Architectural Aspects of Self-Aware and Self-Expressive Computing Systems: From Psychology to Engineering

Peter R. Lewis, Aston University
Arjun Chandra, Studix AS
Funmilade Faniyi, University of Birmingham
Kyrre Glette, University of Oslo
Tao Chen, University of Birmingham
Rami Bahsoon, University of Birmingham
Jim Torresen, University of Oslo
Xin Yao, University of Birmingham

Work on human self-awareness is the basis for a framework to develop computational systems that can adaptively manage complex dynamic tradeoffs at runtime. An architectural case study in cloud computing illustrates the framework's potential benefits.

Advanced computing systems generally contain many heterogeneous subsystems, each with a local perspective and goal set, which interconnect in changing network topologies. The subsystems must interact with each other and with humans in ways that are difficult to understand and predict while robustly maintaining performance, reliability, and security even with unforeseen dynamics, such as system failures or changing goals.

To meet these stringent requirements, computational systems—ranging from robot swarms and personal music devices to Web services and sensor networks—must achieve sophisticated autonomous behavior by adapting themselves at runtime and through learning processes that enable ongoing self-change. Managing tradeoffs among conflicting local and global goals at runtime requires considerable awareness of both the system's current state and its environment. Yet researchers have only recently begun to understand the implications of self-awareness principles and how to translate them into system engineering. Consequently, there is no general methodology for architecting self-aware systems or for comparing their self-awareness capabilities.

To address this need, we examined how human self-awareness can serve as a source of inspiration for a new notion of computational self-awareness and associated self-expression, and we developed a general framework for describing a computing system's self-awareness properties. As part of this work, we created a reference architecture, which we used to derive architectural patterns for

0018-9162/15/$31.00 © 2015 IEEE

MULTIDISCIPLINARY WORK ON SELF-AWARE SYSTEMS

A self-aware system has knowledge of itself and its experiences, permitting reasoning and intelligent decision making to support effective, autonomous adaptive behavior. Several research efforts relate self-awareness to computing, but reviews of this work[1–3] have clustered contributions either by community or theme to highlight implied meanings of the term.

AI AND ROBOTICS

AI researchers are concerned with metacognition—how a system can reason about its own reasoning and select a reasoning method appropriate to the situation at hand.[4] Researchers in autonomous robotics have argued that self-awareness is essential for safety and ethics.[3] Self-awareness is not only a property that is observable at an individual level, but is also something that can arise collectively. For example, a group of robots with simple behavioral rules and local interactions might arrive at an emergent awareness of a global state distributed across the individual units.[5]

SYSTEMS ENGINEERING

More practical efforts examine how to engineer systems that explicitly consider knowledge about themselves. One group argues that self-awareness can avoid or reduce the need to consider resource availability and constraints at design time.[6] However, proposed solutions are only in a specific application context, not a general framework.

AUTONOMIC COMPUTING

Autonomic computing targets the challenge of managing increasingly complex computing systems. Self-management solutions rely on the self-awareness property[7] and explicitly combine autonomic managers with a knowledge repository.

ORGANIC COMPUTING

Self-awareness is also a concept in organic computing, which aims to more deeply understand the emergent dynamics of large autonomous systems.[8] An important goal is to manage the behavior of such complex systems, leading to the concept of systems under observation and control.

References

1. P.R. Lewis et al., "A Survey of Self-Awareness and Its Application in Computing Systems," *Proc. 5th IEEE Int'l Conf. Self-Adaptive and Self-Organizing Systems Workshops* (SASOW 11), 2011, pp. 102–107.
2. J. Schaumeier, J. Pitt, and G. Cabri, "A Tripartite Analytic Framework for Characterizing Awareness and Self-Awareness in Autonomic Systems Research," *Proc. 6th IEEE Int'l Conf. Self-Adaptive and Self-Organizing Systems Workshops* (SASOW 12), 2012, pp. 157–162.
3. J. Pitt, ed., *The Computer after Me: Awareness and Self-Awareness in Autonomic Systems*, Imperial College Press, 2014.
4. M. Cox, "Metacognition in Computation: A Selected Research Review," *Artificial Intelligence*, vol. 169, no. 2, 2005, pp. 104–141.
5. M. Mitchell, "Self-Awareness and Control in Decentralized Systems," *Proc. AAAI Spring Symp. Metacognition in Computation*, 2005; www.cs.pdx.edu/~mm/self-awareness.pdf.
6. A. Agarwal et al., *Self-Aware Computing*, tech. report AFRL-RI-RS-TR-2009-161, MIT, 2009.
7. J.O. Kephart and D.M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, pp. 41–50.
8. C. Müller-Schloer, H. Schmeck, and T. Ungerer, *Organic Computing: A Paradigm Shift for Complex Systems*, Springer, 2011.

determining whether, how, and to what extent system engineers can build self-awareness capabilities into a system.

