Attaining Meta-self-awareness through Assessment of Quality-of-Knowledge

Abdessalam Elhabbash*, Rami Bahsoon†, Peter Tino†, Peter R. Lewis‡, Yehia Elkhatib§
* *DS-Lab, School of Computing and Communications*, *Lancaster University*, Lancaster, United Kingdom
† *School of Computer Science*, *University of Birmingham*, Birmingham, United Kingdom
‡ *Faculty of Business & IT*, *Ontario Tech University*, Oshawa, Ontario, Canada
§ *School of Computing Science*, *University of Glasgow*, Glasgow, United Kingdom
{a.elhabbash, r.bahsoon, p.tino, peter.lewis, yehia.elkhatib}@lancaster.ac.uk*,cs.bham.ac.uk†,ontariotechu.ca‡,glasgow.ac.uk§

***Abstract*—Self-awareness is a crucial capability of autonomous service-based systems that enables them to self-adapt. There are different types of self-awareness whereby certain types of knowledge are captured at various levels. We argue that effective management of the trade-offs of dependability requirements can be achieved through "seamless" switching between different levels of awareness. However, the assessment of the quality of knowledge to enable dynamic switching between self-awareness levels has not been tackled yet. We propose a general architecture that exploits symbiotic simulation in order to tackle the complexity of assessing the quality of knowledge and attaining the meta-self-awareness property, wherein the system can reflect on its different levels of awareness. We conduct a thorough real-world study in the context of volunteer services. We conclude that a system made meta-self-aware using our approach achieves optimal performance by activating the most suitable awareness level. This comes at the cost of a modest computational overhead.**

***Index Terms*—Quality of Knowledge, Meta-self-awareness, Self-adaptation, Self-awareness, Symbiotic Simulation**

I. Introduction

Modern software systems are difficult to maintain. They span potentially millions of lines of code and hundreds of microservices (*e.g.,* [1], [2]). They are deployed in dynamic environments with uncertainty associated with their decision making relating to code errors [3], [4], complex configuration [5], [6], and unpredictable changes in user behaviour [7] and other operational parameters [8], [9]. This is especially evident in environments with high complexity and multiple stakeholders such as datacenter applications, cyberphysical systems, and volunteer services (VS).

As such, adaptation at runtime is unavoidable. Complexity and dynamism call for more effective self-adaptive approaches that enable complex software systems to maintain their goals in the face of heterogeneity, uncertainty, and emerging behaviour. Specifically, complex systems need to be able to gain sufficient runtime awareness to be able to plan and implement appropriate means of adaptation effectively. Such autonomous qualities rely not only on acquiring knowledge that is actionable and tractable, but also on doing so in a software-intensive manner.

Self-awareness and self-expression properties, have received increasing attention in software engineering for self-adaptive and managed systems [10], [11]. They have been used in the context of engineering large scale systems to enrich self-adaptation capabilities and manage dynamic tradeoffs at runtime. Examples of large scale environments that benefited from self-awareness are numerous [12], [13], [14], [11].

There has been a number of concerted efforts advocating computational self-awareness. Examples include the projects EPiCS [15], SEEC [16], ASCENS [17], and the SEAMS Dagstuhl Seminars [18], among others. Although those efforts presented different approaches to engineer self-awareness, the common trait is that self-awareness is fundamentally dependent on acquiring and representing knowledge on the system internal state and its environment. The acquired knowledge will then inform the adaptation decision making. Some of the approaches pay more attention to knowledge and treat it at a fine-grained level where different types of knowledge are considered *e.g.,* [12]. However, they introduce conceptual architectures that lack concrete realisation of how systems would self-adapt using different types of knowledge, an issue which we address in this paper.

However, when knowledge availability is limited, performing online adaptation (*e.g.,* proactive) based on "best bet" or "trial and error" becomes unavoidable; which is obviously an expensive and risky exercise. Therefore, one of the challenges facing self-aware systems is the assessment of the *quality of knowledge* (QoN) that informs self-adaptation. This encapsulates key knowledge properties such as: recency, decay, sufficiency, and suitability to the adaptation capabilities. Systems should be able to acquire insights on the quality of decision making when utilising the acquired knowledge and also on the way it is acquiring and representing knowledge. Such challenges call for approaches for QoN assessment. Intuitively, this can be achieved by continuous and active evaluation of knowledge via its actual use in the decision making process. However, this can also result in expensive and inefficient adaptation decisions in case QoN is not at a reasonable level. Therefore, we argue that approaches for the assessment of QoN before leveraging the knowledge into the adaptation process are demanded.

