Architecting Self-aware Software Systems

Funmilade Faniyi
School of Computer Science
University of Birmingham
Birmingham B15 2TT, UK
fof861@cs.bham.ac.uk

Peter R. Lewis
School of Engineering and Applied Science
Aston University
Birmingham B4 7ET, UK
p.lewis@aston.ac.uk

Rami Bahsoon, Xin Yao
School of Computer Science
University of Birmingham
Birmingham B15 2TT, UK
{r.bahsoon—x.yao}@cs.bham.ac.uk

***Abstract*—Contemporary software systems are becoming increasingly large, heterogeneous, and decentralised. They operate in dynamic environments and their architectures exhibit complex trade-offs across dimensions of goals, time, and interaction, which emerges internally from the systems and externally from their environment. This gives rise to the vision of self-aware architecture, where design decisions and execution strategies for these concerns are dynamically analysed and seamlessly managed at run-time. Drawing on the concept of self-awareness from psychology, this paper extends the foundation of software architecture styles for self-adaptive systems to arrive at a new principled approach for architecting self-aware systems. We demonstrate the added value and applicability of the approach in the context of service provisioning to cloud-reliant service-based applications.**

Keywords*—*Self-adaptation, Architecture style, Self-awareness

I. Introduction

As users of software systems find more sophisticated use for computing capabilities, the ensuing complexities of modern software systems are manifold. These systems are generally large scale and require managing heterogeneous and time-varying objectives. The autonomic computing vision [1] was initiated in response to these challenges, with the goal of making computing systems self-managing. The intended benefit being a reduction in the administrative cost and burden of controlling complex software systems.

Architecture-based self-adaptation [2] [3] has been recognised as one of the prominent ways of designing so-called autonomic systems. In the architecture approach, the system to be *managed* is endowed with a *managing* system, which typically consists of an adaptive engine equipped with sensors for monitoring and effectors for impacting the system. The adaptation engine is composed of architectural models for reasoning about adaptation actions in response to sensed data from the system and environment. Adhering to principles of architecture styles that abstract common features of architecture instances in a specific domain [4] is known to serve as a useful guide to the architect when designing software systems.

A handful of architecture styles has been contributed in line with the vision of architecture-based self-adaptation. These approaches often make simplified assumptions when modelling and managing possible trade-offs encountered in dynamic, open systems. As a result, the quality of self-adaptation tends to be limited as it does not fully capture complex trade-offs arising from heterogeneity of the interacting nodes, the operating scale, openness and dynamism of the environment. We have observed that "fine-grained" representation of knowledge when coupled with a multi-level online learning framework can provide the necessary primitives for more reliable and efficient self-adaptation [5] [6]. Inspired by the concept of self-awareness from psychology, we characterise the primitives of knowledge representation required to architect self-adaptive systems.

System self-awareness has long been recognised as an enabler for advanced autonomic behaviour [1]. To better reason about self-awareness in technical systems, concepts from psychology and cognitive science have been reinterpreted in a computational context [6]. Accordingly we define a *self-aware* computational node as one that "possesses information about its internal state and has sufficient knowledge of its environment to determine how it is perceived by other parts of the system" [6].

Our self-aware architecture style builds on the primitives of knowledge representation, adaptation of knowledge, and provides extensible support for online learning at multiple levels of abstraction. It adheres to tested architectural principles such as separation of concern, and extends them to improve architectural analysis. Drawing on a case study of adaptive resource allocation to cloud-reliant service-based applications, we motivate the need for enriching the capabilities of self-adaptive architectures with computational self-awareness. We contribute to an approach for architecting self-adaptive software systems using the principles of self-awareness, leading to the self-aware architecture style. Scenarios from the case study are used to demonstrate the applicability of the approach.

The rest of the paper is structured as follows. Section II motivates the case study. The self-aware style and its underlying primitives are presented in sections III-V. Section VI illustrates the applicability of the approach. Related work are discussed in section VII. Section VIII concludes the paper.

