Embodied Self-Aware Computing Systems

Henry Hoffmann, *Member, IEEE,* Axel Jantsch, *Member, IEEE,* and Nikil Dutt, *Fellow, IEEE*

***Abstract*—Embodied self-aware computing systems are embedded in a physical environment with a rich set of sensors and actuators to interact both with their environment and with their own embodiment. Through this interaction they learn about their situation, their own state and their performance. Although they are application specific like traditional embedded systems, they are significantly more flexible, robust and autonomous; they can adapt to a wide range of environmental variation and can cope with deterioration and shortcomings of their own performance. As such, embodied self-aware computing systems are an evolution of traditional embedded and cyber-physical systems into the direction of more autonomy, robustness and flexibility. Where traditional embedded systems operate in a changing world by demanding unchanging and fully characterized computing resources, embodied self-aware computing systems adapt to a changing world and changing computing resources.**

This article surveys the methods and methodologies used for embodied self-aware computing systems structured along the faculties of (A) sensory observation and abstraction, (B) self-aware assessment, and (C) hierarchical goals and control. The discussion is exemplified by application cases in the areas of systems on chip, control systems, health monitoring and condition monitoring in industrial production systems.

***Index Terms*—Embedded systems, Self-Aware computing, control, machine learning.**

I. Introduction

EMBODIED self-aware computing systems. As computing platforms and systems become more capable (e.g., with multiple heterogeneous processing units, increased storage, and sophisticated software stacks) and with their deployment across a wide range of application domains with competing goals and constraints (e.g., high performance, low energy, high reliability), there is a critical need to make these platforms adaptive, robust and responsive. This article focuses on self-aware embodiment for computing systems, drawing from lessons learned in cognitive robotics and other engineered autonomous systems.

The concept of embodied cognition has a long history in psychology and neuroscience, with the notion that any discussion of learning and intelligence must be placed in the context of an agent experiencing the real world via interactions with the environment through its sensorimotor system; with the resulting phenomenon of emergent intelligence [1]. Krichmar and Edelman's work on brain-based devices [2] postulated that embodied cognition – the idea that the brain is not functional without embedding into a physical body that experiences and interacts with the physical world – is critical for the development of intelligent robots. This pioneering work led to the active field of cognitive robotics where a robot with an embedded brain can learn, reason, and operate in a complex world in the face of complex goals. In this context, we identify an embodied computing system as an agent (i.e., the computational platform) embedded in, and interacting with both its physical environment, as well as its own embodiment via a rich set of sensors and actuators. The faculty of self-awareness empowers the computing system to become more flexible, robust and autonomous. We define the notions of computational embodiment and computational self-awareness through the following characteristics and their implications:

- **Physical Body**: The agent distinguishes between its own body and the environment; it has a model of its physical body; it keeps track of the boundary between its body and its environment.
- **Spatial and time relations**: The agent is aware of spatial and time relations between its body and its environment. The spatial relations are based on absolute (a coordinate system) and/or relative (right, left, above, below, ...) relations. The time relations are also based on absolute (some world time) and/or relative (earlier, later, ...) relations and imply real-time properties.
- **Situatedness**: This already implies that the agent is always in a situation as defined by the spatial and time relations of its body and the environmental bodies; the agent lives through and keeps track of a sequence of situations.

As illustrated in figure 1, self-awareness brings in

- **Model of behavior**: The agent has a model of its own behavior and of the environment.
- **Goals and control**: The agent's goals can be characterized by a combination of achieving desired objectives (e.g., maximize performance) while meeting some constraints (e.g., energy budget or thermal cap). Control strategies are applied on models of the agent to achieve objectives while meeting the constraints.
- **Assessment**: The agent can assess itself, i.e., compare its behavior with expected behavior; it can assess other agents in the environment and it can assess parts of its own behavior or body.
- **On-line learning**: The capacity for learning facilitates the continuous improvement and adaptation of the agent's own model of the physical body, the spatial and time relations, and its situatedness based on the sensory data it receives and a-priori assumptions. Learning can also build up the agent's own behavioral self-model. Furthermore, learning means that agents are individuals; every agent is different, specialized and customized to the sequence of situations it has experienced so far. Learning also means the agent can adapt to and optimize for a range of environments and tasks.

H. Hoffmann is with the University of Chicago.

A. Jantsch is with TU Wien and with the CD-Labor Embedded Machine Learning, Vienna, Austria

N. Dutt is with the University of California, Irvine.

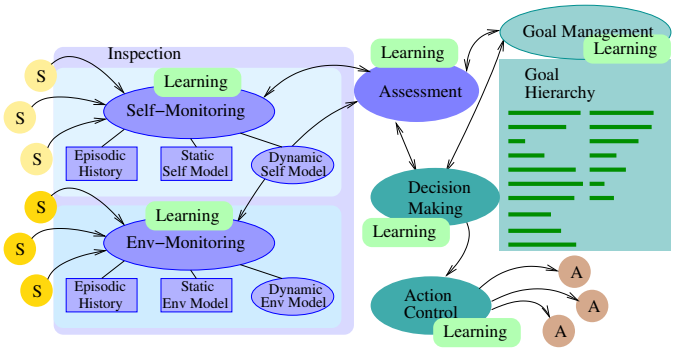


Fig. 1. A conceptual overview of an embodied self-aware computing system. The embodiment is facilitated with the sensors (S) and actuators (A) but also requires that the self-model and the self-assessment also includes and relates to the environment. Learning capabilities contribute to many, if not all, of the self-awareness features.

We note that contemporary engineered systems capture a wide range of self-awareness and embodiment properties, ranging from autonomous Cyber-Physical Systems (e.g., UAVs and robots) that capture higher levels of embodiment and self-awareness, to traditional computing platforms and IoT devices which lack many aspects of embodiment and self-awareness. This article focuses on the latter class of computing platforms that are the workhorses for embedded and general purpose computing. After outlining the basic characteristics of self-awareness and embodiment, we highlight both the deficiencies of many existing computing platforms and the potential benefits that these systems can gain from employing self-awareness and embodiment principles.

The rest of the article is structured as follows: Section II outlines how embodied computing systems interact with the external world/environment through sensory observation and abstraction. Section III describes typical control strategies deployed by embodied computing systems to achieve system goals. These control strategies are coupled with sensory observation and abstraction to effect actuations in the external world/environment. Section IV presents self-aware assessment that enables the embodied computing system to reflect on its own behavior and provide mechanisms for evaluating the quality and integrity of the decisions made by the controllers. Section V presents two use cases – Cyber-Physical Systems-on-Chip and personal health monitoring systems – that are analyzed to determine which facets of self-aware computational embodiment are deployed, and which are lacking. Section VI presents ongoing and future work on dynamic learning, adaptation and self-optimization that enables an embodied computing system to continually adapt to changes in the environment, as well as navigate the challenges of meeting multiple, possibly conflicting goals. Finally, Section VII concludes with a discussion of challenges faced in developing complete embodied, self-aware computing systems.

II. Sensory Observation and Abstraction

Embodied systems occupy a unique place in the world and continuously interact through sensors and actuators. While all systems come with a built-in/implicit model and more or less explicit assumptions about their bodies and behavior, information gained through data collected by the sensors allows for continuous improvements and adaptations of these (implicit or explicit)self-models. However, the raw sensory data has to be filtered, processed, abstracted and interpreted to reveal useful information that can be used to enhance the system models.

A. The Observation Circle

For a system to be fully self-aware, several aspects have to be considered [3], as illustrated by the observation circle in Figure 2.

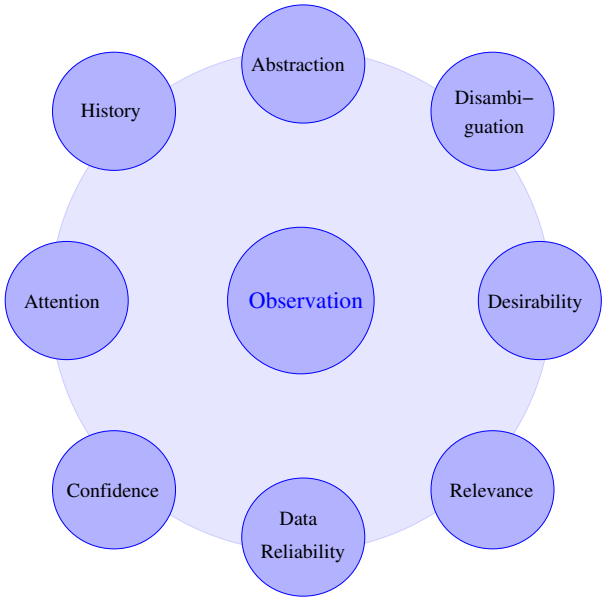


Fig. 2. The Observation Circle [3].

Abstraction: The data has to be abstracted to the "right" level for efficient processing and to be meaningful for the self-model. While data abstraction is commonplace in all systems equipped with sensors, *automated* abstraction is a key ingredient for self-aware and autonomous systems and poses many challenges as we elaborate below.

Disambiguation: Sensor data is always ambiguous and the best interpretation has to be identified using contextual information (e.g., Bayesian inference [4]) or via other sensors (e.g., through sensor fusion) [5].

Desirability: While the raw data by itself is neither good nor bad, when it acquires meaning through abstraction and in relation to the self-model of the system it is eventually labeled as more or less beneficial for or detrimental to a particular objective. This quantification in terms or a *goodness metric* allows for comparing otherwise unrelated properties and for formulating trade-offs and compromises. Indeed most realistic scenarios involve difficult decision making via unrelated metrics. For instance, in a healthcare IoT application, is it more important to have high precision in measuring a bio-signal or to save power and prolong operation time?

Data Reliability: Just as important as the raw data itself is the meta-data about its accuracy and precision. Accuracy is the deviation of the measured mean value from the ground truth and precision is the standard deviation of the measurements. Both are important to assess the utility of data and are critical in assessing the relative trustworthiness of contradicting data.

Relevance: Even if data is reliable it may not be relevant for a particular objective. For instance, a super accurate and precise temperature sensor, that is poorly attached to a person's skin, may be irrelevant to the assessment of the patient's health.

Confidence: While reliability and relevance are associated to specific data, a measure of confidence can be attributed to data, a sensor setup, or any processing step like abstraction or disambiguation. Based on prior knowledge or its own experience the system may place high or low confidence in its own sensors, information received from certain actors, results from specific computations, and conclusions derived by certain reasoning methods. It can be attached to any procedure and its results, e.g. it can and should be attached to self-models. Confidence is part of the meta data of measurement and computing procedures and subsumes the more specific metrics of reliability and relevance [6].

Attention is a means to allocate scarce resources. Since not all data needs to be captured continually, and not every computation needs to be performed, limited resources should be allocated to the tasks most urgently needed at a given time [7], [8].

History allows for extraction of statistics from observations, the assessment of behavior and performance of the system over time, its improvements or deterioration. Based on such historical data, the system is given the capability to predict its performance in the near future, the expected failure of sensors and actuators, and future trends in the environment. In turn, this understanding can motivate the system to evolve its goals, strategies, and tactics [3].

B. Automated Abstraction

System designers often have a good understanding of the application domain and are able to generate typical/useful abstractions of raw sensory data. For instance, an Electrocardiogram (ECG) monitoring device deploys a dedicated algorithm to extract the key feature of the human ECG. The image processing system in an autonomous car is trained extensively at design time to identify traffic signs, pedestrians and cars. In the Cyber-Physical System-on-Chip (CPSoC) [9] platform specific algorithms extract relevant performance metrics from bus load and buffer occupancy figures.

However, an ideal self-aware system should also be able to deal with novel regularities and patterns, identify anomalous patterns, and learn to predict previously unknown scenarios. For instance, a truly autonomous car should be able to learn to identify kangaroos, predict their behavior and devise strategies to best avoid collisions even if it has never been trained on kangaroos at design time. A key enabling feature for such abilities is automatic extraction of appropriate abstractions from raw data and signals. However, automated abstraction has rarely been researched or used in self-aware or autonomous systems, although abstraction techniques have been developed in different domains[1][10], in particular for time series signals [11] and images [12], [13], but general abstraction techniques that find and tune in to the abstraction most appropriate for a given context and purpose still constitute a major challenge.