In this research, we marry concepts from the foundations of autonomic computing and software engineering in order to assess QoN for self-adaptive software-intensive systems. From the field of autonomic computing, we identify the concept of

meta-self-awareness [15], [12] *i.e.,* the state of a system when it is aware of being aware, as being instrumental in enabling a system to reflect on its different levels of awareness and respond according to the changes in its context. To attain this, we employ the software engineering paradigm of *symbiotic simulation* [19] in which a simulation system and a real system interact to benefit each other. Data collected from the real system is used to simulate cases and obtain insights that are fed back to the real system to inform the the process of decision making [20].

In this sense, we propose an architecture that characterises varying levels of self-awareness that enable representing different types of knowledge and hence supporting different adaptation strategies. A symbiotic simulator is employed to use data collected from the real system to perform *what-if* experiments to assess the benefits and overhead of adopting each strategy. The assessment is based on multiple criteria that are used to evaluate the system goals. Consequently, an adaptation strategy is suggested to the real system using a multi-criteria decision making (MCDM) technique [21].

Specifically, the novel contributions of this paper are:

1) A general architecture to achieve meta-self-awareness intended to serve as a reflective control layer to dynamically switch between different awareness levels. The architecture makes novel use of a symbiotic simulation – in the heart of the self-adaptive process – to predict the performance of self-adaptation using different self-awareness levels.
2) A general quantitative approach for self-adaptation at the meta level that enables managing the trade-off between system goals and overheads by employing a MCDM technique.
3) A case study evaluation of our meta-self-awareness approach based on real experiments with volunteer computing systems. We compare our approach against versions of the framework that are self-aware but not meta-self-aware.

To develop and evaluate our work, we draw on a case from volunteer service composition (VSC) [22] as an environment that is characterised by openness, heterogeneity, scale and unpredictability. The results show the benefits of self-adaptation using the meta-self-awareness approach resulting in achieving optimal performance of the system in terms of requests satisfaction and resource utilisation.

II. Background on Symbiotic Simulation

For its central role in our framework, we present some preliminaries on symbiotic simulation. Inspired by symbiotic interactions between biological species, the notion of symbiotic simulation systems (SSS) was introduced by the National Science Foundation (NSF)[1] and at the Dagstuhl Seminar on *Grand Challenges for Modelling and Simulation* [24]. The Dagstuhl Seminar defined SSS as "*one that interacts with the physical system in a mutually beneficial way*". As such, both the physical and simulation parts of a SSS benefit each other. The simulation receives real data collected by the physical system, which is necessary to run scenarios related to the decision-making process carried out by the physical part. The simulation is carried out by performing multiple what-if experiments to evaluate alternatives. The physical part uses the simulation results to inform the decision-making process and optimise its performance.

In [25], [26] this definition is extended to involve other types of relationships between the two systems, which were also inspired by biological symbiosis. The extension introduced other types of SSS that may or may not be mutually beneficial. In these types, the symbiotic simulation receives real data from physical systems and may send feedback accordingly. The physical system may then apply a decision based on this feedback or use it as guidance. In general, there are two types of symbiotic simulations: closed-loop and open-loop systems. In a closed-loop system, feedback is created, analysed, and utilised to make a decision to control the physical system. Such feedback can be used in one of two forms: a *decision* that will be applied directly to the physical system; or a *suggestion* that may or may not be applied. On the other hand, in the open-loop type, no feedback will be communicated to the physical system. The feedback will instead be used by external operators or tools for specific purposes, *e.g.,* visualisation. In our approach, we use closed-loop symbiotic simulation to provide *suggestions* to the real system.