II. Motivating Example: Adaptive Cloud Application

We consider service-based applications (SBAs) deployed in a cloud environment. Every SBA is composed of *abstract services*, which are instantiated by functionally equivalent *concrete services* at run-time. Each SBA has the goal of satisfying its QoS requirements. These goals are not the same for every SBA. Additionally, the architecture of each SBA differs in topology and its constituent abstract services. Each SBA has only a local perspective of its goal, which may change over time, for example due to a need to respect new SLAs. SBAs share a common pool of cloud services. Cloud services offer various QoSes at prices which may change with time,

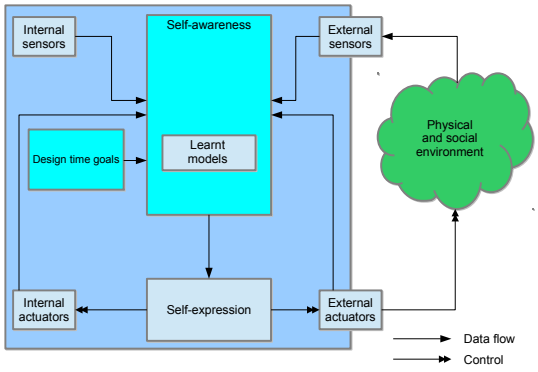

Fig. 1. Overview of Self-Aware Architecture Style

for example due to over subscription from multiple competing applications. Externally, SBAs interact via the shared service infrastructure and possibly interfere with one another when competing for services. Consequently, there is a tension between each application's local goal satisfaction and the (global) objective of fair resource distribution.

In order to manage application dynamics, each SBA is endowed with an adaptive component. The objective of the component is to manage the pool of available services on behalf of the SBA and ensure its QoS goals are met. Thus the component makes service selection decisions. It also reacts to the time-varying workload patterns by making on-demand request for services and releasing them in an elastic manner to avoid under/over-provision. Moreover, the adaptive components owned by different SBAs autonomously coordinate and resolve conflicts among themselves to ensure their goals are achieved whilst respecting the global objective.

To realise this self-adaptive capability, the software architecture of the adaptive component should cater for fine-grained representation of knowledge pertaining to changing goals, workload, and service availability. The self-aware style offers primitives for modelling this knowledge using an approach that simplifies run-time trade-off analyses.

III. Overview of Self-Aware Architecture Style

We describe an architecture-style for self-awareness by looking at a self-aware node. In this context, a *node* need not be a specific physical system, but instead provides a conceptual container for the system being considered: the element in that context which is being referred to as *self*. A node could therefore be, for example, an autonomous agent, a running thread, a physical machine, or a collective of these. Importantly, the node represents a level of abstraction at which knowledge acquisition, representation and behaviour determination occurs.

Figure 1 depicts the internal composition of a constituent node in the said style. It describes their structure, interaction, and relationship with the environment. The style describes the primitives for knowledge acquisition, representation, and behavioural processes for self-adaptation. It introduces a multi-level approach that capture concerns related to goals, interaction, and time when analysing and reasoning about self-adaptation and the emergent trade-offs.

Due to the unpredictability associated with both deployment environments and the dynamics within them, one key challenge in realising self-awareness and self-expression in computing systems is the appropriate use of effective online learning schemes. In the self-aware architectural style, online learning algorithms instantiate two conceptual components in order to provide adaptive knowledge acquisition and behaviour in the self-awareness component: (i) sensor data is collected, analysed and, if appropriate, knowledge obtained from it is represented in learnt models (ii) behavioural learning (e.g. action selection and strategy selection) takes place, informed by knowledge present as part of the node's self-awareness.

Self-awareness processes are able to collect information both from internal sensors (regarding private experiences internal to the node and typically externally unobservable) and external sensors (regarding experiences of the node's physical environment as well as of other nodes). Additionally, self-awareness processes are able to observe the actions taken by the node, and have access to goals specified for the node at design time.

