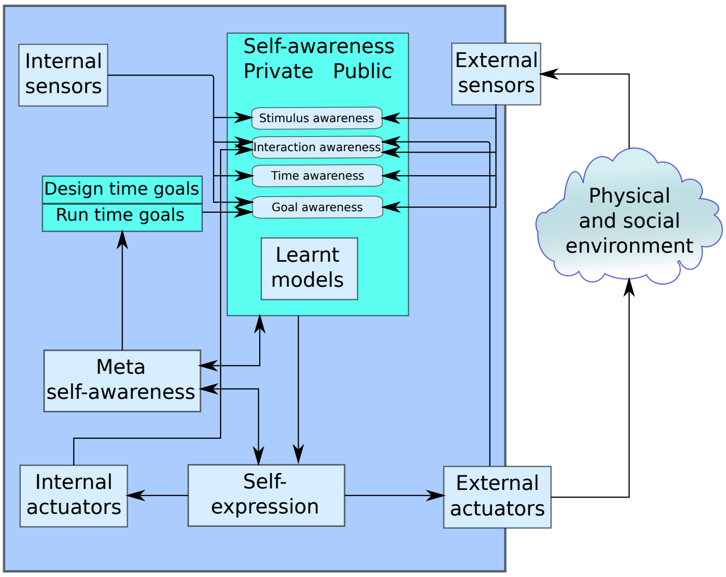

Figure 3: Reference architecture for self-aware computing systems proposed by Lewis et al. [56].

reasoning (for example explore, explain, report, suggest, self-adapt, or impact their environment)

in accordance with *higher-level goals*, which may also be subject to change.

Item 1 can be found in the green learning boxes of figure 2 and item 2 in the boxes dynamic self model, goal management and decision making.

A self-awareness reference architecture has been proposed by Lewis et al. [56] as shown in figure 3. All its important elements can be mapped to the paradigmatic architecture of figure 2 but a few points are worth noting.

Meta self-awareness refers to the ability to be aware of and reason about its own self-awareness. It allows to control and dynamically change the level of self-awareness and thus the resources expended on the self-awareness processes themselves. In some situations, or perhaps most of the time, it is unnecessary to keep these processes active because other tasks have higher priority. In figure 2 this is not made explicit but can be considered part of the goal management strategy and decision procedure. We have chosen to keep it implicit because meta self-awareness is a rather specialized feature and expected to be present in only few, high-end self-aware systems.

Similarly, *time awareness* is made explicit in figure 3 and refers to the capability to explicitly reason about the state changes of the system and its environment over time. In the paradigmatic sketch of figure 2 this is not explicit but the means for it are provided by keeping track of the history, by representing dynamic changes and by the goals and decision routines. In the same way, stimulus awareness, interaction awareness and goal awareness refer to abilities to reason about specific aspects of the system. They are not made explicit in figure 2, but may be included as part of specific goals, decision procedures and the dynamic internal models.

Figure 3 distinguishes between private and public self-awareness. Public aspects can be inspected from the outside like physical size, battery load level, and initiated actions. Private aspects are not directly visible outside and may refer to internal sensors, counters and registers. Both are part of the static self model in figure 2 but not distinguished.

On the other hand, figure 3 does not make explicit history mechanisms, the distinction between self model and environment models, attention, desirability, goal management and the various places where learning contributes to continuous tuning and optimization.

In figure 2 the processes of self-monitoring and environment-monitoring are fairly separated and only their results are only combined for decision making. In contrast, figure 3 treats both as one integrated process. We consider awareness to be the result of a hierarchical process where in the first level data from individual sensors are preprocessed and filtered individually after which more and more sensory information is gradually fused to establish increasingly abstract concepts. The integration of information from internal and external sensors occurs in most cases rather late in the hierarchy. Thus, it is justified to represent the processes responsible for self- and environment-monitoring as separate activities. However, they may exchange information at every step and for some systems a more integrated solution may be preferable as depicted in figure 3. In fact, complete isolation and complete integration of these two processes should be considered as extreme points in a continuous design space with practical solutions will almost always fall somewhere in between.

Dutt et al. [16, 17, 46] have listed relevant features in self-aware systems and have defined them as follows [16]:

- **Semantic Interpretation** includes an appropriate abstraction of the primary input data and a disambiguation of possible interpretations.
- **Desirability Scale** provides a uniform goodness-scale for the assessment of all observed properties.
- **Semantic Attribution** maps properties into the desirability scale suggesting how good or bad an observation is for the system.
- **History of a Property:** Awareness of a property implies awareness of its change over time.
- **Goals** provide the context in which interpretation and semantic attribution is meaningful.
- **The Purpose** of a smart embedded systems is to achieve all its goals.
- **Expectation on Environment:** The system expects a specific environment and detects if the environment deviates significantly from expectations.
- **Expectation on Subject:** Similarly, the system's own state and condition are continuously assessed to detect deviations, degradation, performance and malfunctions.

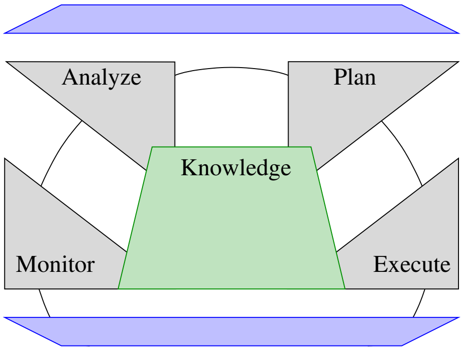

Figure 4: MAPE-K Loop illustrating a Monitor-Analyze-Plan-Execute cycle based on Knowledge [43].

- **Inspection Engine:** Continuously monitoring and assessing the situation requires a specific machinery that integrates all observations into a single, consistent world.

All these processes can be identified in figure 2. Semantic interpretation and attribution are not shown in the figure and are performed in the monitoring blocks and influenced by the goals and their priorities. A dynamically changing goal hierarchy will also modify the semantic attribution and attention. The inspection engine is not explicit in figure 2, but is part of the self-monitoring block with the help of several other blocks. An interesting point in this list of features is the emphasis on data abstraction and the semantic interpretation in the context of goals and an application. The importance of these processes have been elaborated by Taherinejad et al. [86] and they are part of the self and environmental monitoring tasks in figure 2.

3 Related Research Directions

3.1 Autonomic Computing

After the formulation of its vision in 2003 [48] the field has quickly grown and flourished. In a keynote at the International Conference on Autonomic Computing [47] Jeff Kephart counted overall 8000 papers published, 200 patents issued, and 200 conferences soliciting papers on the topic of autonomic computing. Since then the field has continued to be active but has diversified and overlapped with control, machine learning, cloud computing and web services. A main feature in much of the work on autonomic computing is a variation of the MAPE-K control loop [43], that illustrates the Monitor-Analyse-Plan-Execute cycle and is based on Knowledge which often means some kind of model. Interestingly, the generality of this model has not increased but in many approaches the models have been customized for a more specific purpose like resource management or maintaining a specific QoS level. An example of work against this trend is the approach of Sans et al. [74], which proposes a secondary control loop on top of the inner control loop resembling an explicit self-model. This outer loop is derived from the design time model but used during operation.