In the context of HW and SW design automated methods for predicate abstraction [14], [15] and datapath abstraction [16] have been developed to facilitate efficient formal verification. These methods go a long way to conservatively ensure formal properties and are geared towards improving the efficiency of model checking and equivalence checking algorithms. In the two decades since their conception these abstraction techniques have been significantly improved, broadened and applied to many domains, e.g. to probabilistic models [17] and Markov decision processes [18]. Thus, this significant body of work provides inspiration and a wealth of methods for abstraction techniques, but to use and adapt them to the problem of efficiently extracting appropriate new abstractions for self- and environment models in an overwhelming stream of sensory data is an open challenge.

Information theory provides a means to quantify information and to guide abstraction [19], [20], [21]. For instance, Li and Ray [22] propose an abstraction method for signal time series to extract the regular patterns by maximizing the mutual information measures of the abstracted symbols and the finite state automaton that reflects the temporal patterns in the input signal. Thus, information theory can be a guide to effective abstraction techniques that reduce information and reveal temporal, spatial and causal patterns. Once these patterns are extracted and represented with the bare minimum of symbols, they represent an appropriate assessment of the system's situation and serve as basis for further decision making.

Like learning, abstraction is highly useful in many of the self-awareness features shown in figure 1. But just as learning methods, abstraction techniques have to be specifically adapted to the particular problem and context. They are obviously key for analyzing sensory data and for building and maintaining self- and environmental models, but they have also a role to play in assessment methods, goal and action modeling, which is still by and large uncharted territory.

C. The CCAM Case Study

We illustrate the notions of sensory observation and abstraction through a case study of a Confidence-based Context-Aware condition Monitoring (CCAM) system [6] that monitors industrial processes, exemplified by AC motors [23] and hydraulic circuits [24], to identify normal behavior, anomalies, drift conditions, and emerging faults without prior models of the System under Observation (SuO). We describe CCAM as an exemplar to demonstrate how some of the observation principles listed above can serve to infer accurate and robust

[1]Often the terms *symbolization* or, less frequently, *generalization* are used.

assessments of a situation without elaborate built-in models of the assessed system. Hence, it serves well to illustrate the concepts of observation, the state of the art of model-free monitoring and assessment, and its current limitations.

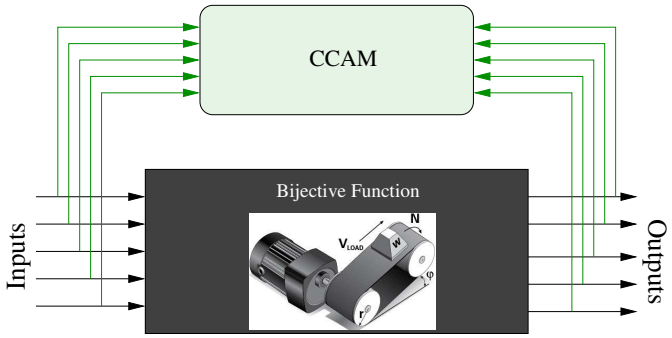


Fig. 3. CCAM monitors inputs and outputs of a System under Observation (SuO).

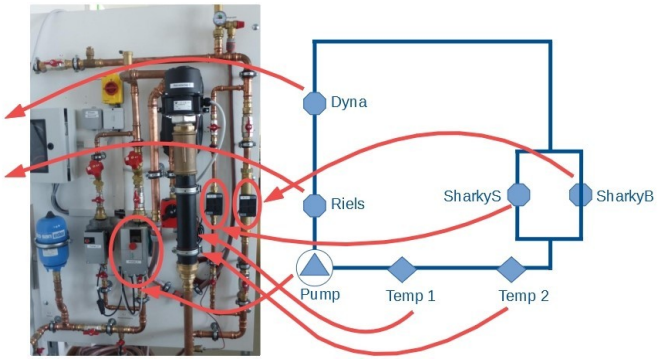


Fig. 4. An HVAC system monitored by CCAM.

As shown in Figure 3, CCAM monitors the inputs and outputs of the SuO and identifies stable patterns of input and output values, called *states*. Under the assumption of bijectivity (i.e., that changes on inputs always correlate to changes on outputs and vice versa), all isolated changes of inputs or outputs are flagged as errors. Thus, after a period of initial observation and learning, a set of operational states are identified and memorized. Unmatched changes of inputs and outputs are flagged as errors as well as drift situations, where an individual input or output signals slowly but systematically changes. Drifts are particularly interesting because in industrial processes they often signal the need for proactive replacement or cleaning of components in the near future.

Because CCAM is model-free and based on general principles, it is applicable to a surprising range of devices, without major configurations or changes. Götzinger et al. [6] report in a detailed study with an industrial AC motor and an HVAC system (Figure 4) that all normal states, wear-out conditions and break-downs that have been experimentally investigated, are correctly categorized by CCAM. A sensitivity analysis shows, that the CCAM operates robustly under a broad range of settings and configurations. CCAM's main limitation is that it cannot infer internal operating states of the SuO, and hence, it cannot be applied to systems with behavior that depends on its history in non-trivial ways. CCAM's generality and robustness is fairly surprising and in part due to the faculties of the observation circle:

a) Abstraction: Automatic abstraction is the core of CCAM since the CCAM states constitute abstractions of observed signals. It is guided by the following principles. A new sample of data, collected from all the monitored input and output signals, is assigned to a previously identified state if the values of the sample are "close enough" to the samples already assigned to the state. If the new sample does not match an existing state, because the value pattern in the sample is "different" from the patterns in existing states, a new state is established. The notion of similarity and difference is based on a relative distance metric between values governed by the rules of fuzzy logic [25], which means that the identified states reflect the clusters and patterns in the sampled data well.

b) Data Reliability: The reliability of individual data values and samples is implicitly assessed based on their consistency with previously sampled data. Individual values that fall outside the range previously found are usually ignored because it requires a minimum number of samples within proximity to be considered a valid state. These rules are also governed by fuzzy logic which avoids binary decisions and every new sample that falls within the proximity of an emerging new state gradually increases the probability that a new state is established. Thus, CCAM is robust against data with low precision because it adapts to the precision found in the data. If the precision of the sensors is low, the values clustered in one state will be dispersed, and precise sensors lead to tighter clusters in the states. CCAM is also robust against low accuracy and systematic bias in the sensed data, because it does not need or use absolute references.

However, data reliability is only checked based on consistency within the sampled data but not against context information.

c) Confidence: Götzinger et al. [6] define confidence as the inverse of the distance between a measured or computed value and the ground truth, with some appropriate distance metric. The closer the value is to the ground truth the higher the confidence in that value. Since the ground truth is usually not known, it has to be estimated. In CCAM confidence is used extensively to quantify how good a particular assessment or decision is. It is used to determine to which state a new data sample should be assigned, if a new state should be established, if a state signifies a malfunction, if drift has been detected, etc.

Again, confidence, i.e., the inverse distance to the ground truth, is estimated based on the collected data itself, but it does not use context information or a-priori models.

d) Other features of observation: The other features of the observation circle are used to a lesser degree. Although one could argue that disambiguation is performed and history is used, both are featured only implicitly and in rather trivial ways in CCAM. Desirability, relevance and attention are not at all used. However, all of them have the potential to improve and generalize CCAM. Context driven disambiguation and relevance assessment could certainly add value and widen the scope of CCAM while attention could direct scarce resources

to the analysis which is most beneficial and most urgently needed.

Thus, from the perspective of self-awareness, CCAM is an encouraging exemplar of the values of the faculties of abstraction, reliability and confidence, with its current limitations and future potential of other yet unexploited faculties.

III. Goals and Control

The design of any embodied computing system must achieve a set of desired goals, and devise strategies to accomplish these goals with system constraints. In this section we identify different categories of goals and constraints, outline strategies to meet constraints, and review classical and control-based strategies developed to achieve system goals while satisfying constraints.

A. Goals in Computing Systems

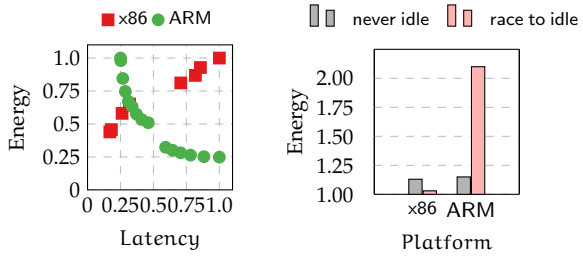
Computing systems have multiple—often competing—*goals*. A key-feature of self-aware systems is that their goals are first-class objects, meaning the self-aware computing system can reason about whether or not it is meeting its own goals [26]. If the system is not meeting its goals, it can change its behavior to find new ways to meet the goals. Therefore, an understanding of goals is essential to understanding the operation of self-aware computing systems.

To begin, computer system goals can be broadly separated into two classes: functional and non-functional requirements [27]. Functional goals are defined by results the system is supposed to produce. Non-functional requirements, in contrast, refer to the system's quantifiable behavior. Examples of non-functional requirements include latency constraints on input processing and battery life requirements for mobile systems. This example also highlights how non-functional goals can compete with each other; for instance, low-latency might require more computational power and negatively affect battery life.

Non-functional requirements can be further divided into *constraints* that must be satisfied for correct operation, while others represent *objectives* (or best-effort goals) that should be minimized or maximized. For example, in many embedded, real-time systems latency is a constraint that must be met, since returning a late answer is as catastrophic as returning the wrong answer (i.e., violating functional requirements). For some mobile users, energy is a constraint, as when the device runs out of energy, it can no longer deliver functional correctness, because it is not operating at all. These examples also demonstrate how goals can change as a system operates. Considering our mobile user again, they might consider latency (or responsiveness) to be a constraint when charge is high (or the device is plugged in) and energy savings would be an objective. When using navigation without an external power source, however, energy might be a constraint and latency should be best effort.

B. Common Ways to Meet Constraints

There are two very popular ways to handle constraints in computing systems. Embedded system designers often handle constraints by designing for *worst case* [28]. Worst case design is conservative and requires the system to be built so that constraints will never be violated under any circumstances. For example, to meet a latency constraint, worst case design requires reserving enough resources to ensure that there is sufficient computational power available under all circumstances. Often the drawback to this approach is that much of the operation does not elicit worst case behavior so the reserved computational power consumes more energy than necessary for most inputs.



(a) Energy vs. Latency (b) Energy Consumption

Fig. 5. Comparison of energy/latency tradeoffs in x86 mobile Haswell and ARM big.LITTLE. (a) normalizes energy and latency to the highest performance and highest energy available on each architecture. Peak performance is similar for both systems; peak power is about 8 Watts for both. (b) shows the energy compared to optimal for two common resource allocation heuristics, demonstrating that there is no one heuristic that works well for both architectures.

In contrast, general purpose computing systems are often designed for the common case. This approach tends to do well on optimizing objectives (especially compared to worst case design). Designing for the common case will violate constraints when workloads deviate sufficiently from expected behavior, however. In large scale systems, especially, the inability to meet constraints in uncommon conditions can cause cascading constraint violations [29].

Whether the system is designed for worst or common case, the problem with both is that they do not account for *dynamics*. That is, the operating environment may shift over time.

1) Example: Meeting Latency Goals with Minimal Energy: As noted above, mobile systems must meet the conflicting goals of consistent performance and low energy. For example, a video encoder must process frames at the same rate they are produced by the camera sensor, but users would prefer those frames are processed in a way that preserves battery life as much as possible. A common heuristic solution to this problem is to *race-to-idle*; i.e., when a new frame is produced, use all resources to encode that frame as quickly as possible and then transition into in a low-power idle state. This strategy has the benefit that it will always meet the latency requirement, but its energy efficiency is platform dependent and can be far from optimal on hardware systems with proportional energy management [30], [31], [32].

Figure 5a shows the latency and energy tradeoffs of a video encoder on both an x86 Mobile Haswell and an ARM big.LITTLE processor [33]. This figure shows that the x86 is more energy efficient at lower latencies—i.e., it reduces energy to run fast—while the ARM is more energy efficient

when latencies are high—i.e., it is more energy efficient to run slow. These different tradeoff spaces mean that different heuristics will achieve vastly different energy savings as shown in Figure 5b). The *race-to-idle* heuristic, which makes all resources available and then idles after completing a job, is near optimal on the x86 because reducing latency is energy efficient. On the ARM processor, over $2\times$ energy reduction is possible by using a *never-idle* heuristic that just meets the latency goal by intelligently using resources.