III. System Architecture

This section outlines the architecture of a meta-self-aware system. The architecture enables systems to acquire different types of knowledge and to define different adaptation capabilities. It then allows systems to adapt between those capabilities at runtime. Fig. 1 depicts the architecture, which consists of the following main components:

Service Repository. This is a repository that stores the metadata of the services published by service providers.

Internal/External Sensors. The sensors collect data about the events that happen internally in the system (*e.g.,* the arrival of requests and the state of the queue) and externally in the

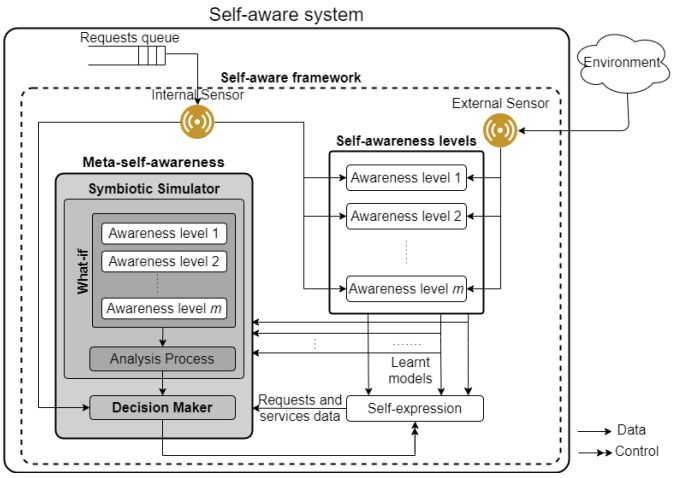

Fig. 1: The conceptual architecture of the meta-self-aware system.

[1] See [23] and http://www.dddas.org/NSFworkshop2000.html

environment (*e.g.,* the performance and offering of external services). The data are passed to the self-awareness levels for knowledge extraction.

Request Queue. User requests are received in a priority queue. The internal sensor fetches the request from the queue and passes it to the self-expression component. The queuing model can be viewed as an M/M/1 model (according to Kendall's notation [25]) where:

- Request arrival follows a Poisson process of rate λ.
- Serving time (the time required to find a composite service that satisfies the request) has an exponential distribution with a serving rate μ, where $1/\mu$ is the average serving time.
- There is one server that generates the composite services.
- The sequences of the requests inter-arrival times and the serving times are independent.

Self-awareness Levels. The architecture assumes the presence of multiple levels of self-awareness, each realized by at least one algorithm that extracts and represents a certain type of knowledge from the collected data. Correspondingly, each level supports an adaptation capability. These capabilities are cumulative over the self-awareness levels where the knowledge at lower levels is included in higher ones. For example, the lowest level can be to extract basic knowledge about the occurrence of change events such as events related to service availability (*i.e.,* stimulus-awareness). Higher levels can be to extract knowledge about historical performance (*i.e.,* time-awareness) which also includes the basic knowledge of event occurrence captured at the lower level.

Self-expression. Once a request arrives, this component executes the selection algorithm to identify services that satisfy the request (*i.e.,* the execution plan). If a request violation is reported, the self-expression component re-executes the selection algorithm to repair the execution plan. In either case, the self-expression component uses the learned models passed from the self-awareness levels.

Meta-self-awareness. This component represents an extra level of self-awareness that acts as the reflective control level of the self-aware system that assesses the benefits and the overhead of each level. It allows the framework to adapt the adoption of the above levels of self-awareness, by activating or deactivating them based on the system state and self-knowledge. The learned models and information on the system state (in terms of the queue size) are passed to this component to allow it to decide whether the activation of a certain self-awareness level is expected to be beneficial or not. To do so, the component is equipped with a symbiotic simulator that performs *what-if* experiments using the real learned models to assess the quality of the collected knowledge, in terms of a set of defined criteria (*e.g.,* percentage of satisfied requests) as explained in section IV-C. In this sense, the meta-self-awareness level permits the system to improve its performance (*e.g.,* by leveraging the captured knowledge for selecting dependable services). Furthermore, it permits the system to degrade smoothly instead of failing fatally (*e.g.,* when the overhead of leveraging the knowledge is unbearable). Meta-self-awareness contains two main sub-components:

- ***Symbiotic Simulator*** simulates the processing of the received requests (*i.e.,* the selection and adaptation decisions) which has been performed by the self-expression component. The idea is that self-expression has used models learned from one of the self-awareness levels. However, the benefits and overheads of using models that have been learned by the other levels of awareness are not observed. For this purpose, the symbiotic simulator performs what-if experiments to evaluate alternative selection and adaptation decisions that could have been taken using each of the self-awareness levels. The results of the what-if experiments are passed to the *Analysis Process* component which analyses the predicted quality of each of the self-awareness levels (that reflects the QoN). The *Analysis Process* suggests the self-awareness level with the best predicted quality to the *Decision Maker* component.
- ***Decision Maker*** takes as input the suggestion of self-awareness level and the learned models about the internal system state (*i.e.,* the state of the system queue). Subsequently, it makes a decision whether to adapt the system to the suggested self-awareness level or not. This decision is based on the expected benefits (*e.g.,* better performance) and overheads (*e.g.,* longer queuing time) of adopting the suggested self-awareness level. The adaptation is performed by notifying the self-expression component.

IV. Meta-Self-Awareness with Symbiotic Simulation

We now detail the mechanism of QoN assessment carried out by the meta-self-awareness level and the procedure of adaptation at the meta-level.

A. Adaptation checkpoints

The decision to switch from one self-awareness level to another is based on the periodic evaluation of system performance. The evaluation is performed every time period T_p. The upper bounds of the time periods are called the checkpoints. The length of T_p varies from $[T_{min}, T_{max}]$ and is updated at run-time according to the changes in the system state. In order to update T_p, we adopt a strategy similar to the one used in [27]. After each evaluation period, if the evaluation resulted in no transition to a new awareness level, then T_p is increased to $min(2.T_p, T_{max})$. Conversely, if the evaluation resulted in a transition to a new level, then T_p is decreased to $max(T_p/2, T_{min})$.

B. Queue stability

The queue stability can be a measure of the performance of the system [28]. According to Loynes' theorem [29], if the request arrival and serving processes are independent, then the queue is stable if the request arrival rate λ is less than the serving rate μ. This condition is necessary to avoid indefinite growth of the queue as that will increase the waiting time and the percentage of dropped requests, ultimately resulting in lower performance of the system. Thus, the meta-self-awareness should tend to choose the self-awareness level such

that queue stability is achieved. We use the queue idolisation factor $\rho = \lambda/\mu$ and require $\rho \leq 1$ for the queue to be stable.

C. Symbiotic simulation decision support

At every checkpoint, the system passes the set of requests that have been processed by the self-expression component and the models learned by the different self-awareness levels. Then the *What-if* simulator simulates the processing of the requests using the real learned models. The simulations evaluate the self-awareness levels in terms of a set of criteria $C = \{c_1, c_2, ..., c_n\}$ that are the same criteria used to specify the system goals. For example, if the system goals are to optimise for resources utilisation and waiting time, then the what-if simulations are performed to evaluate the utilisation and waiting time that could have been achieved using the different self-awareness levels.

Upon receiving the what-if simulation results at every checkpoint, the *Analysis Process* applies a simple additive weighing method [30] to express the benefit of each of the awareness levels. Formally, at each checkpoint, the *Analysis Process* performs the following calculation steps:

Step 1. The criteria values form a matrix, D, which has the following form:

$$D = \begin{array}{c} \\ level_1 \\ level_2 \\ \vdots \\ level_m \end{array} \begin{array}{c} \begin{array}{cccc} c_1 & c_2 & \dots & c_n \end{array} \\ \begin{pmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ r_{21} & r_{22} & \dots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m1} & r_{m2} & \dots & r_{mn} \end{pmatrix} \end{array} \tag{1}$$

where each r_{ij} is a measurement of the performance of the corresponding awareness level under the given criteria.