However, most of the work in the field has adopted less general and more specialized self-models. Even today, central themes are still self-adaptation, self-optimization, self-configuration and self-healing [24, 66], but in industrial practice its original vision has not fully materialized. There, trigger based approaches are still dominant [1,60], which means that triggering rules fire when a metric such as resource utilization or load imbalance exceeds a threshold value. In academia a number of systems with model based performance and resource management have been developed to assure Quality of Service levels, for instance DiVA [61], MADAM [22], MUSIC [33] and SASSY [59]. They typically use formalisms like Petri nets [49], queuing networks [59], stochastic process algebras [26], statistical regression [19], or kriging models [21] for performance modelling. However, from our perspective their self-models are limited because they all do not take the software architecture and the execution environment of the system into detailed account. A survey from Becker et al. [4] confirms this impression. Hence, these systems have limited self-awareness. On the other hand, approaches that do take the software system and execution environment into account are mostly used at design time and not part of the system during operation [52].

It seems that the more sophisticated aspects of the autonomic computing vision has had limited impact and practical solutions based on traditional performance models and heuristic rule based approaches have so far been sufficient to address the industry's need. This can on one hand be attributed to the conservative instinct of managers that prefer practically well proven and understood solutions and, on the other hand, to the availability of inexpensive computing, memory and communication resources that provide little incentives to find the most optimal or efficient solution. We have still limited understanding of the implications at the system level when advanced techniques from the machine learning, the control theory and optimization domains are integrated with complex models. This has also been concluded by Kounev et al. at a 2015 Dagstuhl Seminar [51]:

> Another finding was that much work remains to be done at the system level. In particular, while there has been considerable success in using machine learning and feedback control techniques to create adaptive autonomic elements, few authors have successfully built autonomic computing systems containing a variety of interacting adaptive elements. Several authors have observed that interactions among multiple machine learners or feedback loops can produce interesting unanticipated and sometimes destructive emergent behaviors; such phenomena are well known in the multi-agent systems realm as well, but insufficiently understood from a theoretical and practical perspective.

3.2 Self-Adaptive Systems

Work on self-adaptive systems naturally emphasize the task of adaptation and considers self-awareness properties only so far as they help to accomplish adaptation. It turns out that for more sophisticated adaptation models of the system and its environment become crucial, leading to *model-based* approaches [54]. Typically, three types of models are distinguished: system models to represent the system state, goal models to represent policies and rules, and environmental models to capture the context [42, 54]. Most work on model-based self-adaptation has been reported for general software systems. Based on the Software Engineering Institute's notion of Software Product Lines [85] a number of approaches model different system features as a basis for selecting dynamically the most appropriate configuration in a particular situation (e.g. [22, 23, 27, 33, 34, 62]; more examples are discussed in the surveys by Huebscher et al. [42] and by Krupitzer et al. [54]). However, it should be noted that a model of different software configurations does not constitute self-awareness. Self-awareness, as we understand the term in this survey, is based on a process of dynamically, if not continuously, acquiring data about the system itself and its environment to infer the current state and condition. Thus, most work on self-adaptive software systems do not cover self-awareness as defined in sections 1.1 and 2 above, even when using various models of the system extensively.

Work on self-adaptive resource constrained cyber-physical systems is more limited but comes closer to our notion of self-awareness. For the domain of smart cities and buildings Gürgen et al [32] propose a self-aware cyber-physical architecture that manages the data collection from sensors, the analysis, the planning and the adaptation of the controlled object (e.g. a building). Smart camera networks have to deal with quickly changing, diverse and complex environments. Esterle et al. [20] argue that fixed configurations are infeasible and the benefits of self-awareness are due to its coordinating effect on a distributed assessment and decision making, flexible rearrangements of the network under performance, cost and real-time constraints. In both examples important features of self-awareness are included but aspects like learning, goal management, attention and a central desirability scale are only rudimentary present or not at all.

3.3 Organic Computing

In the early 2000s, the increasing complexity of computing systems led people to conclude that unexpected emergent behavior is unavoidable once a certain complexity has been reached. Consequently, the design of desired and the control of undesired emergent behaviour were identified as main challenges. In 2005, Schmeck formulated the vision of Organic Computing as a response to the threatening view of being surrounded by interacting and self-organizing systems which may become unmanageable, showing undesired emergent behaviour [83]. In the following years the paradigm of Organic Computing was explored in a series of research projects, and in 2011 this work has been nicely summarized in the book Organic Computing, edited by Müller-Schloer, Schmeck and Ungerer [63]. Kramer et al. proposed a two level monitoring approach to self-awareness [53]. The Low-Level Monitoring is based on counting events such as cache misses, fault occurrences, or performance counters. In principle, any event that can be counted can be subject to this mechanism. The monitor can be programmed during operation to associate any type of event with event-IDs allowing for flexibility with respect to the kind of events under observation. High-Level Monitoring uses the event counts for state classification to reflect relevant information about the systems performance and state. Since event grouping and limited event abstraction is possible, the resulting system can be considered rudimentarily self-aware. In a similar spirit, Learning Classifier Systems (LCS) [41] and eXtended Classifier Systems (XCS) [11] have been used to assess a systems state as a base for decision making such as load management and task allocation [6]. The decisions are coded in rules. A rule consists of a condition, an action and a predicted reward value. If the condition of a rule matches, i.e. if the system is in the state described by the classifier, and the expected reward is sufficiently high, the action part of the rule is triggered. The rules are optimized by heuristics such as genetic algorithms and reinforcement learning. In the DodOrg project these and other ideas have been integrated to provide self-awareness in a many-core architecture, which in turn is used for power and thermal management [18].

In summary, the organic computing community has developed a number of innovative approaches to monitoring, adaptation, self-organization, distributed control and particularly contributed to a better understanding of phenomena of emergent behaviours such as emergent control [69]. However, similar to the autonomic computing endeavours, it has focused more on the decide and act parts of the observe-decide-act cycle. For instance, Kramer et al. have observed that in order to enable required self-organization capabilities, a monitoring infrastructure has to provide self-awareness [53], but have not well defined what is meant with self-awareness and have used a rather limited and static scope of the concept. Interesting aspects of awareness such as abstraction, attention, awareness of the historic changes in its own behaviour and in the environment have been hardly touched upon.

3.4 Control Theory

All the problems mentioned in section 1.2 have been successfully addressed without the explicit label of self-awareness. Numerous algorithms for scheduling and task allocation have been developed and deployed in real, demanding large scale systems and temperature and power managers are routinely built into each and every chip on the market. For instance, on-chip dynamic power management [5, 69] has been accomplished by control loops like more or less complex PID (Proportional, Differential, Integral) controllers where measured or estimated temperature, current flows, energy levels in batteries are used to tune voltage, frequency, and application load in order to meet given constraints and optimization objectives. Power management is a case in point how exceedingly complex internal models have been used as the problems become increasingly challenging and sophisticated over time. In simple processors of the 1980's and 1990's hardcoded and simple algorithmic solutions have dominated [5], while recent many-core SoCs operating at the edge of thermal stability require advanced power management based on detailed information reflecting the state and objectives of the hardware and the applications [69]. In many core heterogeneous SoCs with several applications concurrently sharing the platform the application behavior regarding the computational load, memory access and communication can vary over orders of magnitude in short time periods and are often highly unpredictable. P. Bogdan and colleagues have shown how accurate, statistical modeling of workload can significantly improve the efficacy power management [7, 8, 14].