Self-expression processes make use of knowledge obtained and represented by self-awareness processes and determine appropriate actions as a result. The self-expression component therefore has control over actuators. The self-expression component has no privileged direct access to the design-time goals, however in a typical instantiation, a self-awareness process will be responsible for representing goal information in a meaningful, useful and efficient manner (e.g. through a utility function), to the self-expression component. In this way, though a node may be designed with multiple complex and context dependent goals, it may possess the ability to be aware of which goals are relevant given its current context, and expose only those to the self-expression component at a given time. This separation can act to simplify the required self-expression behaviour.

IV. Self-Awareness and Types of Knowledge

Lewis et al. [6] highlighted three key aspects of the psychological literature on self-awareness: 1) the notions of public and private self-awareness, concerned with external and internal sources of knowledge respectively, 2) that self-awareness is not a binary property, but there exist various levels of self-awareness, corresponding to the capability to represent and reason about various types and complexities of knowledge, and 3) that self-awareness can be a property of a collective system as well as a single system. Since in this paper we are concerned with the architecture of a single system, we do not address the third aspect. In this section however, we draw attention to the first and second.

Firstly, the architecture style deals with the concept of public and private self-awareness [6], by specifying both external and internal sources of data, from which the self-awareness component constructs knowledge. Data connectors clearly establish this relationship.

Secondly, we have found it particularly useful to develop a novel computational interpretation of the levels of self-awareness introduced by Neisser [7]. Neisser's levels, discussed in a computational context in [6], describe increasingly complexity in terms of an individual's self-awareness

capabilities. Building on this, we have developed five levels of *computational self-awareness*, which in a similar way, can be used to describe a computing system's self-awareness capabilities. Our five levels of computational self-awareness (see figure 2) follow, along with their relevance to either public or private self-awareness or both.

Stimulus-aware: A node is stimulus-aware if it has knowledge of stimuli. The node is not able to distinguish between the sources of stimuli. It does not have knowledge of past/future stimuli. It enables the ability in a node to respond to events. It is a prerequisite for all other levels of awareness. Since stimuli may originate both internally and externally, stimulus-awareness can be both **private** and **public**.

Interaction-aware: A node is interaction-aware if it has knowledge that stimuli and its own actions form part of interactions with other nodes and the environment. It has knowledge via feedback loops that its actions can provoke, generate or cause specific reactions from the social or physical environment. It enables a node to distinguish between other nodes and environments. Interaction-awareness is typically based on external phenomena and is therefore a form of **public** self-awareness, however one can also envisage a system which learns about the effects of internal interactions with itself, which would constitute a form of **private** self-awareness.

Time-aware: A node is time-aware if it has knowledge of historical and/or likely future phenomena. Implementing time-awareness may involve the node possessing an explicit memory, capabilities of time series modelling and/or anticipation. Since time-awareness can apply to both internal and external phenomena, it can either be both **private** and **public**.

Goal-aware: A node is goal-aware if it has knowledge of current goals, objectives, preferences and constraints. It is important to note that there is a difference between a goal existing implicitly in the design of a node, and the node having knowledge of that goal in such a way that it can reason about it. The former does not describe goal-awareness; the latter does. Example implementations of such knowledge in a node include state based goals and utility based goals. Since goals may exist privately to the node, or collectively as a shared or externally imposed goal, goal-awareness can be both **private** and **public**.

V. Reasoning about Adaptation Actions via Meta-Self-Awareness

The most advanced of the levels of self-awareness, the fifth level, concerns *meta-self-awareness*, an awareness of ones own self-awareness capabilities (or lack thereof). Therefore, in a self-aware system, learning can occur not only at the adaptation level, but also at the meta level [5]. Online learning at the meta level, as shown in figure 2, occurs in a meta-self-awareness component, where models of the node's own behaviour are built online, and acted upon.

As an example of the role of the meta-self-awareness component, consider that at the adaptation level, it would be possible for various instances of (possibly the same or different) online learning algorithms to instantiate several conceptual components, to achieve different purposes simultaneously. For example, a node may instantiate a complex form of time-awareness and a more simplistic form of goal-awareness. But the node's changing run-time context may require a change to a more complex form of goal-awareness to achieve sufficient adaptivity. Conversely, the node may become aware (at the meta level) that the overhead associated with performing highly complex time series modelling is wasteful, when considered in a trade-off with the benefits arising from such modelling. In this case, it may be beneficial for the node to cease such modelling.