As the sidebar "Multidisciplinary Work on Self-Aware Systems" describes, various disciplines have devoted efforts to interpreting these self-awareness capabilities. Self-awareness concepts from psychology, for example, are inspiring new approaches for engineering computing systems that operate in complex, dynamic environments. Our framework builds on these ideas to understand the role of self-awareness in computing and to identify the potential system benefits of increased self-awareness.

Our psychology-grounded approach focuses on architectural aspects that are common to a variety of possible self-aware systems, building on layered control loops, which are typically part of self-adaptive, self-managing, and self-organizing systems. Examples include the monitor-analyze-plan-execute (MAPE) loop, which is often augmented with knowledge to form MAPE-K;[1] an observer/controller;[2] and a three-layer architecture for self-adaptive software systems[3] inspired by efforts in robotics and multiagent systems. Our framework differs from these architectures in two important ways:

- We translate concepts from psychology to engineering, presenting a reference architecture and derived architectural patterns that explicitly consider different self-awareness levels.
- Our framework does not presuppose that self-awareness is bolted on through an additional

management or control layer. Rather, it recognizes that engineers must consider the entire system and its environment when providing self-awareness capabilities.

We recently applied our reference architecture and self-awareness patterns, which we documented in a handbook,[4] to a range of applications, including active music devices, heterogeneous multicore systems, smart camera networks, and cloud computing systems. All of these share the characteristics of being large, decentralized, dynamic, uncertain, and heterogeneous, and all have benefited from the explicit consideration of self-awareness properties. Our cloud computing study, which uses one of our derived patterns, considers several of these properties.

HUMAN SELF-AWARENESS

Alain Morin defines self-awareness as "the capacity to become the object of one's own attention," which translates to your ability to consider yourself as an object.[5] Through this objective or explicit self-awareness, you focus attention on yourself as an entity within the world, observing and considering your own behavior and acquiring a public self-awareness—how the rest of the world might view that behavior.

Another facet of self-awareness is subjective or implicit. The self in this view is the subject (the "I") of experiences. You are aware of your experiences within the world and that these are subjective and unique experiences, private to you and typically not externally observable.

This distinction between public and private self-awareness is one of the three foundational principles we transfer to an engineering perspective in computational self-awareness. The public–private distinction underlines the need for systems to be concerned not only with knowledge of their internal aspects (state, capabilities, and so on), but also of their external experiences and their impact on and role within their physical and social environments.

A second foundational principle is the existence of various self-awareness levels—from basic stimulus awareness to meta–self-awareness, or an individual's awareness that it is self-aware.[5] Of the various psychological descriptions, Ulric Neisser's self-awareness levels capture the broadest range of human self-awareness.[6] This range is important, as the complexity of self-aware computing systems can vary considerably.

The third foundational principle is that self-awareness can be a property of collective systems, even when no single component has global awareness of the entire system.[7] In some cases, self-awareness might be considered an emergent property. Melanie Mitchell proposes that examples of this form of self-awareness include the brain, the immune system, and ant colonies.[7] In these systems, knowledge about global state is collected and maintained in a decentralized way, building up in a statistical fashion. This knowledge then feeds back to drive the adaptation of lower-level components. From an architectural perspective, this view of self-awareness as collective and emergent means that a self-aware system need not possess a global omniscient component.

COMPUTATIONAL SELF-AWARENESS

Our reference architecture captures the core aspects of computational self-awareness, providing a common, principled basis on which researchers and practitioners can structure their work. These core aspects are rooted in psychological foundations that might stimulate a range of ideas for engineers to consider in designing future computing systems. They serve as a template for identifying ways to implement self-awareness capabilities or the basis for comparing and evaluating architectures with the same capability.

Public and private self-awareness

Public and private self-awareness translates to a computing context in a fairly straightforward manner. Private self-awareness is the system's ability to obtain knowledge based on internal phenomena and requires internal sensors. Public self-awareness is the system's ability to obtain knowledge based on external phenomena. Such knowledge depends on how the system senses, observes, and measures its environmental aspects, including knowledge of its situation and context and its potential impact on and role within its environment.