There have been also efforts to hierarchically manage complex many-core systems by leveraging different structures of feedback control loops. For instance, in [64], a number of nested feedback control loops with different knobs and actuation epochs have been hierarchically deployed for power management with the objective of maximizing the performance while respecting the thermal design power budget. A centralized power management approach with the same objective is presented in [68] which considers both communication and computation characteristics of many-core systems in the power management policy. In a similar fashion, in [67], a coordinated power management approach with multiple scopes of actuation (virtual machine, cluster, server, core) is presented to cap the power consumption of the system and balance the utilization of the blades. Even though these approaches have proved to be effective to manage complexity, they focus on a single objective which is the main reason why they use several simple single-input single-output (SISO) PID controllers to form a larger manager for the respective problem at hand, which is often *maximizing performance under a power cap* [88].

There has been recently some contributions to leverage more advanced control theory approaches such as Linear Quadratic Gaussian (LQG) controller [84] to implement Multiple Input, Multiple Output (MIMO) formal control for maximizing resource efficiency. For instance, in [65], the authors utilize a MIMO controller to track throughput (billion of instructions committed per second) and power consumption for an out-of-order single-core processor in a coordinated manner. Even though MIMO controllers have the advantage of tracking different references with different priorities, they cannot efficiently be applied to complex systems as obtaining state-space models for complex systems is impractical, if not infeasible.

In general, tracking a single or multiple reference values, which is also called regulatory control, is the main application of control theory. As can be observed from the aforementioned examples, this property is essentially useful for problems where minimizing the tracking error of a parameter is the main goal. For instance, providing a certain quality of service (e.g., frame rate) for a real-time application, capping power consumption of a system, or controlling the thermal behavior of the chip are among popular use-cases for control theory. However, effects of actuations that cannot be modeled using difference functions (e.g, task migration) or problems that need optimization (e.g., minimize an objective function under constraints) cannot be properly addressed using classic control theory. From another perspective, thanks to the feedback-based structure, control theory based approaches are the best fit for problems such as disturbance rejection (e.g., disturbance due to workload variations when applying dynamic voltage and frequency scaling to control power) or handling noise/uncertainty in measurements (e.g., noisy sensors, virtual sensing), however it is ineffective to adapt or react to anomalies (e.g., faults), surprises, or radical changes in high-level goals.

In summary, control theory provides guarantees, has the ability to learn from feedback, and has the luxury of formal reasoning and methodology. However, restrictions such as the difficulty to obtain control theoretic models (e.g., transfer functions) in the form of difference equation and lack of a straightforward process to specify reference values limits its efficacy to be solely used for managing complex computing systems. On the other hand, while this approach works for a limited set of parameters and objectives, it does not scale well when the complexity of modeling the system dynamics increases. Modeling complexity escalates with the number of control inputs (i.e., knobs), measured outputs (i.e., sensors), and sub-systems (e.g., cores) in multi- and many-core systems. On

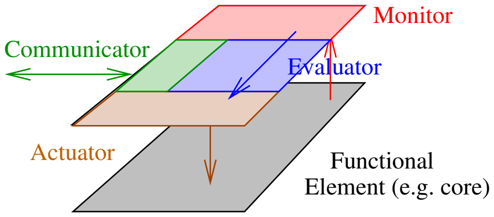

Figure 5: ASoC has two layers: A functional layer consisting of cores and the like, and an autonomic layer that controls the functional elements via a monitor-evaluator-actuator loop [10].

top of that, heterogeneity of sub-systems (e.g., in big.LITTLE style processors) makes the system modeling/identification even more complex. For instance, when in addition to temperature limits and battery life-time also aging effects, hard and soft real-time constraints, transient and permanent faults have to be considered and tuning knobs are available at circuit, architecture, operating system and application level, control loops become too complex to be used. It should be noted that the first step to design a control-theoretic approach is to have an accurate-enough model in hand.

Self-awareness offers the promise to be a scalable heuristic in that it can integrate any number of parameters and still provide workable solutions in real time and with sufficient quality. As in control theory, the set-points need to be specified by a higher entity, the integration of self-awareness with control theory can provide a layer of cognition for controllers to coordinate them towards the current goal of the system. So far this claim is still largely a promise but recent work and also the articles in this special issue show encouraging progress.

4 Self-Awareness on Chip

Features of self-awareness have found their way in many SoC resource management solutions. The vast majority follow a classic control loop approach, opportunistically extending and customizing them in ad-hoc ways as needed. In the following we discuss four examples that stand out in that they have self-awareness built into their architectures from the very start.

ASoC The Autonomic SoC platform (ASoC) [9, 10] is based on the organic computing paradigm and aimed at many-core architectures. Functional processing units, which are traditional cores, accelerators, memories, and other functional hardware units, are monitored and controlled by units in a parallel layer, called the *autonomic layer*. For each core or similar components in the functional layer there is a corresponding element in the autonomic layer, named the *autonomic element*, that consists of a monitor, an evaluator, an actuator and a communicator, as illustrated in figure 4. For instance, the autonomic element may monitor the load level in the functional element and update the frequency accordingly. The communicators allow the autonomic elements to communicate with each other. Since each functional element is shadowed by an autonomic element, we have a distributed control system.

The evaluators are rule based. Each rule consists of a matching pattern, an action and a reward value. The patterns match the monitored values and determine which rule can apply in a given situation. E.g. a pattern could encode "too high load". The action encodes the change of frequency and the reward value estimates how much the action will improve the situation. This reward value is then updated based on the actual improvement as observed by the monitor.

ASoC exhibits some of the features of our paradigmatic architecture in figure 2. There are self-monitoring, decision making and execution components. Learning is present in a limited form. The desirability scale and the goals are implicitly coded in the rules. Attention, environmental monitoring, goal management and the more sophisticated elements of self-monitoring, such as the assessment of the reliability of the measured data, are missing. However, it is conceivable to extend the ASoC architecture to include those elements of self-awareness as well.

SEEC Hoffmann and coworkers at MIT have developed SEEC [39], a general framework for self-aware computing using an observe-decide-act paradigm. The system cyclically monitors key features, applies a control and decision algorithm, and deploys appropriate actions to adapt to changes in the environment and its own state. It is based on the heartbeats API library [38], which defines a cyclic event called a heartbeat. Through API functions the application can register rate and latency performance goals in terms of the heartbeat period. Hence, the heartbeats API is a standardized means to monitor the performance of an application. The SEEC controller adapts and optimizes the system's behavior, for instance, by allocating and scheduling resources appropriately. The approach has been evaluated in several applications for performance optimization [39], power management [37, 40], and managing multiple objectives [36]. Also, the concept of *knobs* has been introduced [40] to expose steering facilities such as processor speed or power modes. SEEC allows to adopt different decision making strategies and algorithms, that have been studied extensively [58, 72].