Race-to-idle is an extreme example of allocating for worst case: as long as the maximum resources in the system are sufficient to meet a worst case input's latency requirement, then the constraint will be met. The problem is that—for many applications—most inputs are not worst case and then those extra resources are wasted, meaning the system meets its constraint (latency) but is far from optimal in its objective (energy). The ability to adapt to the dynamic needs of particular inputs (frames in this example) would greatly enhance the computing system.

The issue with these common heuristic approaches is that they lack flexibility and continue to apply the same strategy regardless of the outcome. Such rigidity has long been noted as a deficiency, not just in computing systems, but in general decision making problems. Indeed, military strategist John Boyd identified the importance of constantly evaluating whether the current strategy is meeting goals by specifying the Observe-Orient-Decide-Act (OODA) loop [34], [35]. The OODA loop describes a continuous decision making process where an agent collects new data (observe), integrates and analyzes that data (orient), creates a new plan (decide), and then implements that plan (act). Agents that are developed with the OODA loop in mind have the flexibility to continually monitor whether or not their goals are being met, and adjust their strategy dynamically. Computer scientists have also noted the importance of this flexibility for dealing with the complexity of operating computing systems in dynamic environments [36], [37] and have developed a similar decision process known as Monitoring-Anaylsis-Planning-Execution (MAPE) [38]. The four stages of the MAPE loop directly correspond to the four stages of the OODA loop. Both OODA and MAPE describe frameworks for continuous decision making, but neither contain the kinds of policies or formalisms that could be directly instantiated in an embodied computing system.

C. Controlling Dynamic Systems

Fortunately, there is a whole field of study—control theory—dedicated to meeting constraints in dynamic systems [39]. Control theory is a broad family of formal techniques for making systems (originally physical systems, like industrial plants) behave as desired despite unpredictable dynamic disturbances. Control systems are particularly effective when respecting constraints is necessary for correct operation because properly designed controllers have formally analyzable behavior. While several seminal works showed that control theory could be effective in computers if applied by experts [40], [41] there has been a recent flurry of work on making control systems accessible to software engineers [42], [43], [44]. Several survey papers cover the broad applicability of control systems for self-adaptive software [45], [46], [47]. Here we briefly highlight techniques useful for meeting constraints in embodied self-aware computing systems.

1) Formulating Constraint Satisfaction as a Control Problem: While control theory encompasses a wide range of techniques several steps are common to control synthesis [46]. We summarize those steps here.

First—and most important—is to identify the constraints as defined above. These constraints must be quantifiable and measureable. While these requirements may seem obvious, they can pose a challenge. For example, many sensing systems have constraints on accuracy, but it can be difficult to measure accuracy online while the system is running because ground truth data is not available (section II).

Once quantifiable and dynamically measurable constraints are determined, the next step is to identify the system's tunable parameters. Suitable parameters should affect the quantifiable measures identified earlier. Furthermore, these parameters must be dynamically tunable. For example, some iterative software has convergence thresholds. Such a parameter might be dynamically tuned to manage latency or accuracy constraints. Computing systems have many tunable parameters, but they are not always dynamically adjustable. Some features, like enabling error correcting codes, may require a system reboot, which limits its effectiveness for dynamically controlling the system.

Having identified the constraints and the parameters, the next step is to model the relationship between the two. This modeling process is, perhaps, the biggest impediment to widespread adoption of control theory in computing systems. Many software engineers are simply not trained to construct suitable analytical models. Indeed, there are a wide range of possible modeling techniques that could be applied and their suitability will vary greatly based on the properties of the constraints and tunable parameters. In applications of classical control theory, a dynamic model will be constructed in either state space form or as an input-output relationship. Filieri et al. provide a broad survey of the many possible modeling techniques [46], while Hellerstein et al. provide great detail on state space and input-output models of computing systems [40]. Shevtsov et al. provide a recent survey on uses of these techniques within computer systems [47]. A significant choice in the modeling phase is how to account for disturbances, or deviations from expected behavior. If the conditions that cause these disturbances, and their effects on the constraints, can be modeled, it is possible to design a controller that measures the conditions that would cause the disturbance and proactively cancels it. Otherwise, the model should be tolerant to some maximum deviation from expected behavior.

Once a model—expressed as a set of equations—has been established, the controller can be designed. As with modeling, a wide variety of choices are available for control design. Control design will be largely influenced by the behaviors the designer wants to induce in the self-aware system. Clearly, the constraints should be respected. Other desirable behaviors include properties such as:

- Transient behavior, or how the system reaches the constraint when initialized or disturbed.
- Robustness to error, or the non-ideal conditions under which the system will still meet the constraints. The possible errors could include both modeling and measurement errors.

While several other properties can be enforced, the key is that the design phase not only produces a controller that meets constraints, but also provides other desirable behaviors.

After design the controller is implemented. This process involves translating the equations that describe the controller into executable code. Several practical concerns must be accounted for in this phase to go from the idealized model to a real computing system. Among these will be how to handle delayed, erroneous, or missing measurements; what happens when tunable parameters reach their limits but the constraints are still not met; and accounting for any delays between changing a parameter setting and observing the change on the measured behavior.

The final step is, of course, to test the implemented controller and verify that it can meet the desired constraints in a representative use case. One of the biggest difficulties is that control design is most useful for systems that operate in unpredictable dynamic environments. Testing the ability to handle unpredictable scenarios is challenging because any test scenario is, by definition, anticipated. Thus, the testing procedure will generally involve putting the controlled system through inputs that test the extreme range of its error tolerance (as determined in the modeling and control design phase). In this way, implementers can have some confidence that the system will behave as expected conditional on the determination of maximum expected error.

2) Common Control Methods: We briefly discuss common patterns of control design. While a full survey is beyond the scope of this paper (see [40] for a broad and detailed survey), we highlight some of the pros and cons of different approaches as they relate to self-aware computing systems.

We consider *feedback* control designs. These are common control approaches where the control input is the difference between the constraint and the measured behavior. Feedback designs are the most robust as they can handle uncertainty, reject disturbances, and even stabilize systems that would otherwise behave erratically. In contrast, open-loop and feedforward control require perfect knowledge of the operating environment and model error to guarantee the constraints are met. Our assumption is that self-aware systems must be able to tolerate error and uncertainty, so feedback is required.

There are three common classes of feedback control design. These include the Proportional-Intergal-Derivative (PID) controller, Optimal Control, and Adaptive Control. The PID controller is a classical control system, which first determines the error between the goal and the current behavior, and then determines its next action as a weighted combination of terms proportional to the error, the integral of the error, and the derivative of the error. A classic example in computing systems is contolling system power consumption [48], [49]. Optimal control not only satisfies a constraint, but simultaneously optimizes objectives and is particularly useful when controlling multiple tunable parameters as there might be many possible ways to meet a constraint, but only one that is optimal for the objective. Examples of optimal control in computing systems include resource managers that optimize the use of multiple resources [50], [51]. A powerful, and common, subclass of optimal control is the model predictive controller (MPC), which works to optimally satisfy constraints by not just considering the current system state, but also on predicted future states. MPC has been used to provide QoS in webservers [52] and been shown to be a generally powerful technique for building self-adaptive software systems [53]. Filieri et al. show how to automatically synthesize MPC for computing systems [54].

Adaptive control is a family of techniques for dynamically altering a controller's behavior. Adaptive control is typically used to augment an existing control solution, by taking a time-invariant design and turning it into a time-variant one. For example, Sun et al. show how to use time-varying models to capture the dynamic, non-linear behavior of web servers [55], while Hellerstein et al. show how to use a control design to adapt the .NET thread pool to varying workloads [56]. Parametric adaptation retains all the control equations from the time-invariant design, but allows some coefficients to change as the controller operates. Other forms of adaptation vary the form of the controller. For example, switching systems change from one set of control equations to another based on the operating conditions. For example, a non-linear system could be controlled by constructing a series of linear controllers corresponding to different parts of the system's operating range. As behavior progresses through the range, the controller switches to the model appropriate for the current behavior. There are a wide range of different techniques for making a controller adaptive.

Unfortunately, a terminology issue arises when applying adaptive control techniques to computer systems because the computing and control communities use the term "adaptation" in closely related, but distinct ways. For most computer engineers, adding a controller (even the simple PID control mentioned above) would make a system adaptive because the controller enables the computer to respond to conditions that it previously ignored. For example, without a controller a computer designer might have to allocate resources to guarantee a latency goal is never violated, but adding a simple PID controller would allow the system to adapt resource usage (the tunable parameter) to observed latency and save energy.

A control designer would not consider such a system adaptive, however. This difference is due to the fact that the control equations are fixed in this example. If the controller accounted for workload (for example estimating whether the current workload was compute or memory bound and then changing control equations accordingly) it would then be considered an adaptive controller.

D. Advantages and Disadvantages of Control

The primary advantage of control techniques is that the modeling and design process results in a series of equations that emit formal guarantees—and perhaps more importantly,

the assumptions required for them to hold—about the system's ability to respond to dynamic behavior. For example, given the system and control model, it is possible to reason about the maximum amount of model and measurement error that could be encountered and still permit the system to converge.

These formalisms provide a very important capability for designers of self-aware computing systems. The goal of self-aware systems is to maintain operating goals (constraints and objectives) despite unforeseen circumstances. As mentioned above, this is a formidable task as it is—by definition—quite difficult to demonstrate a system reacts to the unforeseen as any demonstration would require foresight. Control formalisms, do not require enumeration of all circumstances, but permit designers to reason about error tolerance. For example, designing self-aware systems with control allows designers to make a system that can handle errors as high as a factor of 10 (or whatever bound is deemed necessary), but they do not have to list, detect, or design for all specific circumstances that might result in a factor of 10 error. Essentially, if the effects are captured by the measures the system is already collecting, then the control system will respond to them.

The advantage of control is also a drawback, however, because the modeling part is quite challenging. The models described above are written as differential or difference equations (depending on whether the system is continuous or discrete) and most computer and software engineers are not trained to develop such models.

A related drawback is that computer systems are typically required to be much more general than the physical systems for which control was originally designed. For example, control systems are developed for airplanes by designing a specific controller for a specific plane. Such a design methodology makes sense because a plane has a fixed function and is not expected to change considerably over its lifetime. In contrast many computing systems are quite general (a typical computer system is supposed to be able to compute any computable function, but a bridge is not required to span any bridgeable gap). Therefore, the behavior of a computing system can vary tremendously over its lifetime depending on how it is used. The problem here is developing models that account for the generality and potentially widely varying usage of a self-aware computing system.

E. Example: Controlling the Video Encoder

To demonstrate the advantages of control systems for dynamic adaptation, we consider tailoring system resource usage to meet latency constraints with minimal energy for mobile video encoding applications. Specifically, we highlight results of the POET control-based resource management system [33], although several other examples exist in the literature; e.g., [57], [58], [59]. We return to the Intel Mobile Haswell and ARM big.LITTLE systems mentioned above. Here the Intel system is part of a Sony tablet, while the ARM system is part of an ODROID XU-E board. We run the x264 video encoder on both and report results of a prior study demonstrating the benefits of control theoretic solutions—that adapt to differing inputs—over using the race-to-idle heuristic. On each system we encode a video consisting of three distinct scenes. The first is difficult to encode and requires all resources—for a brief period—to meet the latency requirement. The second and third scenes are easier and place differing demands on the two different hardware systems.

Figure 6a shows the results of using the race-to-idle heuristic in this scenario, with time (measured in frames) on the x-axis and normalized latency (top) and power in Watts (bottom) on the y-axes. The vertical dashed lines show the scene changes (at frames 500 and 1000). As can clearly be seen in the figure, the first scene is the hardest, requiring maximum resources to meet the latency constraint, and consuming the most power. The other two scenes are easier, although which is easiest depends on the hardware platform.

Figure 6b shows the results of an adaptive controller (from [33]) applied to this same problem. The control system keeps the latency near the target, while greatly reducing energy on both platforms. The mean absolute percentage error between the desired and achieved latency is less than 3% across both platforms. The energy savings is nearly 50% on the ODROID and about 11% on the Vaio.