Some work in self-aware computing considers only private self-awareness, but producing integrated conceptual models requires accounting for public aspects as well. The distinction, inclusion, and synthesis of public and private self-awareness facilitate the engineering of self-aware computing systems.

Self-awareness levels

Unlike public and private self-awareness, Neisser's human self-awareness levels require some mapping to align with an engineering perspective. Table 1 shows how our five levels of computational self-awareness correspond to

TABLE 1. Our framework's computational self-awareness levels versus Neisser's human self-awareness levels.

Our framework's levels	Our definition	Neisser's levels	Neisser's definition and example
Stimulus awareness	The system knows of the stimuli acting on it and can use that knowledge to respond to events.	Ecological self	Self as perceived with respect to the physical environment: I am the person in this place, engaged in this particular activity.
Interaction awareness	The system can learn that stimuli, and its own actions, constitute interactions with other systems and the environment.	Interpersonal self	Species-specific signals of emotional rapport and communication: I am the person who is engaged in this particular human interchange.
Time awareness	The system can obtain knowledge of historical and likely future phenomena.	Extended self	Based primarily on personal memories and anticipations: I am the person who had certain specific experiences, who regularly engages in certain specific and familiar routines.
Goal awareness	The system can obtain knowledge of current goals, objectives, preferences, and constraints.	Private self	Appears when an individual first notices that some experiences are not directly shared with others: I am, in principle, the only person who can feel this unique and particular pain.
Meta–self-awareness	The system can obtain knowledge of its own awareness levels and how they are exercised.	Conceptual self	Also referred to as self-concept, draws its meaning from the network of assumptions and theories in which it is embedded. Some theories concern social roles (husband, father), some postulate internal entities (soul, mental energy, brain), and some establish socially significant dimensions of difference (intelligence, attractiveness, wealth).

Neisser's five human self-awareness levels (http://the-mouse-trap.com/2009/11/01/five-kinds-of-selfself-knowledge). Our levels recognize that although self-knowledge is important to achieving computational self-awareness, the ability of the system to obtain this knowledge throughout its lifetime is the true self-awareness enabler. Thus, a system with knowledge but no way to update or add to it is not computationally self-aware, but rather the product of a domain expert's programming.

Some translations from Neisser's levels are straightforward; others are aimed at capturing the spirit of Neisser's definitions in terms that are more suitable for computing systems engineering. Translating concepts to the computing domain gives designers a common language for evaluating the need for self-awareness capabilities. In some cases, a system might have all five levels with several processes responsible for one or more levels; in others, one or two levels might be more appropriate.

Stimulus awareness. A stimulus-aware system knows of the stimuli acting on it and can use that knowledge to respond to events. At this level, the system has no knowledge of past or future stimuli—only current stimuli. Because stimuli can originate both internally and externally, stimulus awareness can be private, public, or both.

Interaction awareness. An interaction-aware system can learn that stimuli and its own actions constitute interactions with other systems and the environment. Through feedback loops, the system learns that its actions can provoke or cause specific reactions from its social or physical environments. Simple interaction awareness enables a system to reason about individual interactions, while a more advanced capability might involve obtaining knowledge of social structures, such as communities or networks.

Typically, interaction awareness is based on external phenomena, so it is a form of public self-awareness. However, a system that learns about causality in interactions with itself exhibits private self-awareness.

Time awareness. A time-aware system can obtain knowledge of historical and likely future phenomena. Implementing time awareness might involve having the system use explicit memory, time-series modeling, or anticipation. Because time awareness can apply to both internal and external phenomena, it can be private, public, or both.

Goal awareness. A goal-aware system can obtain knowledge of current goals, objectives, preferences, and constraints. Providing goal awareness is more than having implicit system design goals; rather, it is ensuring that the system has access to its goals and can reason about or manipulate them. Goal awareness can be achieved through a range of goal models, including state-based goal models—the system knows what might (or might not) be a goal state; or utility-based goal models—the system can learn a utility or objective function. A goal-aware system can adapt to goal changes, and if it is also interaction- and time-aware, the system might learn of others' goals or reason about likely future goals. Because goals can be private to the system or exist