Step 2. Since some criteria can be negative criteria (*i.e.,* the higher the value the lower the benefit *e.g.,* the waiting time and the resources waste) and others are positive (*i.e.,* the higher the value the higher the benefit *e.g.,* throughput) and since each criteria has its own range and units, the metrics should be scaled to make them comparable. The scaled matrix, N, has the form:

$$N = \begin{array}{c} \\ level_1 \\ level_2 \\ \vdots \\ level_m \end{array} \begin{array}{c} \begin{array}{cccc} c_1 & c_2 & \dots & c_n \end{array} \\ \begin{pmatrix} \eta_{11} & \eta_{12} & \dots & \eta_{1n} \\ \eta_{21} & \eta_{22} & \dots & \eta_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \eta_{m1} & \eta_{m2} & \dots & \eta_{mn} \end{pmatrix} \end{array} \tag{2}$$

where $\eta_{ij} = \frac{r_{ij} - {r_j}^{min}}{{r_j}^{max} - {r_j}^{min}}$ for positive criteria, $\eta_{ij} = \frac{{r_j}^{max} - r_{ij}}{{r_j}^{max} - {r_j}^{min}}$ for negative criteria, ${r_j}^{max}$ is the maximal value of a criterion in matrix D, and ${r_j}^{min}$ is the minimal value of a criterion in matrix D.

Step 3. The importance of each of the identified metrics, as specified by the system admin, is expressed through the multiplication of matrix N by the weight vector W resulting in a weighted matrix V:

$$V = \begin{pmatrix} \eta_{11} & \eta_{12} & \dots & \eta_{13} \\ \eta_{21} & \eta_{22} & \dots & \eta_{23} \\ \vdots & \vdots & \ddots & \vdots \\ \eta_{31} & \eta_{32} & \dots & \eta_{33} \end{pmatrix} \begin{pmatrix} w_{c_1} & 0 & \dots & 0 \\ 0 & w_{c_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & w_{c_n} \end{pmatrix} \tag{3}$$

where $w_{c_i} \geq 0$ and $\sum w_{c_i} = 1$.

Step 4. The overall benefit of each level is computed by adding the corresponding row elements of V resulting in a vector of the quality of each self-awareness level $\hat{Q_{L_i}}$.

$$\hat{Q} = \begin{array}{c} \\ level_1 \\ level_2 \\ \vdots \\ level_m \end{array} \begin{pmatrix} \hat{Q_{L_1}} \\ \hat{Q_{L_2}} \\ \vdots \\ \hat{Q_{L_m}} \end{pmatrix} \tag{4}$$

Step 5. The expected stability of the queue is calculated for each self-awareness level using:

$$\rho_i = \frac{k}{\sum_{r=1}^{k} \mu_{ri}} \tag{5}$$

where k is the number of the simulated requests, μ_{ri} is the simulation time of serving the request r, and ρ_i is the expected stability of the queue assuming the self-awareness level i is adopted. This results in the vector $\hat{S}$.

$$\hat{S} = \begin{array}{c} \\ level_1 \\ level_2 \\ \vdots \\ level_m \end{array} \begin{pmatrix} \rho_{L_1} \\ \rho_{L_2} \\ \vdots \\ \rho_{L_m} \end{pmatrix} \tag{6}$$

Step 6. The overall expected benefit B_{L_i} of each self-awareness level L_i is calculated as the product of the expected quality $\hat{Q_{L_i}}$ and the expected stability ρ_{L_i} where $B_{L_1} = \hat{Q_{L_i}} \times \rho_{L_i}$. This results in the vector $\hat{B}$.

$$\hat{B} = \begin{array}{c} \\ level_1 \\ level_2 \\ \vdots \\ level_m \end{array} \begin{pmatrix} B_{L_1} \\ B_{L_2} \\ \vdots \\ B_{L_m} \end{pmatrix} \tag{7}$$

Step 7. The self-awareness level with the maximum expected benefit will be selected by the meta-self-awareness level and passed to the *Decision Make*r as a suggestion.

D. Adaptation at the meta-level

The *Decision Maker* component performs the actual adaptation at the meta-self-awareness level. The *Decision Maker* makes the decision of adaptation based on the suggested level of awareness and the current state of the queue stability. Algorithm 1 depicts the pseudo-code of the selection function performed by the *Decision Maker*. Recalling that the higher the level of awareness the longer the serving time, then adapting to a higher level of awareness is implausible if the queue is not stable. Therefore, the *Decision Maker* will refuse the suggested

Algorithm 1: Decision Maker Algorithm

Function ***selectAwarenessLevel*** *currentLevel* L_i, *suggestedLevel* L_j, *QStability* ρ

```
if j > i AND ρ > 1 then
    return L_i;
return L_j;
```

level of awareness if the transition is suggested to a higher level of awareness and the queue is currently unstable.