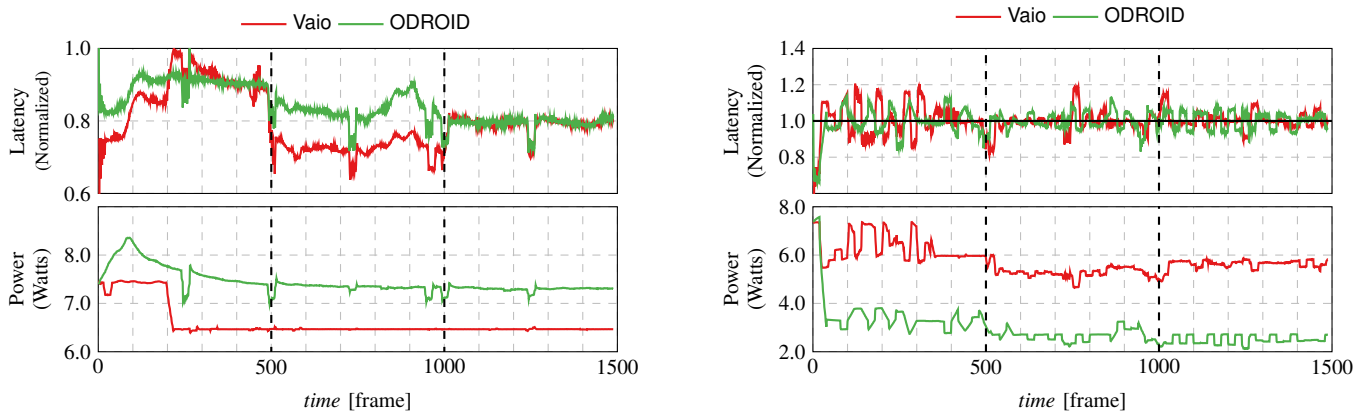
These results demonstrate the potential benefits of applying control theoretic solutions to manage computing systems to meet goals. Here the constraint is on latency and the objective is to minimize energy consumption, but the principles could apply to a wide variety of computing systems' goals. Control approaches have been demonstrated to provide excellent outcomes when the behavior of the computing system can be modeled in advance. In this example, we use a parameterized control approach designed to be portable across systems [33], but even that approach requires a separate specification of the application's (video encoder, in this case) response to various systems resources. Clearly, greater flexibility and generality is needed to realize a truly self-aware computing system.

F. Handling Conflicting Goals

As mentioned above, an embodied self-aware system will often have conflicting goals. In fact, the video encoder example from above is such a case, where the performance constraint is in conflict with the desire for low-energy. This example demonstrates one way to handle such competing goals: set one as a constraint, and another as an objective to be optimized. Additionally, one advantage of control systems is that advanced control methods allow the specification of multiple goals [44], [54], [60]. In such situations the controller essentially solves constrained optimization problems with multiple constraints. Because the control mechanisms all have formalisms associated with them, the controller can report back to a human operator if the constraints have no feasible solution given the set of actions available to the controller. A trivial extension would prioritize the goals and the controller could ignore lower priority goals until a feasible solution is found [61].

IV. Self-aware Assessment

Comprehensive observation (Section II) forms a solid basis for correct self-assessment and the behavior and performance



(a) Race-to-idle

(b) Adaptive Control System

Fig. 6. Controlling a video encoder on two different mobile platforms. (a) shows the race-to-idle heuristic which always keeps latency below the target (1.0, in these figures), but consumes much more power (and thus energy) than required. (b) shows the effects of the using adaptive control to solve this problem, which sometimes violates the latency goal, but reduces energy consumption significantly.

of the system has to be continuously monitored and compared to the given goals and objectives (Section III) for achieving broad self-awareness. As discussed in the previous section, control theory provide means to monitor deviations from goals to bring the system adatpively back on track to meet its objections. However, self-awareness implies monitoring on a system level above individual controllers to assess all system functions such as sensors and the quality of data, the communication with other systems, and the relevance of a system's actions in a given situation. However, this area is still in its infancy and in the following we describe techniques and results that point towards this vision, but a comprehensive methodology or a complete toolbox are still missing.

A. *Confidence based assessment*

Interestingly, just as in observation, confidence has also been used as quality metric for the system's performance.

For instance, specific confidence metrics have been used to improve and guide machine learning. Neshatpour et al. [62] partition the Convolutional Neural Network (CNN) AlexNet [63] into Micro CNNs which allows them to iteratively improve the classification until a given confidence threshold is obtained. The used confidence metric is a side effect of the CNN classification reflecting how close a given match is to images of the training set. Hence, it is a self-assessment that facilities a desired trade-off between accuracy and computing effort.

In another example, Forooghifar et al. [64] apply confidence guided classification to the detection of epileptic seizures. Two variants of Support Vector Machines (SVMs) are used for categorization, one yielding high accuracy with high effort and one, which is less complex with lower accuracy. The used confidence metric is another SVM classifier that predicts how well the low and the high effort SVM variants will perform on a given input pattern. It is used to decide at run-time which of the two are invoked.

Based on the insight that different Machine Learning (ML) algorithms work better on some tasks than others, Kholerdi et al. [65] provide three different classifiers for image categorization: SVM, Neural Network (NN) and Naive Bayesian. Only one of them is used, however, as long as it yields sufficiently high confidence on its classification. But if the confidence drops below a threshold, a second classifier is invoked. Again, self-assessment is used to monitor and assess the system's behavior and to steer its quality above a target level while keeping the costs and effort low.

Machine learning algorithms like CNNs or SVMs provide error estimates that can conveniently be used as confidence metrics [62], [65] or explicit additional effort is expended to compute the confidence [64], [6]. As discussed by Taherinejad et al. [66] it may resemble probability [62], [65], [64] or distance metrics [6] with different pros and cons.

In all these cases confidence as a quality metric of a specific task, like image classification, is immediately used to guide the task itself. A broader notion of self-awareness can more extensively steer the system's operation by changing what data is collected in the first place or what help is requested from the outside. Preden et al. [7] develop the concepts of attention and situation awareness to steer data collection and analysis in a personal health monitoring system. It determines the health condition of a patient and is equipped with sensors for measuring pulse rate, acceleration, position, heart rate, oxygen levels, and other vital signals. An important part of the health assessment constitutes the awareness of the situation and the patient's model. Is she indoor or outdoor, climbing stairs or running, sitting at a desk or driving a car? A comprehensive situation awareness is a prerequisite for a correct health assessment. In principle, the more sensors and sensor data that are available the better the assessment will be. However, for most situations only a relatively small subset of the sensory data is required. Thus, the system proposed by Preden et al. [7] uses only few sensors in common situations. For instance, only the pulse rate and acceleration are required to tell apart resting, driving a car and indoor walking. But to distinguish between upstairs and downstairs walking an additional sensor for altitudes has to be invoked. The mechanism to decide

which data is collected and what analysis steps are taken is called *attention*. Essentially, it is a flexible resource allocation technique, that can pause or start activities, invoke or retire sensors, and request additional information from other agents.

Similar concepts have been proposed and studied by Anzanpour et al. [67] to improve Early Warning Score (EWS), a standard medical procedure in hospitals to monitor the health of stationary patients. Similar to Preden's work several sensors are used to establish the environmental situation of the patient and his mode of activity. This understanding in turn facilitates a more accurate assessment of health risks. For instance, a high heart rate is expected if the person is running, but signifies a potential hazard if he is resting on a couch. Anzanpour and coworkers demonstrate that a comprehensive situation awareness can not only improve the quality of assessment, but can also significantly reduce the energy expenditure of the monitoring system leading to longer battery usage.

B. Architectures for Self-Awareness

Over the last two decades the research community has developed a number of architectures for agent controllers as part of self-* systems with different emphases. Key examples include Observe-Decide-Act (ODA) [71] and Monitor-Analyze-Plan-Execute over a shared Knowledge (MAPE-K) [72]. However, while systems based on these models often possess some level of self-awareness, the models do not explicitly capture self-awareness concerns. More recently, reference architectures have been developed to explicitly emphasize these concerns and provide an architectural perspective on a system's (possible) self-awareness capabilities.

Two widely known approaches are the EPiCS architecture [68], [69] and the *Learn-Reason-Act* loop [70], as illustrated in Figure 7. Both include limited self-awareness consisting of self-monitoring, self-models and a way to reason about their own capabilities. The EPiCS model (Figure 7a) distinguishes between five kinds of awareness inspired by Neisser's levels of self-awareness in humans [73]. *Stimulus awareness* means that the system knows of the stimuli acting on it and can respond to events. *Interaction awareness* means that the system knows that external stimuli and its own response constitute interactions with other systems. *Time awareness* makes the system knowledgeable about historical and likely future phenomena. *Goal awareness* means the system knows about its own goals, objectives, preferences and constraints. *Meta-self-awareness* means that the system obtains knowledge of its own awareness. Goal awareness and meta-self-awareness allow the system to manage its goals during its lifetime, adapt its preferences and resource usage to the changing needs and environmental conditions. Lewis et al.[68] report that an EPiCS based system effectively handles dynamism and uncertainty in application goals, workloads and environmental conditions. From their experiments it can be concluded that the key advantage of self-awareness properties is increased flexibility which bears fruit if the system's internal condition and its goals, and the environment is subject to a high degree of uncertainty and dynamism. If uncertainty is low and interaction dynamics is missing, fixed design-time solutions are always satisfactory.

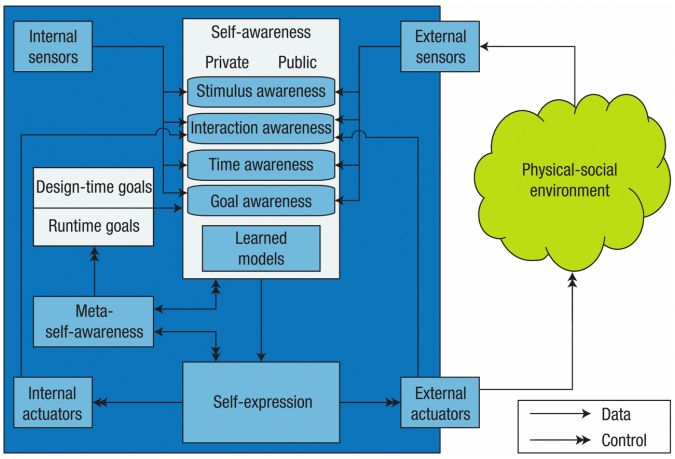
The model-based learning and reasoning loop (LRA-M loop), introduced by Kounev et al. [70] and illustrated in Figure 7b, improves the self-model dynamically based on continuous observations and a learning and reasoning processes. Reasoning here means a mechanism that can find deviations from expected behavior and can trigger new actions or new combinations of actions, thus making the system adaptive.

Recently a formal model of self-reflection has been proposed [74] that allows a self-aware agent to turn its attention to any of its own internal processes to assess its behavior and performance. As this also includes the very process of self-reflection as possible target it is called *recursive self-reflection*. For this to work, abstraction is crucial to significantly reduce the amount of information and complexity of the target process under assessment. Also, the abstraction has to be automatic to be fully flexible and for applying the reflection and assessment to any target process without arbitrary limiting it to a design-time defined set of processes. So far recursive self-reflection is the most generic and general mechanism to self-awareness since it can be applied to about any internal process of a system in any application as long as the process' inputs and outputs are observable and its goals are known. However, its effectiveness, efficiency and utility still have to be demonstrated in a realistic setting.

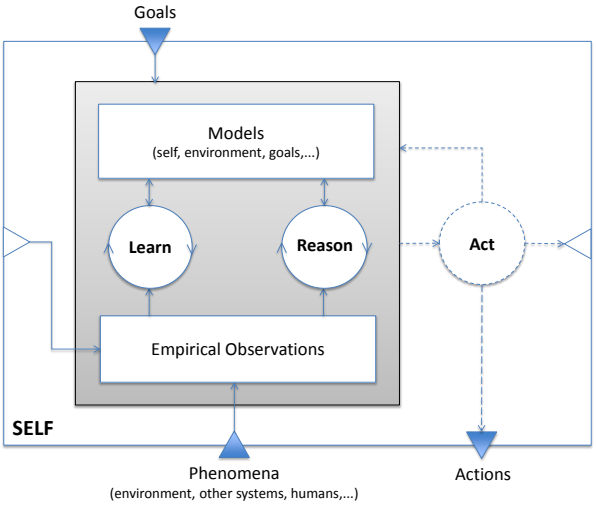
C. Self-assessment, Expectations and Goals

In Section III we discussed goals from the perspective of control theory. But in fact, goals usually appear at several hierarchical levels, they may change dynamically and are sometimes mutually contradicting. Even for the limited domain of many-core resource allocation Rahmani et al. [75] have shown that goals change over time due to varying application workload, changing internal state (e.g., discharging of the battery), and fluctuating user requirements. When the battery is full and the workload high, high throughput optimization is a sensible policy, but when the battery is low and the user requests low-power mode, an energy conserving policy is appropriate even if the workload is high. Moreover, goals are not always equal. If the system faces acute thermal overheating, a low-power request from the platform management should overrule user or application preferences for high performance. Hence, in complex Cyber-Physical Systems (CPSs) with multiple components and sub-systems it is evident that we face a hierarchical, dynamic goal structure [76].