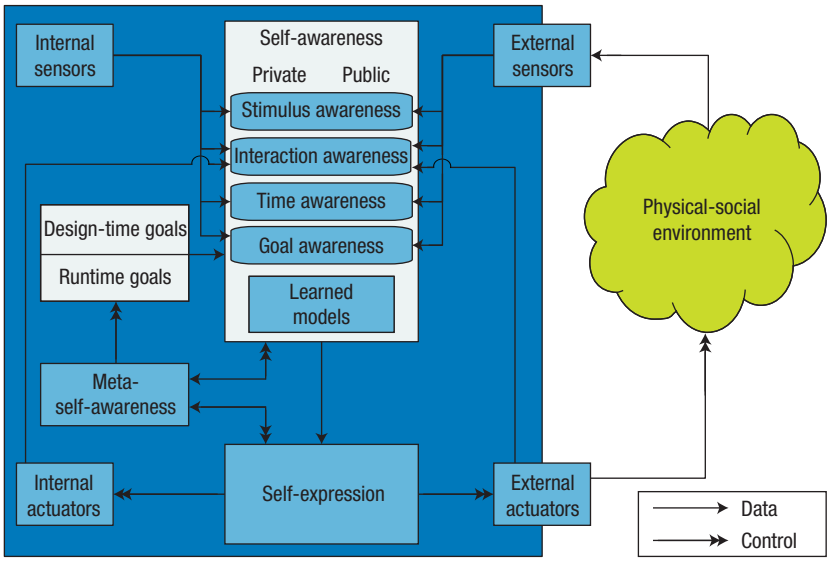

FIGURE 1. Reference architecture for self-aware and self-expressive computing systems. The architecture clarifies that computational self-awareness is a process or set of processes concerned not just with models of system knowledge but also with the ways in which the system updates that knowledge, such as through online learning.

collectively as a shared or externally imposed goal, goal awareness can be private, public, or both.

Meta–self-awareness. A meta–self-aware system can obtain knowledge of its own awareness levels and how they are exercised. Such awareness permits metacognitive processes to reason about the benefits and costs of maintaining a certain awareness level. A meta–self-aware system can adapt the way in which it realizes a self-awareness level, for example, by changing algorithms for realizing that level or by deciding whether or not to employ that level at all. Because meta–self-awareness is concerned only with internal process knowledge, it is a form of private self-awareness.

Collective and emergent self-awareness

Systems within a collective that interact with each other locally as part of a bigger system might not individually possess knowledge about the system as a whole. Although global knowledge is distributed, each system within the collective can work with other systems , giving rise to the collective itself obtaining a sense of its own state and thus being self-aware at one or more of the five self-awareness levels. For example, a decentralized system in which individual components learn about subgoals relevant to their own system role might exhibit a broader goal awareness at the global level. An example is recent work on distributed smart camera networks.[8] There is no global network view, yet self-awareness capabilities at the camera level give rise to a collective self-awareness that allows the effective and efficient management of tradeoffs across the network.

REFERENCE ARCHITECTURE

As these computational self-awareness principles highlight, self-awareness can be a property of an autonomous agent that can obtain and represent knowledge about itself and its experiences. Indeed, much of the literature on autonomous and intelligent agents is concerned with techniques for agent learning, knowledge acquisition and representation, and supporting architectures. Self-awareness can also be collective, so a self-aware entity is not limited to a single agent.

Our reference architecture unites these two ideas, underlining the notion that computational self-awareness is a process or set of processes concerned not only with the system knowledge captured in models, but also with the ways in which the system continually updates that knowledge, for example, through online learning. The system's goal knowledge also drives its self-expression—behavior based on self-awareness—and empowers it to use learned models in a variety of decisions. Such self-expressive systems can use various decision-making mechanisms for a given knowledge base.

Figure 1 shows our reference architecture's building blocks: internal and external sensors, internal and external actuators, and mechanisms to realize self-awareness and self-expression.

Internal and external sensors

Private and public self-awareness rely on continuous datastreams from sensors that observe phenomena on which to base self-awareness. Internal sensors measure aspects inside the system, such as temperature, battery sensors, or proprioceptive sensors attached to a robot's limbs. External sensors might include light sensors, cameras, and microphones.

Internal and external actuators

An interaction-aware system has privileged internal access to knowledge of its own actions. Internal actuators exercise a system's actions on itself, although sensors might need to observe the eventual outcome of those actions. For example, internal actuators could affect the system's energy consumption by throttling CPU speed or adjusting a camera's zoom level. External actuators exercise actions between the system and its environment, such as through radio transmitters or loudspeakers.