The selected level will be passed to the *self-expression* component which will use the corresponding knowledge to serve the user requests.

V. Case Study: Volunteer Service Composition

Volunteer computing is a paradigm that is characterised by heterogeneity, dynamism, and uncertainty. Volunteers make their idle computing and/or storage available for some application to use, which provides massive resources for complex applications. However, a volunteer can withdraw their resources at anytime, which triggers replacement of those resources. Such adaptation can utilise different types of awareness about the offered resources. These characteristics make volunteer computing a typical environment to demonstrate the feasibility of our proposed approach. Therefore, this section evaluates our proposed meta-self-awareness approach through a case study of volunteer storage system (VSS) where volunteer storage resources are offered as services and composition of those services is required to satisfy user needs.

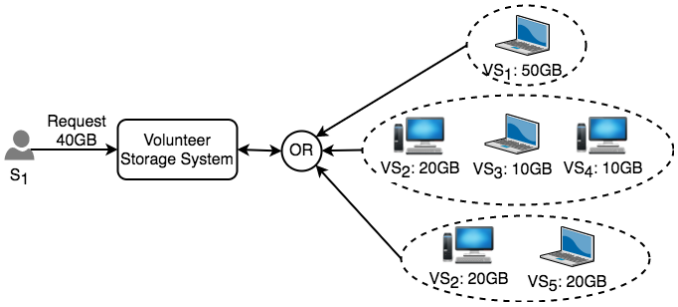

Fig. 2: Composition request of S_1 for a storage service.

A. Motivating scenario

We motivate the need for meta-self-awareness and its engineering principles by looking at VSC. The case assumes a heterogeneous and dynamic environment that consists of varied computing nodes like PCs, laptops, smartphones, etc. connected via a network. Individual people owning these nodes, known as publishers, offer their idle storage resources as services using a publish/subscribe model. A subscriber can explore the network and search for volunteer storage services to use. If she finds the required storage, while satisfying her requirements (*e.g.,* location, security, etc.), she will request it for her use. Otherwise, volunteered storages can be composed together to form a service that meets the subscriber's needs. Fig. 2 shows an example in which the subscriber S_1 submits a request to search for storage of capacity 40 GB. To make this available to S_1, the system inspects the published storages and returns three possible composition strategies:

First: Using the storage promised by VS_1.
Second: Composing the storages of VS_2,VS_3, and VS_4.
Third: Composing the storages of VS_2 and VS_5.

There are different ways of satisfying such request. One possibility is to randomly pick any of them and when one of the services involved in the selected strategy violates the requirements, the system initiates an adaptation action to repair the strategy. However, a question arises here about the feasibility of that adaptation action, *i.e.,* will the undertaken adaptation result in better performance? On the other hand, if the system is able to anticipate the performance of the services, it can select a strategy so that violations are less likely to occur. The deeper the knowledge the system has on the performance of the services, the more intelligent the selection decision will be (*e.g.,* by avoiding the selection of services that exhibited poor performance in the past). Here, self-awareness can be adopted to reason about the self-adaptation actions; enabling intelligent selection and adaptation decisions. For example, assume that S_1 submitted a request at time t_1, and assume that the performance of VS_5 is anticipated to be poor at t_1, then the system will avoid the selection of the third strategy. But, if we assume that S_1 submitted a request at time t_2 and that the performance of VS_1 is anticipated to be good at t_2, then the system will select the third strategy. In addition, the reason for the existence of different strategies can be the use of different self-aware approaches to composing the services. The more intelligent the approach, the more appropriate services are selected. However, the advantages are often accompanied by overheads. Here, meta-self-awareness can be applied to switch between the different approaches of service selection, based on the benefits and overheads of each approach.