Relating the self-awareness features of SEEC to the paradigmatic architecture of figure 2, we note that the monitoring-deciding-execution loop is thoroughly elaborated in SEEC while learning, history and attention mechanisms are not emphasized or not used at all. An interesting aspect of SEEC is that the goal formulation and management is assigned to the application. The SEEC platform provides knobs, control algorithms and measurements to the application, which in turn is responsible to formulate and adjust its goals. Similarly, the desirability scale, as it is related to and dependent on goals, is not part of the SEEC framework.

HAMSoC With HAMSoC (Hierarchical Agent Monitoring SoC) Guang et al. have proposed a four-level hierarchical control structure [28–30], as illustrated in figure 6. Each cell agent monitors and controls a core, an accelerator or another functional hardware block, which is similar to the functional elements and autonomic elements in ASoC. Cell agents have only local knowledge but are in turn monitored and steered by cluster agents which pursue optimizations for each respective local cluster. The platform agent is responsible for the entire SoC platform and can pursue platform-global optimizations. It interacts with

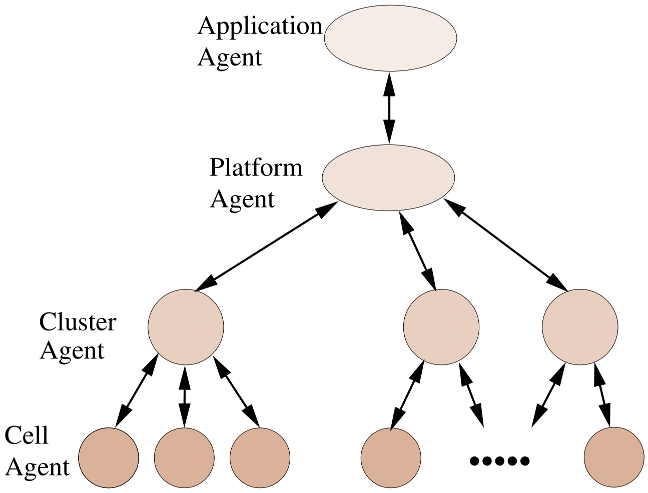

Figure 6: The four-level control structure in HAMSoC [29].

the application agent that provides application specific gals and requirements, based on the heartbeat concept of SEEC [38]. In contrast to ASoC, which has only one level of controllers communicating with each other, HAMSoC proposes a hierarchy of controllers that each is host different objectives from the local to the application level, as exemplified in a power and resource management scenario [44].

Even though the HAMSoC framework of hierarchical control has the potential to accommodate most of the self-awareness properties of our paradigmatic architecture of figure 2, it has not been fully elaborated and exploited. The HAMSoC controller hierarchy would be an appealing match to a hierarchical goal management system. However, neither goal management, nor attention, history or learning mechanisms have been explored.

CPSoC CyberPhysical Systems-on-Chip (CPSoC) [76] is self-aware embedded system paradigm that enhances traditional MPSoCs with a sensor-actuator-rich platform deploying a closed loop paradigm emulating large-scale Cyber-Physical Systems, enhanced with smartness through adaptivity and limited self-awareness [15]. CPSoC was developed primarily in the context of managing and exploiting hardware variability using the under-designed and opportunistic (UNO) computing paradigm [31].

The high-level system architecture of CPSoC is shown in Figure 7. The middle of this figure shows various levels of abstraction for the CPSoC platform, from the lowest (device/circuit level) layering up to the highest (application level). At each abstraction level, the CPSoC platform gathers information (through sensor fusion) using virtual and physical sensors, and in turn actuates (through actuator fusion) via virtual and physical actuators. The CPSoC architecture supports two classes of feedback loops: adaptive control (Red box in Figure 7) and self-aware supervisory management that generates supervisory policies (Tan box in Figure 7). These feedback loops are embedded within the adaptive, reflective middleware that orchestrates cross-layer sensing and actuation.

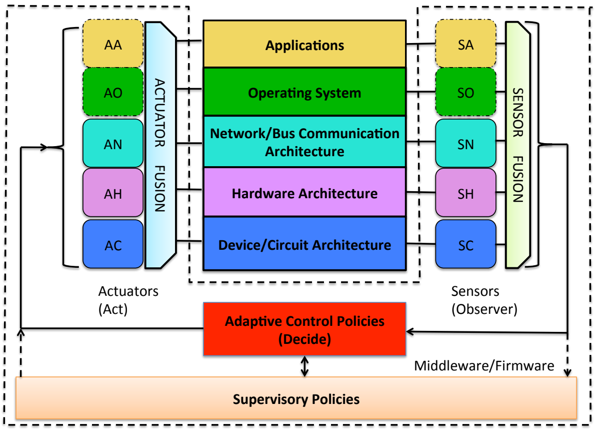

Figure 7: Cross-layer virtual sensing and actuation at different layers of CPSoC [77].

Figure 8 shows a more detailed view of the CPSoC architecture. On the top right of the figure is a template of an individual CPSoC computational Core, comprised of the computational units, memories, interfaces, and the on-chip sensing and actuations (OCSA) block that allows ubiquitous sensing and actuation at the CPSoC-Core level. These CPSoC Cores are tiled into a (homogeneous or heterogeneous) CPSoC computational fabric (lower right of Figure 8), using a Network-on-Chip (NoC) interconnect. Note that each router box in the NoC is also equipped with a sensing-and-actuation block (colored green) that enables monitoring and actuation at each NoC router. The left side of Figure 8 expands the abstraction layers of Figure 7, showing the CPSoC tiled hardware fabric at the lowest layer, and the applications executing on this platform at the highest layer. The adaptive, reflective middleware layer (yellow box on the left side of Figure 8) orchestrates the distributed sensing and actuation approach, where each component and core can make local decisions to manage the fabric.

The CPSoC architecture achieves self-awareness through three key ideas: **1) Cross-Layer Virtual and Physical Sensing & Actuation:** CPSoCs are sensor-actuator-rich MPSoCs that include several on-chip physical sensors (e.g. for aging, oxide breakdown, leakage, reliability, temperature) on the lower three layers as shown by the on-chip-sensing-and-actuation block (OCSN) and the Introspective Sensing Units (ISUs) in Figures 8. Virtual sensing and actuation [81] is accomplished across the abstraction stack. For instance virtual actuations such as application duty cycling, and checkpointing are software/hardware interventions that can predictively influence system design objectives. Virtual actuation can be combined with physical actuation mechanisms commonly adopted in modern chips [75]. **2) Simple and Self-Aware Adaptations:** Two key attributes of the self-aware CPSoC are adaptation of each layer and multiple cooperative Observe-Decide-Act (ODA) loops. As an example, the unification of an adaptive computing platform (with combined dynamic voltage and frequency scaling, adaptive body biasing, and other actuation means) along with a bandwidth adaptive NoC offers extra dimensions of control and solutions in comparison to traditional MPSoC architecture. **3) Predictive Models and On-line Learning:** Predictive modeling and on-line learning abilities enhance