Self-awareness mechanisms

The computational process that realizes each self-awareness capability analyzes

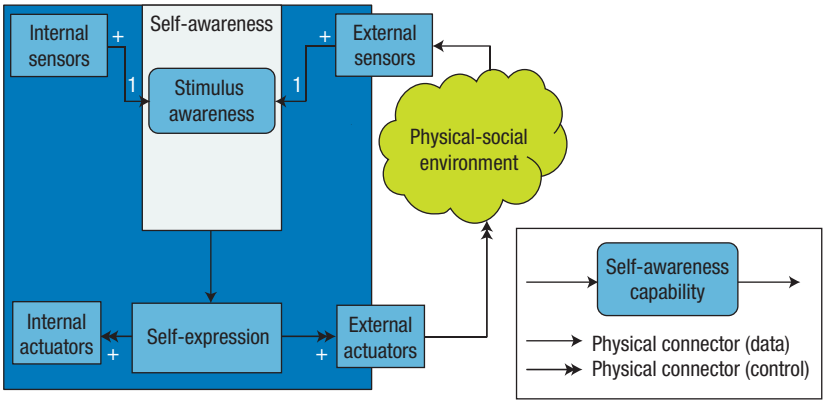

FIGURE 2. Basic architectural pattern. The operator next to the sensor and actuator boxes indicates the permitted number of such capabilities in an interaction: + is one or more and 1 is exactly one.

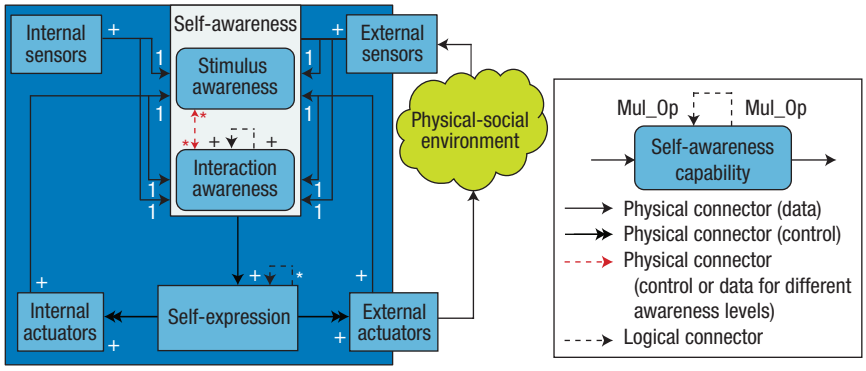

FIGURE 3. Coordinated decision-making pattern, which adds interaction awareness to the basic pattern. The operator next to the self-awareness capability and logical connector (Mul_Op) indicates the permitted number of such capabilities in an interaction: * is zero or more, + is one or more, and 1 is exactly one.

sensor data to produce subjective models of the internal or external phenomena that the system must consider. With goal awareness, a system can obtain both design and runtime goals, which various self-awareness levels use to construct models that further affect system actions.

At the stimulus-awareness level, the system might receive input from neighboring systems. At the interaction-awareness level, a process could build a neighborhood map. Incorporating time awareness could add communication history to this map, which could be used for future communication decisions. The goal-awareness level could, for example, monitor a goal of sensory coverage based on sensing and network topography. A meta–self-awareness process could monitor the performance of the model-building algorithms in the current environment, performing algorithm selection to switch between sensing strategies.

Meta–self-awareness plays a key role in managing the set of goals a system works with during its lifetime. Different operational environments or internal states can require the system to change focus from one goal to another. The meta–self-awareness component enables the system to perceive the tradeoff between various goals, given feedback from the environments and states that stem from the system's actions. This allows a system to continuously monitor its behavior in terms of these goals.

Self-expression mechanisms

A system uses the knowledge obtained through self-awareness processes, including knowledge about goals, when deciding to take a particular action. The results of the self-expression processes are commands for the internal or external actuators. Because the self-expression component involves decision-making processes, a clear separation between this component and self-awareness can help designers evaluate process possibilities.

A self-expression process builds on knowledge from self-awareness processes, and could, for example, choose to increase sensor range on the basis of knowledge about progress toward some goal.

DESIGN PATTERNS

Developing a system with computational self-awareness requires architectural and design guidelines. To meet this need, we have developed a handbook of eight architectural patterns for self-aware systems, each with a different set of capabilities.[4] Figures 2 through 4 show three of these patterns. The same handbook also provides a systematic pattern-selection method that uses a set of questions to help designers identify the problem-specific requirements relative to each self-awareness level.