In robot planning and cognitive architectures the management of goals has a long tradition [77]. Goal Driven Autonomy (GDA) [78], [79] is a conceptual framework for dynamic planning in autonomous systems. It starts with an initial goal and generates new goals when differences between an expected and observed state are detected. Jaidee et al. [80] have integrated GDA into a Q-learning loop to dynamically improve goal selection based on the observed system behavior and Shamsa et al. [81] have used it to manage on-chip resources in many-core systems. Because GDA allows for dynamic generation, retirement and re-prioritization of goals, it is an effective framework for dynamic management of hierarchical goal structures. Thus it matches well with self-awareness which requires to compare observed behavior with



(a) The EPiCS architecture [68], [69].



(b) The *Learn-Reason-Act* loop [70].

Fig. 7. Two reference architectures for self-aware computing systems.

expected behavior. The latter can be derived from currently active goals. Hence, a promising avenue of research is the integration of self-awareness with GDA or similar frameworks for dynamic goal management, which hopefully leads to self-aware, autonomous computing systems with a closed loop between continuous, dynamic self-assessment and goal management, as illustrated in Figure 8.

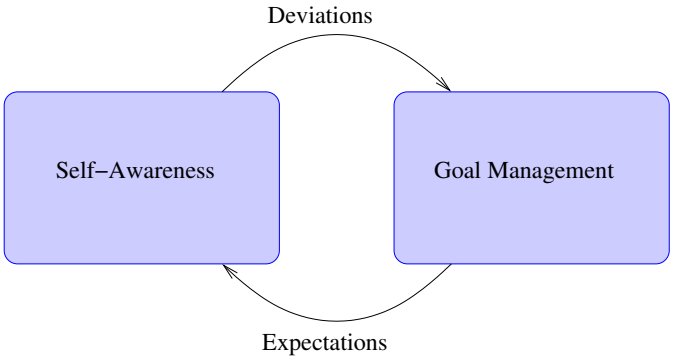


Fig. 8. The interplay between self-awareness and dynamic goal management.

D. Summary

The architectures and approaches discussed in this section provide examples of the state of the art for how to realize self-awareness and self-assessment in computing systems. It illustrates that there is no single best solution but two principles become clear. (i) Self-awareness requires an elaborate assessment of the system's and the environment's current state, which has to be derived from the sensory data and interpreted in the given context. (ii) Explicit modeling of goals and expectations are necessary to relate the current state to desired states.

How these principles are realized depend on the resources available and the specific application with all its requirements and constraints.

V. Sample Embodied, Self-Aware Computing Systems

We now present two exemplars of embodied, self-aware computing systems: CPSoC and healthcare monitoring. For each exemplar we analyze which features of embodied self-awareness they incorporate and comment on those features that are missing.

A. CPSoC

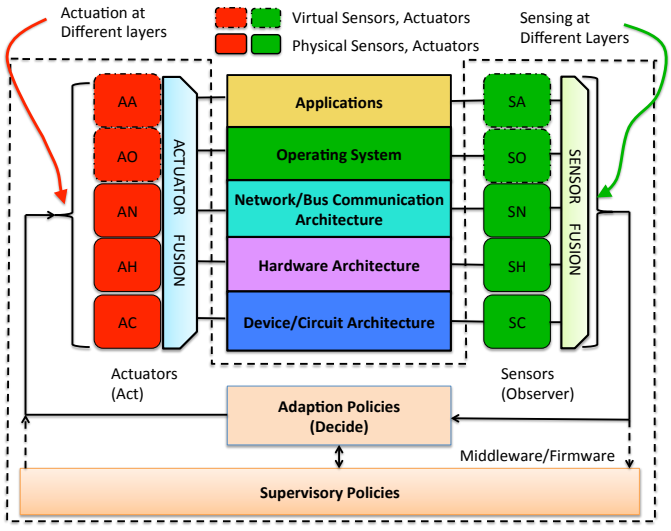


Fig. 9. Cross-layer virtual sensing and actuation in CPSoC [9].

CPSoC [82], [9] is a System-on-Chip (SoC) architecture with many physical and virtual sensors, an Introspective Sentient Unit (ISU) and middleware for building a comprehensive model of the SoC status. Figure 9 shows the usage of sensors and actuators from the physical to the application level. A variety of real and virtual sensors monitor temperature, power consumption, bus and network load, occupancy of buffers,

TABLE I
EWS TABLE EXTRACTED FROM [83].

Score	3	2	1	0	1	2	3
Heart rate[1]	<40	40–51	51–60	60–100	100–110	110–129	>129
Systolic BP[2]	<70	70–81	81–101	101–149	149–169	169–179	>179
Breath rate[3]		<9		9–14	14–20	20–29	>29
SPO_2 (%)	<85	85–90	90–95	>95			
Body temp.[4]	<28	28–32	32–35	35–38		38–39.5	>39.5

[1]beats per minute, [2]mmHg, [3]breaths per minute, [4] °C

Quality of Service (QoS) parameters, etc. Based on the sensory data CPSoC has the ability to adapt to varying environmental conditions and application requirements using multiple co-operating and hierarchical ODA loops, that effectively track multiple platform and application goals. In addition, CPSoC provides predictive models and on-line learning capabilities to build and continuously improve models for performance, power and energy consumption, that allow for better prediction and control of these parameters. The predictive models are based on linear regression and NN based techniques.

CPSoC exhibits several features of a self-aware SoC in that it uses sensors extensively to build up, and continuously improve, a self-model of its own state with respect to various performance metrics, power, energy, and temperature. While it also features a significant dynamic learning component, the overall architecture is hard-coded and determined at design time. If a new application comes with a novel objective (e.g., a certain kind of user experience or accuracy-energy trade-off not considered by the designers), CPSoC has no means to relate this objective to sensory data and actuator knobs, and to devise a strategy to meet this novel goal.

B. Healthcare Monitoring

EWS is a medical procedure widely used in hospitals since the 1990s to closely track the condition of a patient [83]. Table I shows the parameters measured and the scores given. The total score is simply summed up and can be between 0 and 15, with higher total scores indicating anomalous health states based on the specific ranges of the parameters.

The traditional EWS is a purely manual procedure implemented by hospital nurses. Anzanpour et al. [84] have designed a wearable, battery driven device that automates the measurements and the computation of scores, resulting in a number of benefits. It relieves both patient and nurse from the tedious measurement procedures, it can collect data continuously, and the patient can move around and even leave the hospital while still being monitored. While this is much more convenient for the patient and allows for more comprehensive monitoring it also implies several complications, because the vital signs change with the type of activity engaged by the patient. The scores in Table I assume that the patient is resting in bed. If this assumption is violated the scores have to take the current activity into account. Consequently, Anzanpour et al. [67] have developed an EWS monitoring device that collects more sensory data, estimates the activity mode and the environmental situation and then computes a refined score. Moreover, in a wearable device sensor data is not always fully reliable because sensors may break or detach from the body. Therefore, the improved EWS monitor also integrates an estimate of the reliability of the individual sensor inputs into the overall assessment.

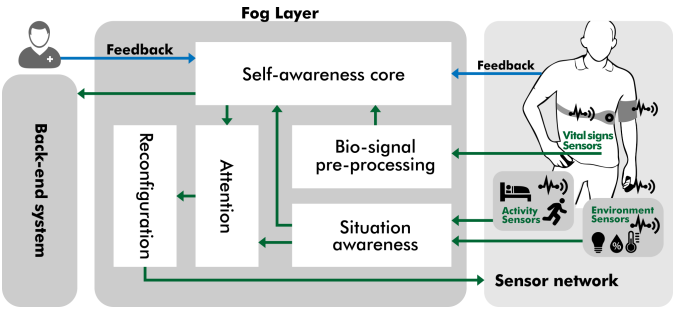


Fig. 10. The architecture of the self-aware EWS [67].

As Figure 10 illustrates, the sensor data is used to estimate the activity mode of the patient, the environmental situation like location, velocity of movements, time of day, etc. and the reliability of the individual sensory data itself. Based on this assessment of the overall situation, the current health of the patients is assesses more accurately than based on the vital signs alone. In addition, because this is a battery driven device, the assessment of the situation is used to operate with power efficiency by adapting the rate of measurements and computation. Over a full day of operation half the energy can be saved without compromising accuracy of assessment [67].

Recently, Anzanpour and coworkers proposed an improvement of the self-aware EWS by explicitly modeling goals which dynamically change depending on the observed situation. Similar to the self-aware EWS the situation is assessed, and the goal, that fits best to the current situation is then invoked. Under abnormal health conditions the goal is “high performance monitoring”, and otherwise it is “long lasting monitoring” or “accurate data collection”, depending on the battery load level. The accurate assessment of the patient’s health status has usually the highest priority, but when energy is scarce, compromises are made.

As the tasks become more complex and dynamic, comprehensive assessment of the environment and the system’s own state turns out to be exceedingly beneficial. This is nicely illustrated by the three instances of the wearable health monitoring devices described above, the traditional EWS [84], the self-aware EWS [67] and the goal driven, generalized EWS [85].

These exemplars exhibit, in increasing quality, several core features qualifying them as partially self-aware. They use multiple sensor sources to build up robust models of the patient’s health, the reliability of sensors and the situation. This elaborate awareness of the situation facilitates accurate monitoring of the patient’s health under various situations while keeping power and energy efficiency reasonably low. These examples demonstrate one core claim of self-awareness

in embodied computing systems: more elaborate and comprehensive assessment of the situation leads to better decisions.

But they fall short with respect to other key abilities. Critically, they do not exhibit general learning capabilities to build up and improve increasingly elaborate models of themselves and the world, and to identify and establish novel correlations and causal dependencies. This inherently limits their adaptability to new and unforeseen situations. In fact, universal learning capabilities are required to cope with and adapt to entirely novel situations.

Furthermore, key features of *embodiment* are only rudimentarily developed. While a rich set of sensors, which both exemplars contain, is the basis for embodiment, neither system treats its own physical body sufficiently explicit to clearly delineate it from the bodies in the environment. Also, spatial relations, time relations and situatedness are weakly developed and only implicitly present. Thus, both exemplars do not serve as complete cases of embodied systems, but they have most of the essential ingredients and it seems a rather small step towards truly embodied, self-aware computing systems. We consider them more as inspiration rather than perfect examples. And in fact, the exponential growth in the number of devices, and their increasing computational and sensory capabilities results in a trend towards more general solutions, which eventually will be embodied, self-aware computing systems that can meaningfully navigate even the most unexpected situations. .

VI. Dynamic Learning, adaptation, self-optimization

Prior sections establish that self-aware computing systems must be aware of goals and be capable of performing self-assessment to determine whether, and how well, they are meeting those goals. We have noted that control systems provide a general theory of meeting goals in dynamic environments. At first glance, it might seem that if we simply embed control into computing systems, then we would have an embodied self-aware computing system that adapts behavior to meet goals. Such an approach, however, would still face many limitations including the fact that this hypothetical system still would not be capable of performing self-assessment. The ability to perform this self-assessment is key to transforming a computing system that adapts through control mechanisms into a self-aware computing system that—not only adapts in response to external stimuli—but has the higher-order ability to evaluate its own adaptation and find better ways of adapting in the future.

This ability to perform self-assessment is not just an academic goal. Rather, it is essential for our ability to build computing systems that respect constraints while optimizing objectives. Constructing such computer systems that can automatically determine the best way to meet their goals will be essential to rapidly and successfully deploying computer systems in the face of increasing complexity.

This capability can be realized by imbuing computing systems with intelligence, specifically an intelligence that allows them to reason about themselves and their environment. Thus, an ideal self-aware computing system would be imbued with artificial intelligence, and determining, or creating, the techniques that would allow the full power of such intelligence within a computing system is an ongoing research challenge. To date, the deployed techniques for these systems fall into the subset of artificial intelligence known as machine learning.