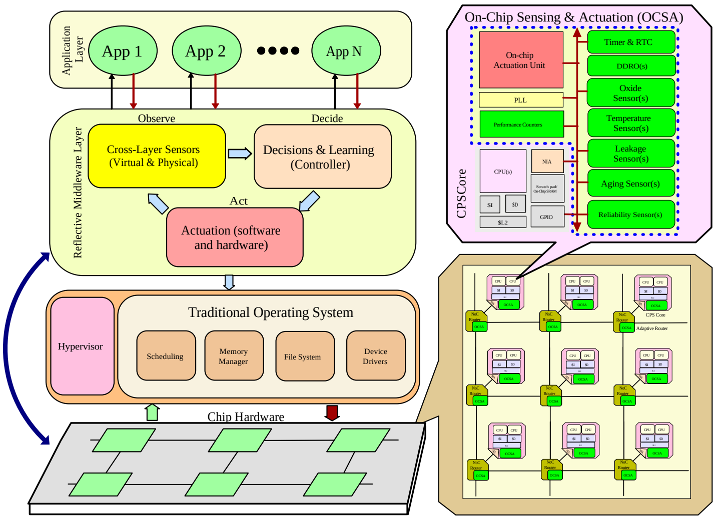

Figure 8: CPSoC architecture with adaptive Core, NoC, and the ODA Loop as Middleware [77].

self-modeling abilities in the CPSoC paradigm. The system behavior and states can be built using on-line or off-line linear or non-linear models in time or frequency domains [57]. CPSoC's predictive and learning abilities improve autonomy for managing system resources and assisting proactive resource utilization [76].

While CPSoC is a good initial exemplar for a self-aware SoC platform, it handles to a limited extent the self-awareness shown in the paradigmatic architecture of Figure 2. The monitoring-deciding-execution loop is intrinsically part of CPSoC, coupled with some limited learning and history mechanisms. Attention mechanisms have not been considered in CPSoC, and the goal hierarchy and goal management is in a very primitive form. The desirability scale is implicitly encoded within the goals, and has not been explicitly modeled within CPSoC. Furthermore, the self-awareness models in CPSoC did not consider malicious attacks, functional design errors and non-functional aberrations, and show a lot of room for growth in its self-awareness capabilities. Thanks to its modular cross-layer architecture, CPSoC has the potentials to cope with the discussed limitation by providing access to a rich set of cross-layer virtual and physical sensors and actuators, and the capacity to become self-aware in all respects.

5 Challenges

The term self-awareness encompasses a host of concepts and techniques that together offer great promises to tackle the design, maintenance and operation of complex, heterogeneous systems that are supposed to be adaptive, autonomous, highly efficient and always sensible. Even though a significant effort has already been spent in exploring this promise, the more intricate challenges still lie ahead. So far we have focused on picking low hanging fruits by incrementally extending existing architectures and methodologies. However, the research community will make faster progress when we do not exclusively focus on incremental development where each additional feature has to be thoroughly and quantitatively justified by the added value it gives. As this survey shows, self-awareness encompasses a host of concepts and techniques that together facilitate a comprehensive understanding of the system's state and its situation in the world. Picking out individual elements may only result in small gains or none at all. Thus we recommend to take a step back, try to comprehensively understand self-awareness, what it is, what it consists of, what it is good for, and, based on this understanding, realize it as a whole in cyber-physical computing systems. This approach would be inspired and informed by the widespread presence of self-awareness in animals given that survival of an expensive feature under relentless evolutionary pressure is a strong evidence for its benefit. A case in point is the maintenance of a history. It has hardly been studied in self-aware computing systems and, consequently, there is no strong experimental evidence for its benefits. We still believe it is indispensable for comprehensive self-assessment based on its importance in psychology [2, 3], and the intuitive argument that a comprehensive understanding of the current situation includes the sequence of events and states that historically led to the current situation. Moreover, historical data is required for future learning that involves a re-assessment of past situations. Thus, including it in research on self-aware computing systems is justified by the expectation that it will turn out to be beneficial.

Apart from considerations of research strategy, we identify five urgent technical challenges, that have to be addressed in order to fully honor the promise of self-awareness: learning, formulation of goals, scalability, ensuring correctness, and an appropriate design methodology.

Learning For truly self-aware systems, continuous, dynamic learning is indispensable. A major reason for the amazing feats of animals and plants is the relentless learning that goes on on all levels from the sub-cellular organelles to the individual and the community. As figure 2 indicates, learning is an integral part of many components and functions. Hence, it must be integrated in the sensor and monitoring nodes, in the attention mechanism, the decision making, the goal management, the execution and actuation, and in virtually every part of the system. Learning is only possible when feedback signals are available. Thus, the system must be pervaded by information flows providing feedback to all the learning elements. Many of our machine learning algorithms are not sufficiently efficient and optimized for the requirements of on-chip learning. Hence, we need both adapted machine learning algorithms and a system architecture that lends itself to continuous, pervasive learning and optimization.

Formulation of goals We need to be able to formulate quantitative goals for the design and the operation phases and we need to study the involved trade-offs. The traditional metrics of performance, power, energy, cost and fault tolerance are well understood. But quantitative metrics for adaptability, resilience, autonomy, self-assessment and situation assessment do not exist, are controversial or are limited in scope. However, we need to quantify these properties to explore the trade-off space spanned by the traditional and the self-awareness metrics. This has to be done for the design phase, but also and even more challenging, for the operation of the system since the system itself has to understand

and decide on these trade-offs in real-time. Research on goal formulation and management has been done in the context of artificial agents [13, 45, 87] which mainly focus on providing the capability to nominate top-level goals, and managing the nominated goals by prioritizing them. However, SoC's requirements and restrictions necessitate customized, light-weight, and minimally conflicting approaches which consider the priority, significance, objectives and requirements of each application, while holistically coupling the overlapping and/or contradicting objectives of different applications to satisfy the system constraints.

Scalable self-awareness Most applications do not require and cannot afford all features of a full blown self-aware system. To apply self-awareness to a wide range of systems, from resource constrained sensor nodes to multiprocessor platforms, designers should be able to easily select the level of capabilities. To this end a design space exploration method has to provide the means to trade-off functions and resources in a well defined self-awareness design space.

Ensuring correctness Validating a fixed, well defined functionality has been proven difficult enough due to the vast state spaces involved. Validating an adaptive system that, by definition, changes its behavior in ways unpredictable at design time seems to be hopeless. Still, if we cannot guarantee that certain bad behavior can never happen, the appeal of self-aware and autonomous systems will be limited to tiny application domains. Interestingly, self-awareness may be part of the solution because it can comprise a safety monitor that checks for and prohibits all unsafe and bad behavior. For this to work, the space of unsafe and bad behavior has to be specifiable in unambiguous and efficient terms leaving the system to freely explore the vast, unlimited space of safe and good behavior.

Design methodology Traditional design methodologies rely on the assumption that we can specify, validate and test the desired and acceptable behavior of the system. When we allow the adaptive, autonomous system to explore behavior that has not been specified at design time, this assumption breaks down. Hence, we have to consider alternative methodologies. For instance, the designers could use a general purpose, self-aware, autonomous machine, that in principle can meet a broad range of goals in any environment. Then the designers "fill" the system with a specific set of goals for a specific application and leave it tot he system to find ways to accomplish these goals. Although this vision seems remote, we will be forced to contemplate such options as the pain of designing more and more adaptive systems with traditional methodologies grows.