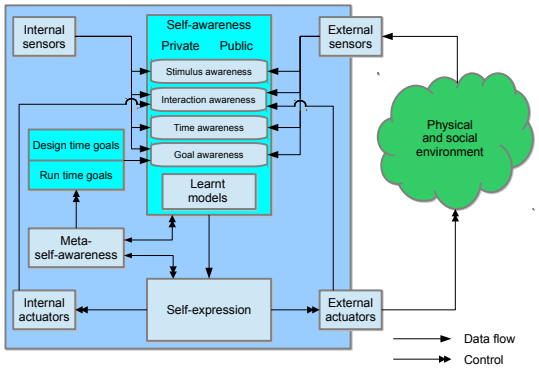

Fig. 2. Levels of Self-Awareness

In general, we anticipate firstly that algorithms will be selected for each component and tuned according to their role in a particular application, and secondly that this algorithm selection and tuning will be managed during run-time by a node's meta-self-aware capability, as context changes.

VI. Applicability of the Approach

In the scenario described in section II, SBAs are allocated a budget based on their wiliness to compete in the service market. Likewise, cloud services charge a time-varying fee. Cloud service and application QoS are represented by a tuple consisting of performance, availability etc.

The external sensor firstly compares application's required QoS with service provider's offered QoS. Stimulus-aware component characterises the dynamic change in user requests. This model is derived from workload generated in response to user request. User requests are distinguished by SLA classes. The goal-aware component makes use of an utility function (see equation 1) to deduce the candidate services which are likely to provide optimal QoS, where w_b is initialised based on the application SLA class. k is a sensitivity factor for tuning the affinity of the application for demanded services. β_{price} is the price demanded by the service provider at the time the service is requested.

$$U(service) = w_b + (k * \beta_{price}) \quad (1)$$

Time-aware component makes use of a locally managed performance repository to store rating of services, in terms of the level to which the service met promised QoS, and the duration of their use. Interaction-aware component captures knowledge about the interaction between services and users clustered in different SLA classes using the performance repository. The objective is to ensure that the local objectives of

the interacting applications are met, taking into consideration the various SLA classes.

Self-Expression component makes service selection decisions based on allocation strategies. Two example strategies are (a) *Strategy 1:* choose service with the lowest price possible. That is, the selling price must be the lowest among available services. If more than one service offers the lowest price, then one is chosen at random; (b) *Strategy 2:* choose services at random, provided the price is acceptable, i.e., price less or equals application's budget. Depending on the strategy selected, there exist a trade-off between the time to select a service and the cost of the service. More details about these strategies and how the resulting trade-off spaces can be managed using market-based computational mechanisms can be found in [8][9].

VII. Related Work

MAPE-K [1] is a widely adopted autonomic architecture style. The (K)nowledge component is shared by the (M)onitor, (A)nalyser, (P)lanner and (E)xecutor components. MAPE-K provides primitives for encoding experts' knowledge about a domain in K. This knowledge is used to reason about run-time adaptation. Our style takes the view of architecting interacting computing systems (self-aware nodes), which are deployed in settings where knowledge about the correct adaptation is *a priori* unknown. MAPE-K models knowledge at a coarse grain, without explicit distinction between knowledge concerns for goals, time, or interaction.

SEEC is another framework that claims self-aware capabilities [10]. SEEC relies on the (O)bserve-(D)ecide-(A)ct (ODA) [10] architecture style. The O and A components in ODA are equivalent to M and E components in MAPE-K respectively, while analysis and planning tasks are subsumed in the Decide component. Our self-aware style decouples the decider (D) into two smaller subunits: learnt knowledge (self-awareness) and decider (self-expression). We may therefore describe our style as an Observe-Learn-Decide-Act (OLDA) architecture style, since the self-awareness component provides primitives for learning. The meta-self-awareness variant of our style is one in which an additional Monitor/Controller acts as an observer to the Learning (L) and Decision (D) processes in the OLDA. Similar to MAPE-K and ODA styles, the self-aware style emphasises a separation or decoupling between the acquisition and representation of the learnt knowledge and the decision making process. However, the knowledge representation component in our style explicitly models goals, time, and interaction concerns of the system-to-be.