In addition to the physical connectors from the reference architecture, we use logical connectors in the patterns to express intracapability interactions and intercapability interactions—those of the same capability across different self-aware nodes. These logical connectors do not require direct physical interaction. For example, self-expression processes across several self-aware systems might logically reach consensus, but such interaction is physically realized through sensors, actuators, and interaction awareness. An additional physical connector (red arrows in Figures 3 and 4) makes

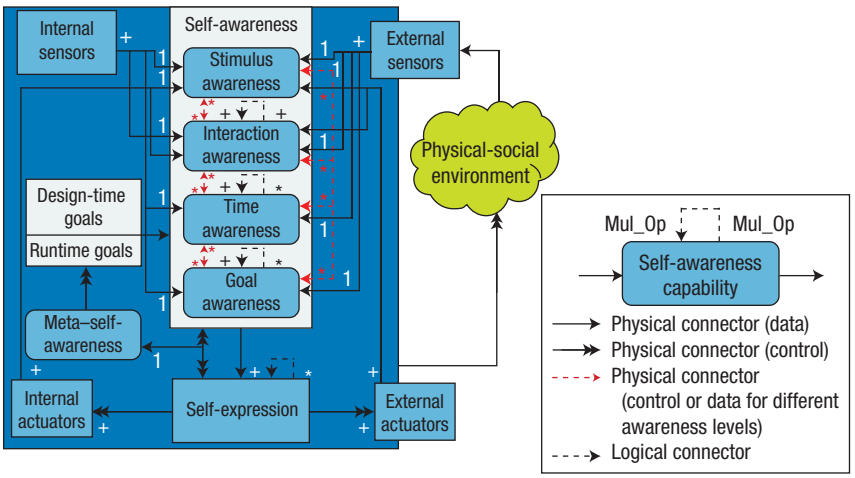

FIGURE 4. The fully self-aware pattern, which adds goal, time, and meta–self-awareness to the coordinated decision-making pattern. The multiplicity operator (Mul_Op) next to the self-awareness capability indicates the permitted number of such capabilities in an interaction: * is zero or more, + is one or more, and 1 is exactly one.

intercapability interactions within a node explicit and identifies physical interactions among awareness levels.

Capabilities are conceptual and need not map strictly to components, so we use a multiplicity operator (Mul_Op) to represent the number of capabilities involved in the interaction.

Basic

Figure 2 shows the basic pattern, which contains only the stimulus-awareness capability. Actions can be triggered on the basis of the detected stimulus type. This is the simplest pattern and enables minimal computational self-awareness.

Coordinated decision making

Figure 3 shows the coordinated decision-making pattern, which enables the coordination of multiple self-aware nodes. With the addition of interaction awareness to the basic pattern, a node's peers can share knowledge of their interactions, including causality and social relationship models. Thus, the nodes can independently adapt their actions to common knowledge and link self-expressive processes so they can agree on which actions to take and coordinate them.

Fully self-aware

The fully self-aware pattern, shown in Figure 4, adds goal, time, and meta–self-awareness to the coordinated decision-making pattern. Goal awareness enables the representation of changing runtime goals, so a node can share its knowledge with other goal-aware nodes and the system can adapt to the changed goals. Time awareness allows the representation of temporal knowledge about goal and interaction awareness, enabling capabilities such as forecasting.

This pattern also adds meta–self-awareness, enabling the system to manage the tradeoffs associated with exercising various self-awareness levels and thereby allowing it to modify goals at runtime. An example of this runtime metareasoning is the dynamic selection of the most appropriate learning algorithm for a particular context.

CASE STUDY

To evaluate the benefits of architectural patterns for creating self-aware systems, we conducted a case study in cloud computing to assess the ability of one of our derived design patterns to address the services-selection problem.

Services selection

Service-based applications are composed of abstract services that concrete services with functional equivalence instantiate at runtime. Applications must satisfy their quality-of-service (QoS) requirements, which typically differ widely across applications. Each application has only a local perspective of its goal, which can change over time, for example, because of the need to respect new service-level agreements (SLAs). Applications share a cloud services pool, offering various QoS levels at fluctuating prices. Applications compete for services in a shared infrastructure and often interfere with one another.