While these challenges seem formidable, researchers can draw from a range of disciplines with long history and large knowledge. Hence, given the state of the art, as summarized in this survey, we can certainly be confident that the development of fully self-aware SoCs is within the reach of the community in the coming years.

Acknowledgement

The authors wish to acknowledge the financial support by the Marie Curie Actions of the European Union's H2020 Programme, as well as the National Science Foundation under grant CCF-1704859.

References

[1] Amazon Web Services. Amazon auto scaling. http://aws.amazon.com/documentation/autoscaling. Accessed: 2017-02-09.

[2] B.J. Baars. *A Cognitive Theory of Consciousness*. Cambridge University Press, 1989.

[3] Alan D Baddeley and Graham Hitch. Working memory. In *The Psychology of Learning and Motivation*, volume 8, pages 48–89. Elsevier, 1974.

[4] M. Becker, M. Luckey, and S. Becker. Model-driven performance engineering of self-adaptive systems: A survey. In *ACM SIGSOFT Int. Conf. on Quality of Software Architectures*, pages 117–122, 2012.

[5] Luca Benini and Giovanni DeMicheli. *Dynamic Power Management: Design Techniques and CAD Tools*. Springer Verlag, 1998.

[6] Andreas Bernauer, Johannes Zeppenfeld, Oliver Bringmann, Andreas Herkersdorf, and Wolfgang Rosenstiel. Combining software and hardware LCS for lightweight on-chip learning. In Christian Müller-Schloer, Hartmut Schmeck, and Theo Ungerer, editors, *Organic Computing - A Paradigm Shift for Complex Systems*, Autonomic Systems, chapter 3.2, pages 253–266. Birkhäuser, 2011.

[7] Paul Bogdan. Mathematical modeling and control of multifractal workloads for data-center-on-a-chip optimization. In *Proceedings of the 9th International Symposium on Networks-on-Chip*, NOCS '15, pages 21:1–21:8, New York, NY, USA, 2015. ACM.

[8] Paul Bogdan, Radu Marculescu, and Siddharth Jain. Dynamic power management for multidomain system-on-chip platforms: An optimal control approach. *ACM Transactions on Design Automation of Electronic Systems*, 18(4):46:1–46:20, October 2013.

[9] A. Bouajila, J. Zeppenfeld, W. Stechele, A. Herkersdorf, A. Bernauer, O. Bringmann, and W. Rosenstiel. Organic computing at the system on chip level. In *2006 IFIP International Conference on Very Large Scale Integration*, pages 338–341, October 2006.

[10] Abdelmajid Bouajila, Johannes Zeppenfeld, Walter Stechele, Andreas Bernauer, Oliver Bringmann, Wolfgang Rosenstiel, and Andreas Herkersdorf. Autonomic system on chip platform. In Christian Müller-Schloer, Hartmut Schmeck, and Theo Ungerer, editors, *Organic Computing - A Paradigm Shift for Complex Systems*, Autonomic Systems, chapter 4.7, pages 413–425. Birkhäuser, 2011.

[11] M. Butz and S.W. Wilson. An alghorithm description of XCS. In P.L. Lanzi, W. Stolzmann, and S.W Wilson, editors, *IWLCS'00, Lecture Notes in Artifical Intelligence*, volume 2321, pages 253–272. Springer, 2001.

[12] A. Chandra, P. R. Lewis, K. Glette, and S. C. Stilkerich. Reference architecture for self-aware and self-expressive computing systems. In Peter R. Lewis, Marco Platzner, Bernhard Rinner, Jim Torresen, and Xin Yao, editors, *Self-Aware Computing Systems: An Engineering Approach*, chapter 4, pages 37–49. Springer, 2016.

[13] D. Choi. Reactive goal management in a cognitive architecture. *Cogn. Syst. Res.*, 12(3-4):293–308, 2011.

[14] R. David, P. Bogdan, and R. Marculescu. Dynamic power management for multicores: Case study using the intel scc. In *2012 IEEE/IFIP 20th International Conference on VLSI and System-on-Chip (VLSI-SoC)*, pages 147–152, Oct 2012.

[15] N. Dutt, A. Jantsch, and S. Sarma. Toward Smart Embedded Systems: A Self-aware System-on-Chip (SoC) Perspective. *ACM Trans. Embed. Comput. Syst.*, 15(2):22:1–22:27, 2016.

[16] Nikil Dutt, Axel Jantsch, and Santanu Sarma. Self-aware cyber-physical systems-on-chip. In *Proceedings of the International Conference for Computer Aided Design*, Austin, Texas, USA, November 2015. invited.

[17] Nikil Dutt, Axel Jantsch, and Santanu Sarma. Towards smart embedded systems: A self-aware system-on-chip perspective. *ACM Transactions on Embedded Computing Systems, Special Issue on Innovative Design Methods for Smart Embedded Systems*, 2016. invited.

[18] Thomas Ebi, David Kramer, Christian Schuck, Alexander von Renteln, Jürgen Becker, Uwe Brinkschulte, Jörg Henkel, and Wolfgang Karl. DodOrg - a self-adaptive organic many-core architecture. In Christian Müller-Schloer, Hartmut Schmeck, and Theo Ungerer, editors, *Organic Computing - A Paradigm Shift for Complex Systems*, Autonomic Systems, chapter 4.3, pages 353–368. Birkhäuser, 2011.

[19] E. Eskenazi, A. Fioukov, and D. Hammer. Performance prediction for component compositions. In *Proc. of the Intl. Symp. on Component-Based Software Engineering*, pages 208–293, 2004.

[20] L. Esterle, J. Simonjan, G. Nebehay, R. Pflugfelder, G. F. Domínguez, and B. Rinner. Self-aware object tracking in multi-camera networks. In Peter R. Lewis, Marco Platzner, Bernhard Rinner, Jim Torresen, and Xin Yao, editors, *Self-Aware Computing Systems: An Engineering Approach*, chapter 13, pages 261–277. Springer, 2016.

[21] A. Gambi, G. Toffetti, C. Pautasso, and M. Pezze. Kriging controllers for cloud applications. *IEEE Internet Computing*, 17(4):40–47, 2013.

[22] K. Geihs, P. Barone, F. Eliassen, J. Floch, R. Fricke, E. Gjorven, S. Hallsteinsen, G. Horn, M. U. Khan, A. Mamelli, G. A. Papadopoulos, N. Paspallis, R. Reichle, and E. Stav. A comprehensive solution for application-level adaptation. *Software Practice & Experience*, 39:385–422, 2009.

[23] Kurt Geihs, Roland Reichle, Michael Wagner, and Mohammad Ullah Khan. *Modeling of Context-Aware Self-Adaptive Applications in Ubiquitous and Service-Oriented Environments*, pages 146–163. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[24] Debanjan Ghosh, Raj Sharman, H. Raghav Rao, and Shambhu Upadhyaya. Self-healing systems - survey and synthesis. *Decis. Support Syst.*, 42(4):2164–2185, January 2007.