Our work complements learning-inspired architecture styles. FUSION, a framework for tuning self-adaptive software system at run-time, was proposed by [11]. FUSION uses feature-based approach and online learning for analysis and adaptation. Whereas [11] takes the view of a centrally managed self-adaptive system, our style is not limited to centralised systems, and is able to cope with the heterogeneity of interacting nodes at a fine-grain. Additionally, [11] promotes the use of reinforcement learning, we provide an extensible framework, where the choice of learning mechanism is adaptively determined by the meta-self-aware component based on trade-offs for goals, time, and interaction.

VIII. Conclusion

We have described an approach for architecting self-adaptive software systems using the principles of self-awareness. A distinctive feature of the style is that it elaborates the knowledge representation at a fine-grained level to handle complex trade-offs across the dimensions of goals, time, and interaction. The style offers extensible support for various learning approaches, while ensuring coherence between the three key activities of self-awareness, self-expression and meta-self-awareness.

We plan to contribute to a catalogue of architectural patterns which architects can exploit to inform the design decisions for self-awareness and measure the added value of partial/full instantiation of its primitives in an application. We will also support the catalogue with metrics for assessing the quality of adaptation on dimensions related to accuracy, improved adaptability, dependability, and incurred overhead.

Acknowledgment

This research was supported by the EPiCS project and received funding from the European Union Seventh Framework Programme under grant agreement n° 257906.

References

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

[2] P. Oreizy, M. Gorlick, R. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, and A. Wolf, "An architecture-based approach to self-adaptive software," *Intelligent Systems and their Applications, IEEE*, vol. 14, no. 3, pp. 54–62, 1999.

[3] B. H. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic *et al.*, *Software engineering for self-adaptive systems: A research roadmap*. Springer, 2009.

[4] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *SIGSOFT Softw. Eng. Notes*, vol. 17, no. 4, pp. 40–52, Oct 1992.

[5] T. Becker, A. Agne, P. Lewis, R. Bahsoon, F. Faniyi, L. Esterle, A. Keller, A. Chandra, A. Jensenius, and S. Stilkerich, "EPiCS: Engineering proprioception in computing systems," in *Proc. of the 15th IEEE International Conf. on Computational Science and Engineering (CSE)*, 2012, pp. 353–360.

[6] P. R. Lewis, A. Chandra, S. Parsons, E. Robinson, K. Glette, R. Bahsoon, J. Torresen, and X. Yao, "A survey of self-awareness and its application in computing systems," in *Proc. Int. Conf. on Self-Adaptive and Self-Organizing Systems Workshops (SASOW)*, 2011, pp. 102–107.

[7] U. Neisser, "The roots of self-knowledge: Perceiving self, it, and thou," *Annals of the NY AoS.*, vol. 818, pp. 19–33, 1997.

[8] P. R. Lewis, F. Faniyi, R. Bahsoon, and X. Yao, "Markets and clouds: Adaptive and resilient computational resource allocation inspired by economics," in *Adaptive, Dynamic, and Resilient Systems*, N. Suri and G. Cabri, Eds. Taylor & Francis, 2013.

[9] F. Faniyi and R. Bahsoon, "Economics-driven software architecting for cloud," in *Economics-driven Software Architecture*, I. Mistrik, R. Bahsoon, R. Kazman, K. Sullivan, and Y. Zhang, Eds. Elsevier, 2013.

[10] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, and A. Agarwal, "SEEC: A framework for self-aware computing," 2010.

[11] A. Elkhodary, N. Esfahani, and S. Malek, "Fusion: a framework for engineering self-tuning self-adaptive software systems," in *Proc. of the eighteenth ACM SIGSOFT Int. symposium on Foundations of software engineering*, ser. FSE '10, 2010, pp. 7–16.