Each application must manage its resources on the basis of its knowledge of changing goals, workload, and service availability. Computational self-awareness provides primitives for modeling this knowledge through a multilevel approach that enables a separation of concerns and simplifies runtime tradeoff analyses.

Goal sharing with time awareness

After following the pattern-selection process in our handbook,[4] we identified the goal sharing with time awareness pattern as the most suitable for a system that could address the services-selection problem. The pattern combines stimulus, interaction, time, and goal awareness, and provides self-expression:

- *Stimulus awareness*. Sensors directly observe user requests categorized in SLA classes (low, medium, high).
- *Time awareness*. A local performance repository stores service ratings of how well the service met the promised QoS.
- *Interaction awareness*. The performance repository also captures knowledge about the interaction between services and users clustered in different SLA classes.

- *Goal awareness*. Given a particular QoS, a utility function deduces the candidate services likely to provide the best quality.
- *Self-expression*. Service-selection decisions are based on allocation strategies.

Performance results and comparison

In our system analysis and comparison, we used the architectural analysis and tradeoff method (ATAM),[9] a validated scenario-based evaluation method that qualitatively evaluates software architectures to reveal the risks associated with architectural decisions. Using ATAM, we compared the performance of our system, designed with the goal sharing with time awareness pattern, against the performance of a system designed using the three-layer architecture.[3] Although comparing our system with the MAPE-K and observer/controller reference architectures might have been interesting, we chose to focus on the three-layer architecture as a point of comparison because, like our framework, it separates knowledge concerns and has key comparison points such as explicit goal representation. The other two reference architectures do not capture knowledge concerns at a fine-grained level.

Overall, our system more effectively handled the dynamism and uncertainty in application goals, workload, and service availability.[10] An important criterion in architectural evaluation is the identification of tradeoff points, which makes design decisions about quality attributes explicit. In our case study, adaptability was a first-class quality attribute, as we believe it is central to system scalability, availability, and performance. Overall, we identified three tradeoff points with this attribute: communication load, scalability cost, and service selection accuracy (versus only two tradeoff points in the three-layer architecture). Failing to account for these points would pose a risk to our system's ability to meet its quality attribute goals.

Our evaluation further revealed that explicitly considering interaction among knowledge concerns (self-awareness) potentially reduces the risks that might otherwise arise. In contrast, the three-layer architecture simply flagged the consequences of such interactions as risks. We also found that communication costs from data exchanges between decentralized self-aware nodes might be higher in our system than those in the three-layer architecture, which shares knowledge components. However, our system's adaptability trades off well with communication cost and adaptive decision accuracy.

For example, explicitly considering computational self-awareness through the use of the architectural patterns ensured that we considered a broad range of possible self-awareness capabilities and included only relevant capabilities justified by identified benefits. Additionally, decomposing a self-aware system into self-awareness, self-expression, and meta–self-awareness processes made implementation substantially easier and reduced the possibility of introducing faults, thus facilitating fault detection.

In dealing with the complexities of future computing systems—size, decentralization, uncertainty, dynamics, and heterogeneity—higher self-awareness levels will become critical. Our notion of computational self-awareness and associated self-expression can provide computing systems with advanced levels of autonomous behavior to enable runtime self-adaption and management of complex tradeoffs in rapidly changing conditions. Using Neisser's broad set of self-awareness levels, designers can explicitly account for a full spectrum of existing and future systems and not be concerned only with what is now considered highly advanced AI.

By providing a reference architecture and architectural patterns for specific systems, we have given designers a common way to communicate about self-awareness and self-expression capabilities from an engineering perspective. However, much is still unknown about how to incorporate self-awareness properties into computing systems. One open problem, which we plan to address, is how systems can learn and adapt to changing conditions at runtime, making decisions on the basis of knowledge about tradeoffs among and between system goals as well as the overhead from the learning process.

Addressing other open problems will require more than one view of self-awareness computing. Significant progress will require a multidisciplinary effort, drawing not only from psychology but also from philosophy, sociology, economics, AI, and engineering.

ACKNOWLEDGMENTS

This work was partially supported by the EU FP7 program as part of the EPiCS project, under grant agreement 257906 (www.epics-project.eu).