[25] Hans Giesen, Benjamin Gojman, Raphael Rubin, Ji Kim, and André DeHon. Continuous online self-monitoring introspection circuitry for timing repair by incremental partial-reconfiguration (cosmic trip). *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 00:111–118, 2016.

[26] S. Gilmore, V. Haenel, L. Kloul, and M. Maidl. Choreographing security and performance analysis for web services. *Formal Techniques for Computer Systems and Business Processes*, pages 200–214, 2005.

[27] Hassan Gomaa and Mohamed Hussein. *Dynamic Software Reconfiguration in Software Product Families*, pages 435–444. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[28] L. Guang, G. Plosila, J. Isoaho, and H. Tenhunen. HAMSoC: A monitoring-centric design approach for adaptive parallel computing. In Phan Cong-Vinh, editor, *Autonomic Networking-on-Chip: Bio-Inspired Specification, Development, and Verification*, chapter 6, pages 135–164. CRC Press, December 2011.

[29] Liang Guang, Ethiopia Nigussie, Pekka Rantala, Jouni Isoaho, and Hannu Tenhunen. Hierarchical agent monitoring design approach towards self-aware parallel systems-on-chip. *ACM Trans. Embed. Comput. Syst.*, 9(3):1–24, 2010.

[30] Liang Guang, Juha Plosila, Jouni Isoaho, and Hannu Tenhunen. Hierarchical agent monitored parallel on-chip system: A novel design paradigm and its formal specification. *International Journal of Embedded and Real-Time Communication Systems (IJERTCS)*, 1(2), 2010.

[31] P. Gupta et al. Underdesigned and opportunistic computing in presence of hardware variability. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 32(1):8–23, 2013.

[32] Levent Gurgen, Ozan Gunalp, Yazid Benazzouz, and Mathieu Gallissot. Self-aware cyber-physical systems and applications in smart buildings and cities. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, pages 1149–1154, San Jose, CA, USA, 2013. EDA Consortium.

[33] S. Hallsteinsen, K. Geihs, N. Paspallis, F. Eliassen, G. Horn, J. Lorenzo, A. Mamelli, and G. Papadopoulos. A development framework and methodology for self-adapting applications in ubiquitous computing environments. *Journal of Systems and Software*, 85:2840–2859, 2012.

[34] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid. Dynamic software product lines. *Computer*, 41(4):93–95, April 2008.

[35] Markus Happe and Ariane Trammel-Keller. Flexible protocol stacks. In Peter R. Lewis, Marco Platzner, Bernhard Rinner, Jim Torresen, and Xin Yao, editors, *Self-Aware Computing Systems: An Engineering Approach*, chapter 10, pages 193–214. Springer, 2016.

[36] H. Hoffmann. Coadapt: Predictable behavior for accuracy-aware applications running on power-aware systems. In *Real-Time Systems (ECRTS), 2014 26th Euromicro Conference on*, pages 223–232, July 2014.

[37] H. Hoffmann, M. Maggio, M.D. Santambrogio, A. Leva, and A. Agarwal. A generalized software framework for accurate and efficient management of performance goals. In *Embedded Software (EMSOFT), 2013 Proceedings of the International Conference on*, pages 1–10, Sept 2013.

[38] Henry Hoffmann, Jonathan Eastep, Marco D Santambrogio, Jason E Miller, and Anant Agarwal. Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments. In *Proceedings of the 7th international conference on Autonomic computing*, pages 79–88. ACM, 2010.

[39] Henry Hoffmann, Martina Maggio, Marco D Santambrogio, Alberto Leva, and Anant Agarwal. Seec: A framework for self-aware computing. Technical Report MIT-CSAIL-TR-2010-049, MIT, Cambrige, Massachusetts, October 2010.

[40] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. Dynamic knobs for responsive power-aware computing. *ACM SIGPLAN Notices*, 46(3):199–212, 2011.

[41] J. H. Holland. Adaptation. In R. Rosen and F. M. Snell, editors, *Progress in Theoretical Biology*, pages 263–293. Academic Press, New York, 1976.

[42] Markus C. Huebscher and Julie A. McCann. A survey of autonomic computing&mdash;degrees, models, and applications. *ACM Computing Surveys*, 40(3):7:1–7:28, August 2008.

[43] IBM Corporation. An architectural blueprint for autonomic computing, 2006. IBM White Paper.

[44] Syed M. A. H. Jafri, Liang Guang, Axel Jantsch, Kolin Paul, Ahmed Hemani, and Hannu Tenhunen. Self-adaptive NoC power management with dual-level agents: Architecture and implementation. In *Proceedings of the Conference on Self-adaptive Networked Embedded Systems*, Rome, Italy, February 2012.

[45] U. Jaidee, H. Muñoz Avila, and D. W. Aha. Integrated learning for goal-driven autonomy. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three*, pages 2450–2455, 2011.

[46] Axel Jantsch and Kalle Tammemäe. A framework of awareness for artificial subjects. In *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*, CODES '14, pages 20:1–20:3, New York, NY, USA, 2014. ACM.

[47] Jeff Kephart. Autonomic computing: The first decade. Keynote at the International Conference on Autonomic Computing (ICAC), 2011.

[48] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[49] S. Kounev. Performance modeling and evaluation of distributed component-based systems using queueing petri nets. *IEEE Transactions on Software Engineering (TSE)*, 32(7):486–502, 2006.

[50] Samuel Kounev. Engineering of self-aware IT systems and services: State-of-the-art and research challenges. In *Proc. of the 8th European Performance Engineering Workshop (EPEW11)*, volume 6977 of *LNCS*, pages 10–13. Springer, 2011.

[51] Samuel Kounev, Xiaoyun Zhu, Jeffrey O. Kephart, and Marta Kwiatkowska. Model-driven Algorithms and Architectures for Self-Aware Computing Systems (Dagstuhl Seminar 15041). *Dagstuhl Reports*, 5(1):164–196, 2015.

[52] H. Koziolek. Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, 67(8):434–458, 2010.

[53] David Kramer, Rainer Buchty, and Wolfgang Karl. Monitoring and self-awareness for heterogeneous, adaptive computing systems. In Christian Müller-Schloer, Hartmut Schmeck, and Theo Ungerer, editors, *Organic Computing - A Paradigm Shift for Complex Systems*, Autonomic Systems, chapter 2.3, pages 163–178. Birkhäuser, 2011.

[54] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 17:184 – 206, 2015. 10 years of Pervasive Computing' In Honor of Chatschik Bisdikian.

[55] M. Kurek, T. Becker, C. Guo, S. Denholm, A.-I. Funie, M. Salmon, T. Todman, and W. Luk. Self.aware hardware acceleration of financial applications on a heterogeneous cluster. In Peter R. Lewis, Marco Platzner, Bernhard Rinner, Jim Torresen, and Xin Yao, editors, *Self-Aware Computing Systems: An Engineering Approach*, chapter 12, pages 241–260. Springer, 2016.

[56] Peter R. Lewis, Arjun Chandra, Funmilade Faniyi, Kyrre Glette, Tao Chen, Rami Bahsoon, Jim Torresen, and Xin Yao. Architectural aspects of self-aware and self-expressive computing systems. *IEEE Computer*, August 2015.