A. Learning for Self-Assessment

The rapid growth in computing system complexity is affecting users and operators at many different levels of the system stack, from processors that have multiple heterogeneous core types [86], [87] to foundational software packages (e.g., Hadoop MapReduce) that have many 100s of configuration parameters [88], [89]. These complex configuration spaces in turn induce a complicated tradeoff space in computing system behavior. For example, combinations of processor resources create different performance/energy tradeoffs [31], while combinations of parameter settings in MapReduce produce various tradeoff spaces, including memory usage vs. throughput [90]. Adding further challenge, the large number of possible configurations and their interactions create local optima in these tradeoff spaces such that formerly effective heuristics can produce far from optimal behavior [31], [91].

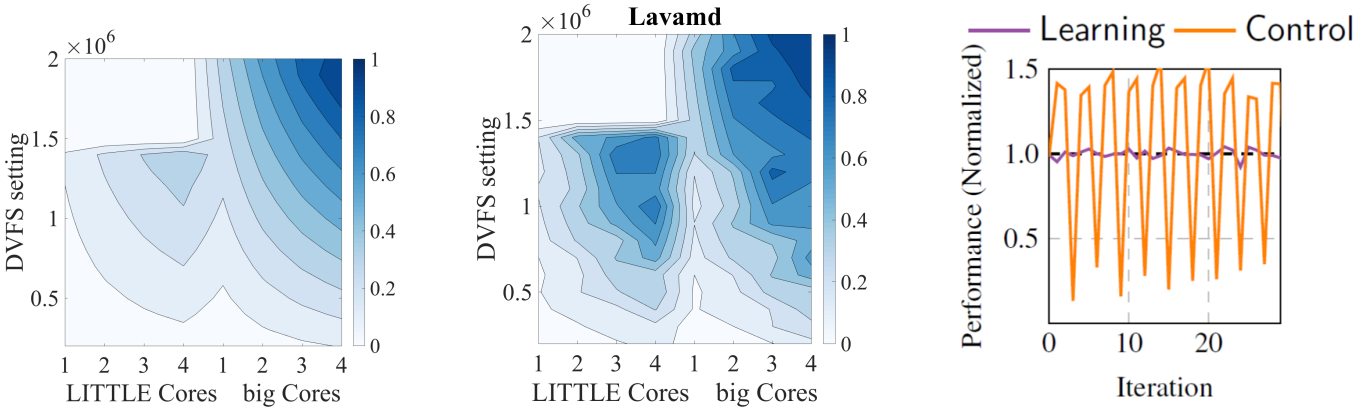
Fortunately, recent years have seen an explosion of artificial intelligence and machine learning techniques that are well-equipped for modeling complicated tradeoff spaces to find and avoid local optima. Many examples exist for modeling both hardware [92], [93] and software systems [94]. A recent survey details applications of artificial intelligence and machine learning to computing systems (with a focus on computer architecture) [95]. Compared to commonly used heuristics these learning-based approaches provide much more rigorous and well-founded methods to avoiding local optima and driving the computing system to meet its constraints while optimizing its objective function.

B. Example of Learning for Computing System Management

We present a brief example showing how learning systems can avoid local optima and produce good outcomes for self-aware computing systems. Here we explore the LAVAMD application running on the ODROID board from the prior control sections. Figure 11 shows the details for this example.

We begin by comparing x264 and LAVAMD's performance on this architecture. The contour plots in Figures 11a and 11b show how differently these two applications respond to the same resources. These figures show resource on the x (cores) and y (frequency) axes, while performance is indicated with increasing darkness. While x264 has a fairly smooth increase in performance for increasing resources, LAVAMD has several local optima and represents a much less smooth space.

The difference between x264 and LAVAMD response to resources creates a difficult problem for controlling these applications to meet latency constraints. Section 6 showed that, even with varying inputs, control designs can still drive the encoder to a target latency with low energy. However, if we use the same control model that worked well for x264 to manage resources for LAVAMD, then we get the wildly



(a) x264 Performance (b) LAVAMD Performance (c) LAVAMD Control

Fig. 11. This figure shows the difference in complexity of the x264 and LAVAMD benchmarks running on a heterogeneous ARM big.LITTLE system. (a) shows x264's performance as a function of cores (x-axis) and clockspeed. (b) shows the same for LAVAMD. (c) shows the results of using a control system built with the model from (a) to control LAVAMD and compares it to a learning system that is capable of finding the local optima exhibited in (b).

oscillating behavior shown in Figure 11c. Using this control system for LAVAMD, the latency is alternatively far above or far below the desired target. This oscillation happens because the controller was built for x264 and the difference between x264's model and LAVAMD's is too large for the controller to correct.

In contrast with the controller, Figure 11c also shows what happens if we use a Hierarchical Bayesian Model (HBM) to select a resource configuration that gives the desired latency with minimal energy [96]. In this case, the learner is able to find a suitable resource configuration. Furthermore, because LAVAMD does not exhibit dynamic changes in behavior on this system, the learner provides near perfect control of the system.

C. Research Challenges

Therefore, machine learning methods provide powerful tools that could be used to add self-assessment capabilities to computing systems. These techniques—in and of themselves—do not represent a complete solution to building self-aware computing systems, however. While ML techniques face fundamental challenges due to their lack of explainability that is critical for self-assessment, we identify at least two additional challenges that must be addressed to successfully deploy learning in self-aware computing systems:

- *Addressing Computing System Dynamics*: A recent position paper on challenges facing machine learning in deployed systems notes the issues of deploying AI to operate in dynamic environments, i.e., environments that may change, often rapidly and unexpectedly, and often in non-reproducible ways [97]. Computing systems are fundamentally dynamic: resources may be added to the system or fail during operation, while applications transition through different phases, each of which has different optimal resource allocations.
- *Making Learners Aware of System Goals*: In addition, a typical way of deploying learners in computer systems is to train the learning system to predict some metric of behavior—typically relating to the constraints and objectives—as a function of the computing system's configurable parameters. The training objective is generally to maximize prediction accuracy. A recent study, however, shows that there is a point where continuing to focus on learning accuracy can actually degrade the overall system behavior for computing systems that must deal with constrained optimization problems [91]. Instead, that study suggests that it may lead to better results if the learning system is aware of the goals its predictions will ultimately be used to achieve and is trained to realize those outcomes.
- *Explanable Outcomes*: As discussed above, embodied self-aware systems will have constraints that must be met for correct behavior. One challenge to deploying learning methods in such systems is what happens when the constraints are not met in practice. If we only use classical control methods, for example, if the constraints are not met we can use the control theoretic formalisms to reason about what went wrong. In contrast, most learning mechanisms do not have the formalisms required to distinguish when a failure case is fundamental—e.g., the system simply does not have the capability to meet a goal—versus the case where the failure is due to the learner itself—e.g., the learner produced a bad prediction that caused it to violate a constraint. This issue is further compounded by the fact that the vast majority of learning approaches deployed today are based on finding correlations between data, and it is well-known that correlation does not imply causation. One approach to this challenge may be deploying learning techniques that explicitly work to find causal relationships between data and thus offer some ability to explain the resulting decisions.

D. Combining Learning and Control

At this point, we have seen that control theory provides a whole body of knowledge for developing systems that understand goals and adapt to meet them in dynamic environments.

The disadvantages of control theory are that: (1) it generally takes expertise to apply and (2) it is not, inherently, a great match for computing systems due to their complexity. Conversely, learning techniques are unrivaled at modeling complex systems, but they have the disadvantages described above: (1) limited ability to handle dynamics, (2) unclear relationship to computing system goals (constraints and objectives), and (3) limited ability to understand the decision-making process.

Intuitively, it seems clear that we should explore combinations of learning and control that might result in a complete approach that uses control to meet goals in dynamic environments and learning to provide self-assessment and modeling for complexity. Several researchers have recently begun exploring such combinations. Recht et al. have proposed several approaches for combining statistical learning models with optimal control [98], [99]. Simultaneously, Hoffmann et al. have developed OS [100], [101] and hardware-level resource management systems [102], [103] that combine learning and control to provide both energy and latency guarantees in dynamic environments. This prior work, however, still requires expertise in both learning and control methods to effectively deploy the proposed solution. Getting this combination into widespread use within self-aware computing systems will require further development of abstractions and interfaces that allow control and learning to work together while maintaining—as much as possible—control theory's formal guarantees.

E. An Example of a Hybrid Learning and Control System

The prior example showed how learning can accurately model the complex, application-specific interactions of hardware resources and create good outcomes where control alone could not. We conclude this section by returning to this example and discussing how it is handled by the CALOREE approach [101] for combining learning and control capabilities within a computing system.

CALOREE manages resources in computing systems to meet latency constraints with minimal energy. The heart of CALOREE is an adaptive control system that observes whether or not latency goals are being met and adjusts resources accordingly. As shown in the previous example, however, applications can have very different behavior for the same hardware and it is not possible to build one control model that captures all possible application behavior. So, CALOREE starts applying control with a generic model of application response. As the controller runs, it naturally collects feedback which it sends to an ML engine. The ML engine uses this data to create an application-specific model of response to resources, customizing the controller online.

In addition to sending this learned model, the learner also sends its confidence in the learned model. The confidence is used to update the control response. If the learner is very confident in its model then the controller will act quickly to any changes in latency. When the learner is less certain, the controller will slow down and collect more observations before reacting.

As it turns out, communicating both the model and confidence appears to be key to getting this hybrid learning and control approach to work well for very complex applications, like LAVAMD (shown in Figure 11b). To demonstrate this, we illustrate the results of using two different approaches to combining learning and control (first published in [101]). The first, simply replaces the standard system modeling done to develop a controller with a machine learning approach. The second is CALOREE, which uses the same learning system, but also communicates uncertainty in the learned model.

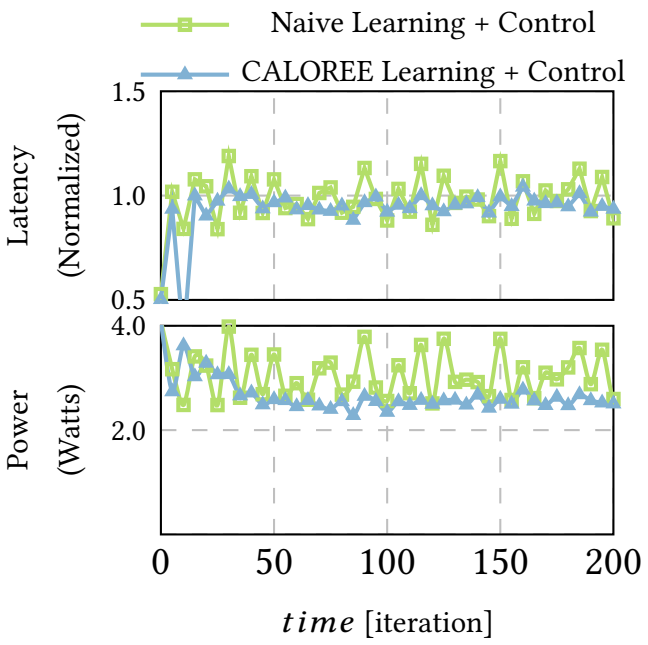


Fig. 12. The latency (normalized to the constraint) and power for the LAVAMD application (from Figure 11b) controlled by two different hybrid learning-control systems. The first is simply a straightforward combination that replaces the standard control modeling step with learning. The second is the CALOREE system that does learning online, and communicates both a learned model and confidence in that model to the controller.

Figure 12 shows the results of controlling LAVAMD with these two different approaches. As we can see from the first approach does little better than a control system built with a generic model (from Figure 11c). CALOREE, however, drives LAVAMD to meet its goals because the controller knows the uncertainty in the model and slows down to collect more observations. Using CALOREE, the application not only does a much better job of meeting its constraints, but it also has a lower energy consumption because it is not oscillating. The LAVAMD application does not exhibit much dynamic behavior, but this combination of learning and control makes the system robust to other, external changes. For example, in the original CALOREE paper LAVAMD controlled by CALOREE can still meet latency constraints even if other applications enter and leave the system during execution.

VII. Conclusions and Outlook

This article sets the context for embodied self-aware computing systems by building on the notions of computational embodiment and computational self-awareness that empowers classical computational platforms to adapt, learn and operate autonomously in the face of varying application requirements,

changing goals and conflicting constraints. Using lessons learned from self-aware computing and robotics, we elaborated on the basic properties of embodied self-aware systems, and defined the actions, goals and control in embodied self-aware computing systems. We discussed the role of self-aware assessment and the need to synergistically couple the benefits of traditional control theory with learning strategies to meet dynamically changing goals. In this context, many of today's computing systems lack self-models, or have fixed models implicitly designed into the system. This lack of explicit and adaptive self-models prevents contemporary computational platforms from establishing embodied self-awareness, together with the attendant facilities of dynamic learning, adaptation, and self-optimization that are required to allow continual adaptation to environmental and application changes, as well as dynamically changing goals. We used some sample exemplar platforms (e.g., CPSoC) and applications (e.g., self-aware healthcare monitoring) to highlight initial forays into computational embodiment and computational self-awareness, with both their salient features as well as their shortcomings. Many difficulties remain.