REFERENCES

1. J.O. Kephart and D.M. Chess, "The

ABOUT THE AUTHORS

PETER R. LEWIS is a lecturer in computer science at the Aston Laboratory for Intelligent Collectives Engineering at Aston University, UK. His research interests include adaptation, online learning, and self-organization in complex agent-based systems with a focus on nature-inspired techniques, heterogeneity, and self-awareness. Lewis received a PhD in computer science from the University of Birmingham, UK. He is a member of IEEE. Contact him at p.lewis@aston.ac.uk.

ARJUN CHANDRA is a researcher at Studix AS, Norway. His research interests include orchestrating systemwide outcomes in computationally intelligent agent collectives, including those engaging humans. Chandra received a PhD in computer science from the University of Birmingham. Contact him at arjun@studix.com.

FUNMILADE FANIYI is a software engineering researcher at the University of Birmingham. His research interests include designing enterprise software applications for industrial stakeholders, software architectures for large-scale self-adaptive systems, and cloud computing. Faniyi received a PhD in computer science from the University of Birmingham. Contact him at f.faniyi@gmail.com.

KYRRE GLETTE is an associate professor of computer science at the University of Oslo, Norway. His research interests include intelligent, adaptive, and biologically inspired systems for embedded and runtime-evolvable hardware systems and evolutionary robotics. Glette received a PhD in computer science from the University of Oslo. He is a member of IEEE. Contact him at kyrrehg@ifi.uio.no.

TAO CHEN is a doctoral candidate and researcher in the School of Computer Science at the University of Birmingham. His research interests include performance modeling and tuning, self-adaptive systems, services computing, and cloud computing. Chen received an MSc in computer science from the University of Birmingham. He is a student member of IEEE. Contact him at txc919@cs.bham.ac.uk.

RAMI BAHSOON is a senior lecturer in software engineering and leads the Software Engineering for/in the Cloud interest group in the School of Computer Science at the University of Birmingham. His research interests include self-adaptive software architecture and service-, cloud-, and economics-driven software engineering. Bahsoon received a PhD in software engineering from University College London, UK. He is a member of ACM. Contact him at r.bahsoon@cs.bham.ac.uk.

JIM TORRESEN is a professor of computer science at the University of Oslo. His research interests include applying nature-inspired computing, adaptive systems, reconfigurable hardware, and robotics to complex real-world applications. Torresen received a Dr.Ing. from the Norwegian University of Science and Technology. He is a member of IEEE. Contact him at jimtoer@ifi.uio.no.

XIN YAO is a professor of computer science at the University of Birmingham. His research interests include evolutionary computation, ensemble learning, self-adaptive systems, search-based software engineering, and real-world applications. Yao received a PhD in computer science from the University of Science and Technology of China. He is an IEEE Fellow and president of the IEEE Computational Intelligence Society. Contact him at x.yao@cs.bham.ac.uk.

Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, 2003, pp. 41–50.
2. C. Müller-Schloer, H. Schmeck, and T. Ungerer, *Organic Computing: A Paradigm Shift for Complex Systems*, Springer, 2011.
3. J. Kramer and J. Magee, "Self-Managed Systems: An Architectural Challenge," *Proc. IEEE Conf. Future of Software Eng.* (FOSE 07), 2007, pp. 259–268.
4. T. Chen et al., *The Handbook of Engineering Self-Aware and Self-Expressive Systems*, tech. report, EPiCS EU FP7 Project Consortium, 2014; http://arxiv.org/abs/1409.1793.
5. A. Morin, "Levels of Consciousness and Self-Awareness: A Comparison and Integration of Various Neurocognitive Views," *Consciousness and Cognition*, vol. 15, no. 2, 2006, pp. 358–71.
6. U. Neisser, "The Roots of Self-Knowledge: Perceiving Self, It, and Thou," *Annals of the New York Academy of Sciences*, vol. 818, 1997, pp. 19–33.
7. M. Mitchell, "Self-Awareness and Control in Decentralized Systems," *Proc. AAAI Spring Symp. Metacognition in Computation*, 2005; www.cs.pdx.edu/~mm/self-awareness.pdf.
8. B. Rinner et al., "Self-Aware and Self-Expressive Camera Networks," *Computer*, vol. 48, no. 7, pp. 21–28.
9. R. Kazman et al., "Experience with Performing Architecture Tradeoff Analysis," *Proc. 21st ACM Int'l Conf. Software Eng.* (ICSE 99), 1999, pp. 54–63.
10. F. Faniyi et al., "Architecting Self-Aware Software Systems," *Proc. Working IEEE/IFIP Conf. Software Architecture* (WICSA 14), 2014, pp. 91–94.