[57] L. Ljung, editor. *System Identification (2Nd Ed.): Theory for the User*. Prentice Hall PTR, 1999.

[58] Martina Maggio, Henry Hoffmann, Marco D. Santambrogio, Anant Agarwal, and Alberto Leva. Decision making in autonomic computing systems: Comparison of approaches and techniques. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 201–204, New York, NY, USA, 2011. ACM.

[59] D. Menasce, H. Gomaa, s. Malek, and J. Sousa. Sassy: A framework for self-architecting service-oriented systems. *IEEE Software*, 28(6):78–85, November 2011.

[60] Microsoft Developer Network. Autoscaling and windows azure. http://msdn.microsoft.com/en-us/library/hh680945(v=pandp.50).aspx. Accessed: 2017-02-09.

[61] B. Morin, O. Barais, J. Jezequel, F. Fleurey, and A. Solberg. Models@run.time to support dynamic adaptation. In *Computer*, volume 42, pages 44–51, 2009.

[62] B. Morin, O. Barais, G. Nain, and J. M. Jezequel. Taming dynamically adaptive systems using models and aspects. In *2009 IEEE 31st International Conference on Software Engineering*, pages 122–132, May 2009.

[63] Christian Müller-Schloer, Hartmut Schmeck, and Theo Ungerer, editors. *Organic Computing A Paradigm Shift for Complex Systems*. Birkhauser, 2011.

[64] Thannirmalai Somu Muthukaruppan, Mihai Pricopi, Vanchinathan Venkataramani, Tulika Mitra, and Sanjay Vishin. Hierarchical Power Management for Asymmetric Multi-core in Dark Silicon Era. In *Proc. of the 50th Annual Design Automation Conference*, pages 174:1–174:9, 2013.

[65] Raghavendra Pradyumna Pothukuchi, Amin Ansari, Petros Voulgaris, and Josep Torrellas. Using Multiple Input, Multiple Output Formal Control to Maximize Resource Efficiency in Architectures. In *ISCA*, pages 658–670, 2016.

[66] Harald Psaier and Schahram Dustdar. A survey on self-healing systems: approaches and systems. *Computing*, 91(1):43–73, 2011.

[67] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. No "Power" Struggles: Coordinated Multi-level Power Management for the Data Center. In *ASPLOS*, pages 48–59, 2008.

[68] A. M. Rahmani, M. H. Haghbayan, A. Kanduri, A. Y. Weldezion, P. Liljeberg, J. Plosila, A. Jantsch, and H. Tenhunen. Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach. In *IEEE/ACM International Symposium on Low Power Electronics and Design*, pages 219–224, 2015.

[69] A.M. Rahmani, P. Liljeberg, A. Hemani, A. Jantsch, and H. Tenhunen. *The Dark Side of Silicon*. Springer, Switzerland, 1st edition edition, 2016.

[70] Mazeiar Salehie and Ladan Tahvildari. Autonomic computing: emerging trends and open problems. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 1–7. ACM, 2005.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2017.2757143, IEEE Design and Test

[71] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):14, 2009.

[72] Marco D Santambrogio, Henry Hoffmann, Jonathan Eastep, and Anant Agarwal. Enabling technologies for self-aware adaptive systems. In *Adaptive Hardware and Systems (AHS), 2010 NASA/ESA Conference on*, pages 149–156. IEEE, 2010.

[73] M.D. Santambrogio, H. Hoffmann, J. Eastep, and A. Agarwal. Enabling technologies for self-aware adaptive systems. In *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 149–156, 2010.

[74] Ricardo Sanz, Ignacio López, Manuel Rdoríguez, and Carlos Hernández. Principles for consciousness in integrated cognitive control. *Neural Networks*, 20(9), 11 2007.

[75] S. Sarma, N. Dutt, and P. Gupta. Strength of diversity: heterogeneous noisy sensor allocation and placement for optimal reconstruction and accurate fullchip thermal estimation. Technical report, Univeristy of California Irvine, 2014.

[76] S. Sarma, N. Dutt, P. Gupta, N. Venkatasubramanian, and A. Nicolau. Cyberphysical-system-on-chip (CPSoC): A Self-aware MPSoC Paradigm with Cross-layer Virtual Sensing and Actuation. In *Proc. of the Design, Automation & Test in Europe Conference*, pages 625–628, 2015.

[77] S. Sarma, N. Dutt, N. Venkatasubramanian, A. Nicolau, and P. Gupta. Cyber Physical-System-On-Chip (CPSoC): Sensor-Actuator Rich Self-Aware Computational Platform. Technical report, Univeristy of California Irvine, 2013.

[78] Santanu Sarma and Nikil Dutt. Cross-layer exploration of heterogeneous multicore processor configurations. In *VLSI Design Conference*, 2015.

[79] Santanu Sarma, Nikil Dutt, P. Gupta, A. Nicolau, and N. Venkatasubramanian. On-chip self-awareness using cyberphysical-systems-on-chip (CPSoC). In *Proceedings of the 12th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, New Delhi, India, October 2014.

[80] Santanu Sarma, Nikil Dutt, P. Gupta, A. Nicolau, and N. Venkatasubramanian. Cyberphysical-system-on-chip (CPSoC): A self-aware MPSoC paradigm with cross-layer virtual sensing and actuation. In *Proceedngs of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*, Grenoble, France, March 2015.

[81] Santanu Sarma, Nikil Dutt, and Nalini Venkatasubramanian. Cross-layer virtual observers for embedded multiprocessor system-on-chip (mpsoc). In *Proceedings of the 11th International Workshop on Adaptive and Reflective Middleware*, ARM '12, pages 4:1–4:7, New York, NY, USA, 2012. ACM.

[82] Santanu Sarma, T. Muck, L. A.D. Bathen, N. Dutt, and A. Nicolau. SmartBalance: A sensing-driven Linux load balancer for energy efficiency of heterogeneous MPSoCs. In *Proceedings of the Design Automation Conference*, July 2015.

[83] H. Schmeck. Organic computing - a new vision for distributed embedded systems. In *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pages 201–203, May 2005.

[84] Sigurd Skogestad and Ian Postlethwaite. *Multivariable Feedback Control: Analysis and Design*. John Wiley & Sons, 2005.

[85] Software Engineering Institute (Carnegie Mellon University). Software product lines. http://www.sei.cmu.edu/productlines/. Accessed: 2017-07-14.

[86] Nima TaheriNejad, Axel Jantsch, and David Pollreisz. Comprehensive observation and its role in self-awareness - an emotion recognition system example. In *Proceedings of the Federated Conference on Computer Science and Information Systems*, Gdansk, Poland, September 2016.

[87] M. Wilson, M. Molineaux, and D. W. Aha. Domain-independent heuristics for goal formulation. In *Proceedings of the Twenty-Sixth Artificial Intelligence Research Society Conference*, pages 160–165, 2013.

[88] H. Zhang and H. Hoffmann. Maximizing Performance Under a Power Cap: A Comparison of Hardware, Software, and Hybrid Techniques. In *Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 545–559, 2016.