One overarching challenge is to provide a framework for an agent to holistically reason about its situation. We have discussed four capabilities of an agent: (i) understanding the environment, (ii) understanding itself, (iii) understanding its goals, and (iv) having the appropriate control techniques to determine how to meet goals in the current environment. While there are large bodies of work in each of these domains, their integration in an agent is not obvious.

As a start we need a formalism to represent what is known about the environment, the agent itself, the agent's goals, and the means to accomplish these goals, such that the agent can coherently reason about these capabilities. Our requirements on flexibility prevent us from settling on a fixed set of knowledge objects or a static structure.

We have argued for automated abstraction at run-time because it is not known beforehand which are the relevant concepts that an agent should deal with. Consequently, the observation process results in dynamically generated, novel and abstract concepts. Also, we have argued that goals are generated and applied dynamically to match a given situation. There are always some primary goals—that are defined at design time and cannot change (survival, efficiency, etc.)—but most of the derived short term goals, steering the immediate actions, should be context dependent.

Finally, we have emphasized the role of dynamic learning to continuously improve the manner in which the agent interacts with and controls the environment.

Thus, how can the agent relate the abstracted observations to the goals generated and to the steadily growing and changing capabilities of control? How does the agent know the relevance of an abstracted observation for its goals? What new sub-goals should be generated because of these new observations? And what are the goals that can be realistically achieved with the recently improved control technique?

Obviously, a common frame of reasoning is required for the agent to assess the relevance, desirability and implications of all the observations about the environment and itself. Given the high expectations on flexibility, adaptability, and learning, the formulation and study of such frameworks are research challenges for years to come.

ACKNOWLEDGMENTS

Henry Hoffmann's work on this project is supported by NSF (CCF-1439156, CNS-1526304, CCF-1823032, CNS-1764039), with additional support from the Proteus project under the DARPA BRASS program and a DOE Early Career award. Axel Jantsch acknowledges financial support from the Austrian Government through BMVIT/FFG in the project SAMBA (FFG 855426) under the *ICT of the Future* program and the project SAVE (FFG 864883) under the *Production of the Future* program, and the Christian Doppler Gesellschaft for support of the CD-Lab *Embedded Machine Learning*. Nikil Dutt's work was partially supported by the NSF under grants CCF-1704859 and WiFiUS CNS-1702950.

REFERENCES

[1] L. B. Smith and M. Gasser, "The development of embodied cognition: Six lessons from babies," *Artificial Life*, vol. 11, no. 1-2, pp. 13–29, 2005. [Online]. Available: https://doi.org/10.1162/1064546053278973

[2] J. L. Krichmar and G. M. Edelman, "Brain-based devices for the study of nervous systems and the development of intelligent machines," *Artificial Life*, vol. 11, no. 1-2, pp. 63–77, 2005. [Online]. Available: https://doi.org/10.1162/1064546053278946

[3] N. TaheriNejad, A. Jantsch, and D. 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.

[4] T. Buchgraber and D. Shutin, "Distributed variational sparse bayesian learning for sensor networks," in *Machine Learning for Signal Processing (MLSP), 2012 IEEE International Workshop on*, September 2012, pp. 1–6.

[5] F. Al-Turjman, *Cognitive Sensors and IoT: Architecture, Deployment, and Data Delivery*. CRC Press, 2017.

[6] M. Götzinger, N. TaheriNejad, H. A. Kholerdi, A. Jantsch, E. Willegger, T. Glatzl, A. M. Rahmani, T. Sauter, and P. Liljeberg, "Model-free condition monitoring with confidence," *International Journal of Computer Integrated Manufacturing*, 2019.

[7] J.-S. Preden, K. Tammemäe, A. Jantsch, M. Leier, A. Riid, and E. Calis, "The benefits of self-awareness and attention in fog and mist computing," *IEEE Computer, Special Issue on Self-Aware/Expressive Computing Systems*, pp. 37–45, July 2015.

[8] N. TaheriNejad, M. A. Shami, and S. M. P. Dinakarrao, "Self-aware sensing and attention-based data collection in multi-processor system-on-chips," in *15th IEEE International New Circuits and Systems Conference (NEWCAS)*, June 2017, pp. 81–84.

[9] N. Dutt, A. Jantsch, and S. Sarma, "Self-aware cyber-physical systems-on-chip," in *Proceedings of the International Conference for Computer Aided Design*, Austin, Texas, USA, November 2015, invited.

[10] C. S. Daw, C. E. A. Finney, and E. R. Tracy, "A review of symbolic analysis of experimental data," *Review of Scientific Instruments*, vol. 74, no. 2, pp. 915–930, 2003. [Online]. Available: https://doi.org/10.1063/1.1531823

[11] J. Lin, L. Wei, and S. Leonardi, "Experiencing SAX: a novel symbolic representation of time series," *Data Mining and Knowledge Discovery*, vol. 15, no. 2, pp. 107–144, October 2007.

[12] D. A. A. G. Singh, S. A. Balamurugan, and E. J. Leavline, "Literature review on feature selection methods for high-dimensional data," *International Journal of Computer Applications*, vol. 136, no. 1, pp. 9–17, February 2016.

[13] J. Cai, J. Luo, S. Wang, and S. Yang, "Feature selection in machine learning: A new perspective," *Neurocomputing*, vol. 300, pp. 70–79, July 2018.

[14] E. Clarke, H. Jain, and D. Kroening, "Predicate abstraction and refinement techniques for verifying verilog," Carnegie Mellon University, Tech. Rep. CMU-CS-04-139, 2004.

[15] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *Computer Aided Verification*, E. A. Emerson and A. P. Sistla, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 154–169.

[16] Z. S. Andraus and K. A. Sakallah, "Automatic abstraction of verilog models," in *Proceed-ings of 41st Design Automation Conference*, 2004, pp. 218–223.

[17] H. Hermanns, B. Wachter, and L. Zhang, "Probabilistic CEGAR," in *International Conference on Computer Aided Verification*, 2008, p. 162175.

[18] X. Zhang, B. Wu, and H. Lin, "Counterexample-guided abstraction refinement for pomdps," *CoRR*, vol. abs/1701.06209, 2017. [Online]. Available: http://arxiv.org/abs/1701.06209

[19] I. Stolz and K. Keller, "A general symbolic approach to kolmogorov-sinai entropy," *Entropy*, vol. 19, no. 12, 2017. [Online]. Available: http://www.mdpi.com/1099-4300/19/12/675

[20] S. Sarkar, P. Chattopdhyay, and A. Ray, "Symbolization of dynamic data-driven systems for signal representation," *Signal, Image and Video Processing*, vol. 10, no. 8, pp. 1535–1542, Nov 2016. [Online]. Available: https://doi.org/10.1007/s11760-016-0967-5

[21] A. Ray, "Symbolic dynamic analysis of complex systems for anomaly detection," *Signal Processing*, vol. 84, no. 7, pp. 1115 – 1130, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0165168404000581

[22] Y. Li and A. Ray, "Unsupervised symbolization of signal time series for extraction of the embedded information," *Entropy*, vol. 19, no. 4, 1 2017.

[23] M. Götzinger, N. TaheriNejad, H. A. Kholerdi, and A. Jantsch, "On the design of context-aware health monitoring without a priori knowledge; an AC-motor case-study," in *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, April 2017, pp. 1–5.

[24] M. Götzinger, E. Willeger, N. TaheriNejad, A. Jantsch, T. Sauter, T. Glatzl, and P. Liljeberg, "Applicability of context-aware health monitoring to hydraulic circuits," in *The 44th Annual Conference of the IEEE Industrial Electronics Society*, 2018.

[25] T. J. Ross, *Fuzzy logic with engineering applications*. John Wiley & Sons, 2009.

[26] H. Hoffmann, J. Holt, G. Kurian, E. Lau, M. Maggio, J. E. Miller, S. M. Neuman, M. Sinangil, Y. Sinangil, A. Agarwal, A. P. Chandrakasan, and S. Devadas, "Self-aware computing in the angstrom processor," in *DAC Design Automation Conference 2012*, June 2012, pp. 259–264.

[27] L. Chung and J. C. S. do Prado Leite, *On Non-Functional Requirements in Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 363–379. [Online]. Available: https://doi.org/10.1007/978-3-642-02463-4_19

[28] G. C. Buttazzo, G. Lipari, L. Abeni, and M. Caccamo, *Soft Real-Time Systems: Predictability vs. Efficiency*. Springer, 2006.

[29] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013. [Online]. Available: http://doi.acm.org/10.1145/2408776.2408794

[30] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar, "Critical power slope: Understanding the runtime effects of frequency scaling," in *Proceedings of the 16th International Conference on Supercomputing*, ser. ICS '02. New York, NY, USA: ACM, 2002, pp. 35–44. [Online]. Available: http://doi.acm.org/10.1145/514191.514200

[31] D. H. K. Kim, C. Imes, and H. Hoffmann, "Racing and pacing to idle: Theoretical and empirical analysis of energy optimization heuristics," in *CPSNA*, 2015.

[32] H. Hoffmann, "Racing vs. pacing to idle: A comparison of heuristics for energy-aware resource allocation," in *HotPower*, 2013.

[33] C. Imes, D. H. Kim, M. Maggio, and H. Hoffmann, "Poet: A portable approach to minimizing energy under soft real-time constraints," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2015 IEEE*. IEEE, 2015, pp. 75–86.

[34] J. Boyd, "The essence of winning and losing," Online document, 1995. [Online]. Available: https://www.danford.net/boyd/essence.htm

[35] F. P. B. Osinga, *Science, Strategy and War: The strategic theory of John Boyd*. Routledge, 2007.

[36] R. Laddaga, "Guest editor's introduction: Creating robust software through self-adaptation," *IEEE Intelligent Systems*, vol. 14, no. 3, pp. 26–29, May 1999. [Online]. Available: http://dx.doi.org/10.1109/MIS.1999.769879

[37] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, no. 2, pp. 14:1–14:42, May 2009. [Online]. Available: http://doi.acm.org/10.1145/1516533.1516538

[38] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan 2003.

[39] W. Levine, *The control handbook*. CRC Press, 2005.

[40] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[41] C. Karamanolis, M. Karlsson, and X. Zhu, "Designing controllable computer systems," in *Proceedings of the 10th Conference on Hot Topics in Operating Systems - Volume 10*, ser. HOTOS'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 9–9. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251123.1251132

[42] M. Maggio, H. Hoffmann, A. V. Papadopoulos, J. Panerati, M. D. Santambrogio, A. Agarwal, and A. Leva, "Comparison of decision-making strategies for self-optimization in autonomic computing systems," *TAAS*, vol. 7, no. 4, pp. 36:1–36:32, 2012. [Online]. Available: http://doi.acm.org/10.1145/2382570.2382572

[43] A. Filieri, H. Hoffmann, and M. Maggio, "Automated design of self-adaptive software with control-theoretical formal guarantees," in *ICSE*, 2014.

[44] ——, "Automated multi-objective control for self-adaptive software design," in *FSE*, 2015.

[45] A. Filieri, M. Maggio, K. Angelopoulos, N. D'Ippolito, I. Gerostathopoulos, A. B. Hempel, H. Hoffmann, P. Jamshidi, E. Kalyvianaki, C. Klein, F. Krikava, S. Misailovic, A. V. Papadopoulos, S. Ray, A. M. Sharifloo, S. Shevtsov, M. Ujma, and T. Vogel, "Software engineering meets control theory," in *10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015, Florence, Italy, May 18-19, 2015*, 2015, pp. 71–82.

[46] ——, "Control strategies for self-adaptive software systems," *TAAS*, vol. 11, no. 4, pp. 24:1–24:31, 2017. [Online]. Available: http://doi.acm.org/10.1145/3024188

[47] S. Shevtsov, M. Berekmeri, D. Weyns, and M. Maggio, "Control-theoretical software adaptation: A systematic literature review," *IEEE Transactions on Software Engineering*, vol. PP, no. 99, pp. 1–1, 2017.

[48] C. Lefurgy, X. Wang, and M. Ware, "Server-level power control," in *Fourth International Conference on Autonomic Computing (ICAC'07)*, June 2007, pp. 4–4.

[49] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu, "Dynamic voltage scaling in multitier web servers with end-to-end delay control," *IEEE Transactions on Computers*, vol. 56, no. 4, pp. 444–458, April 2007.

[50] R. P. Pothukuchi, A. Ansari, P. Voulgaris, and J. Torrellas, "Using multiple input, multiple output formal control to maximize resource efficiency in architectures," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 658–670.

[51] A. Sharifi, S. Srikantaiah, A. K. Mishra, M. Kandemir, and C. R. Das, "Mete: Meeting end-to-end qos in multicores through system-wide resource management," *SIGMETRICS Perform. Eval. Rev.*, vol. 39, no. 1, pp. 13–24, Jun. 2011. [Online]. Available: http://doi.acm.org/10.1145/2007116.2007119

[52] L. Wang, J. Xu, H. A. Duran-Limon, and M. Zhao, "Qos-driven cloud resource management through fuzzy model predictive control," in *Proceedings of the 2015 IEEE International Conference on Autonomic Computing*, ser. ICAC '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 81–90. [Online]. Available: https://doi.org/10.1109/ICAC.2015.41

[53] K. Angelopoulos, A. V. Papadopoulos, V. E. Silva Souza, and J. Mylopoulos, "Model predictive control for software systems with cobra," in *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '16. New York, NY, USA: ACM, 2016, pp. 35–46. [Online]. Available: http://doi.acm.org/10.1145/2897053.2897054

[54] M. Maggio, A. V. Papadopoulos, A. Filieri, and H. Hoffmann, "Automated control of multiple software goals using multiple actuators," in *ESEC/FSE*, 2017.

[55] Q. Sun, G. Dai, and W. Pan, "Lpv model and its application in web server performance control," in *2008 International Conference on Computer Science and Software Engineering*, vol. 3, Dec 2008, pp. 486–489.

[56] J. L. Hellerstein, V. Morrison, and E. Eilebrecht, "Applying control theory in the real world: Experience with building a controller for the .net thread pool," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 3, pp. 38–42, Jan. 2010. [Online]. Available: http://doi.acm.org/10.1145/1710115.1710123

[57] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time CPU scheduling for mobile multimedia systems," in *SOSP*, 2003.

[58] A. M. Rahmani, B. Donyanavard, T. Mück, K. Moazzemi, A. Jantsch, O. Mutlu, and N. Dutt, "Spectr: Formal supervisory control and coordination for many-core systems resource management," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '18. New York, NY, USA: ACM, 2018.

[59] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu, "Dynamic voltage scaling in multitier web servers with end-to-end delay control," *Computers, IEEE Transactions on*, vol. 56, no. 4, 2007.

[60] C. Imes, L. Bergstrom, and H. Hoffmann, "A portable interface for runtime energy monitoring," in *FSE*, 2016.

[61] H. Hoffmann, "Coadapt: Predictable behavior for accuracy-aware applications running on power-aware systems," in *2014 26th Euromicro Conference on Real-Time Systems*, July 2014, pp. 223–232.

[62] K. Neshatpour, F. Behnia, H. Homayoun, and A. Sasan, "ICNN: an iterative implementation of convolutional neural networks to enable energy and computational complexity aware dynamic approximation," in *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018*, 2018, pp. 551–556. [Online]. Available: https://doi.org/10.23919/DATE.2018.8342068

[63] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, June 2017.

[64] F. Forooghifar, A. Aminifar, and D. A. Alonso, "Self-aware wearable systems in epileptic seizure detection," in *21st Euromicro Conference on Digital System Design, DSD 2018, Prague, Czech Republic, August 29-31, 2018*, 2018, pp. 426–432. [Online]. Available: https://doi.org/10.1109/DSD.2018.00078

[65] H. A. Kholerdi, N. TaheriNejad, and A. Jantsch, "Enhancement of classification of small data sets using self-awareness - an iris flower case-study," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, Florence, Italy, May 2018.

[66] N. TaheriNejad and A. Jantsch, "Improved machine learning using confidence," in *IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*, Edmonton, Canada, May 2019.

[67] A. Anzanpour, I. Azimi, M. Götzinger, A. M. Rahmani, N. TaheriNejad, P. Liljeberg, A. Jantsch, and N. Dutt, "Self-awareness in remote health monitoring systems using wearable electronics," in *Proceedings of Design and Test Europe Conference (DATE)*, Lausanne, Switzerland, March 2017.

[68] P. R. Lewis, A. Chandra, F. Faniyi, K. Glette, T. Chen, R. Bahsoon, J. Torresen, and X. Yao, "Architectural aspects of self-aware and self-expressive computing systems," *IEEE Computer*, August 2015.

[69] P. R. Lewis, M. Platzner, B. Rinner, J. Torresen, and X. Yao, Eds., *Self-Aware Computing Systems: An Engineering Approach*. Springer, 2016.

[70] S. Kounev, Peter Lewis, K. Bellman, N. Bencomo, J. Camara, A. Diaconescu, L. Esterle, K. Geihs, H. Giese, S. Götz, P. Inverardi, J. Kephart, and A. Zisman, "The notion of self-aware computing," in *Self-Aware Computing Systems*, S. Kounev, J. O. Kephart, A. Milenkoski, and X. Zhu, Eds. Springer, 2017, pp. 3–16.

[71] H. Hoffmann, M. Maggio, M. 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*, Sept 2013, pp. 1–10.

[72] IBM Corporation, "An architectural blueprint for autonomic computing," 2006, iBM White Paper.

[73] U. Neisser, "The roots of self-knowledge: Perceiving self, it, and thou," *Annals of the New York Academy of Sciences*, vol. 818, pp. 19–33, 1997.

[74] A. Jantsch, "Towards a formal model of recursive self-reflection," in *International Workshop on Autonomous Systems Design (ASD)*, March 2019, pp. 6:1–6:16.

[75] A. M. Rahmani, A. Jantsch, and N. Dutt, "HDGM: Hierarchical dynamic goal management for many-core resource allocation," *IEEE Embedded Systems letters*, vol. 10, no. 3, September 2018.

[76] A. Jantsch, A. Anzanpour, H. Kolerdi, I. Azimi, L. C. Siafara, A. M. Rahmani, N. TaheriNejad, P. Liljeberg, and N. Dutt, "Hierarchical dynamic goal management for IoT systems," in *Proceedings of the IEEE International Symposium on Quality Electronic Design (ISQED 2018)*, USA, March 2018.

[77] P. Langley, M. Barley, B. Meadows, D. Choi, and E. P. Katz, "Goals, utilities, and mental simulation in continuous planning," *Advances in Cognitive Systems*, June 2016.

[78] M. Klenk *et al.*, "Goal-Driven Autonomy For Responding To Unexpected Events In Strategy Simulations," *Computational Intelligence*, 2013.

[79] D. Choi, "Reactive goal management in a cognitive architecture," *Cognitive Systems Research*, 2011.

[80] U. Jaidee *et al.*, "Integrated learning for goal-driven autonomy," in *Proc. Int. Joint Conference on Artificial Intelligence*, 2011.

[81] E. Shamsa, A. Kanduri, A. M. Rahmani, P. Liljeberg, A. Jantsch, and N. Dutt, "Goal-driven autonomy for efficient on-chip resource management: Transforming objectives to goals," in *Proceedings of the Design and Test Europe Conference (DATE)*, Florence, Italy, March 2019.

[82] N. Dutt, A. Jantsch, and S. 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*, vol. 15, no. 2, pp. 22–27, February 2016, invited.

[83] R. W. Urban *et al.*, "Modified early warning system as a predictor for hospital admissions and previous visits in emergency departments," *Advanced emergency nursing journal*, vol. 37, no. 4, pp. 281–289, 2015.

[84] A. Anzanpour, A.-M. Rahmani, P. Liljeberg, and H. Tenhunen, "Internet of things enabled in-home health monitoring system using early warning score," in *Proceedings of the 5th EAI International Conference on Wireless Mobile Communication and Healthcare*, ser. MOBIHEALTH'15. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2015, pp. 174–177. [Online]. Available: https://doi.org/10.4108/eai.14-10-2015.2261616

[85] A. Anzanpour, H. Rashid, A. M. Rahmani, A. Jantsch, N. Dutt, , and P. Liljeberg, "Energy-efficient and reliable wearable internet-of-things through fog-assisted dynamic goal management," in *Elsevier International Conference on Ambient Systems, Networks and Technologies (ANT19)*, Belgium, May 2019.

[86] H. Hoffmann, D. Wentzlaff, and A. Agarwal, "Remote store programming," in *HiPEAC*, 2010, pp. 3–17.

[87] Y. Shin, K. Shin, P. Kenkare, R. Kashyap, H.-J. Lee, D. Seo, B. Millar, Y. Kwon, R. Iyengar, M.-S. Kim, A. Chowdhury, S.-I. Bae, I. Hon, W. Jeong, A. Lindner, U. Cho, K. Hawkins, J. Son, and S. Hwang, "28nm high- metal-gate heterogeneous quad-core cpus for high-performance and energy-efficient mobile application processor," in *ISSCC*, 2013.

[88] T. Xu, L. Jin, X. Fan, Y. Zhou, S. Pasupathy, and R. Talwadker, "Hey, You Have Given Me Too Many Knobs! Understanding and Dealing with Over-Designed Configuration in System Software," in *Proceedings of the 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'15)*, Bergamo, Italy, Aug. 2015.

[89] J. Gray, "Why do computers stop and what can be done about it?" in *Symposium on Reliability in Distributed Software and Database Systems*, 1986.

[90] S. Wang, C. Li, H. Hoffmann, S. Lu, W. Sentosa, and A. I. Kistijantoro, "Understanding and auto-adjusting performance-sensitive configurations," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2018, Williamsburg, VA, USA, March 24-28, 2018*, 2018, pp. 154–168.

[91] Y. Ding, N. Mishra, and H. Hoffmann, "Generative and multi-phase learning for computer systems optimization," in *ISCA*, 2019.

[92] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," in *ASPLOS*, 2013.

[93] E. Ipek, O. Mutlu, J. F. Martinez, and R. Caruana, "Self-Optimizing Memory Controllers: A Reinforcement Learning Approach," in *ISCA*, 2008.

[94] A. J. Oliner, A. P. Iyer, I. Stoica, E. Lagerspetz, and S. Tarkoma, "Carat: Collaborative energy diagnosis for mobile devices," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '13. New York, NY, USA: ACM, 2013, pp. 10:1–10:14. [Online]. Available: http://doi.acm.org/10.1145/2517351.2517354

[95] D. D. Penney and L. Chen, "A survey of machine learning applied to computer architecture design," arXiv, Tech. Rep. 1909.12373v1, 2019.

[96] N. Mishra, H. Zhang, J. D. Lafferty, and H. Hoffmann, "A probabilistic graphical model-based approach for minimizing energy under performance constraints," in *ASPLOS*, 2015.

[97] I. Stoica, D. Song, R. A. Popa, D. Paerson, M. W. Mahoney, R. Katz, A. D. Joseph, M. Jordan, J. M. Hellerstein, J. Gonzalez, K. Goldberg, A. Ghodsi, D. Culler, and P. Abbee, "A berkeley view of systems challenges for ai," arXiv, Tech. Rep. 1712.05855v1, 2017.

[98] S. Tu and B. Recht, "Least-squares temporal difference learning for the linear quadratic regulator," arXiv, Tech. Rep. 1712.08642v1, 2017.

[99] S. Dean, H. Mania, N. Matni, B. Recht, and S. Tu, "On the sample complexity of the linear quadratic regulator," arXiv, Tech. Rep. 1710.01688v1, 2017.
[100] H. Hoffmann, "Jouleguard: energy guarantees for approximate applications," in *SOSP*, 2015.
[101] N. Mishra, C. Imes, J. D. Lafferty, and H. Hoffmann, "CALOREE: learning control for predictable latency and low energy," in *ASPLOS*, 2018.
[102] M. H. Santriaji and H. Hoffmann, "GRAPE: minimizing energy for GPU applications with performance requirements," in *MICRO*, 2016.
[103] Y. Zhou, H. Hoffmann, and D. Wentzlaff, "CASH: supporting iaas customers with a sub-core configurable architecture," in *ISCA*, 2016.