B. Volunteer storage system

To realize the VSS, a web-service based system has been developed. On the one hand, the system aggregates the storage from the volunteer end devices. On the other hand, it offers the aggregated storage in a service-oriented perspective as composite services; based on the subscriber requirements. Fig. 3 depicts the architecture of the VSS which consists of the following parts, *Volunteer Software*, *Subscriber frontend*, and the *Self-aware manager*.

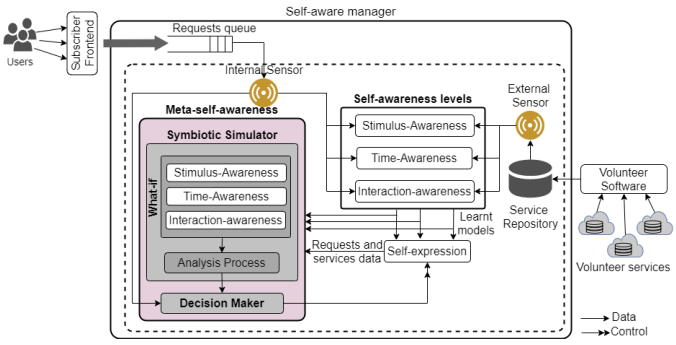

Fig. 3: Architecture of the Volunteer Storage System.

1) Volunteer Software: The software enables volunteers to submit their VS information specifically, the volunteered storage space stg, the availability period $T = [a, b]$ where a is the start date and b is the end date, and the binding information required t access the physical storage location. The software periodically sends an "*I am alive*" heartbeat during the availability period to confirm the availability to the self-aware manager. The VS is implemented as a RESTful web service which exposes the file manipulation operations. The storage content is delivered/retrieved directly to/from the volunteer machine without routing through the server hosting the central manager. The VS is implemented as a RESTful web service which exposes the file manipulation operations.

2) Subscriber Frontend: Subscribers use the frontend to submit their requests. A request is expressed in terms of the required storage space stg^R and the required availability period $T^R = [a^R, b^R]$. The VSS processes each request in the queue to find a service or a composite service that satisfies it. Furthermore, the frontend enables sending feedback on each VS to the VSS. The feedback information involves:

- Reporting any violation in the promised storage space, which is done automatically by the frontend.
- Reporting the VS reputation level $Rep\,(VS_i)$, which is done manually by the subscriber. The reputation level has the value of 2 if no violation has been experienced, 1 if any minor data loss/corruption occurs, or 0 if a major data loss/corruption occurs. The representative reputation of a certain service VS_i is the average of the subscribers' feedback on it.

3) Utility-based service selection: The self-expression component performs the service selection and composition in order to satisfy the requests. We use a utility model to provide a systematic approach for volunteer service selection by measuring the amount of contribution that each service exhibits to satisfy the request and greedily selecting the highest-contributing services. The model is used to compute the storage and availability time utilities for each service using the utility functions 8 and 9, respectively. We refer to these as the promised utilities.

$$U_{Stg}(VS_i) = \begin{cases} e^{-\beta(Stg_i - Stg^R)}, & \text{if } Stg_i \geq Stg^R \\ e^{\alpha(Stg_i - Stg^R)}, & \text{if } Stg_i < Stg^R \end{cases} \tag{8}$$

where $0 < \beta < \alpha < 1$.

$$U_{Time}(VS_i) = \begin{cases} 0, & \text{if } b_i \leq a^R \text{or } b^R \leq a_i \\ e^{\gamma(b_i - b^R)}, & \text{if } a_i < a^R \text{and } a^R < b_i < b^R \\ e^{\gamma(a^R - a_i)}, & \text{if } b_i > b^R \text{and } a^R < a_i < b^R \\ \frac{e^{\alpha(b_i - a_i)}}{e^{\alpha(b^R - a^R)}}, & \text{if } a_i \geq a^R \text{and } b_i \leq b^R \\ \frac{e^{-\beta(b_i - a_i)}}{e^{-\beta(b^R - a^R)}}, & \text{otherwise} \end{cases} \tag{9}$$

where $0 < \beta < \gamma < \alpha < 1$.

The system also computes the average reputation of the each service $Rep\,(VS_i)$, and finds the non-dominant set of services using (10) and randomly selects one of them.

$$\textit{maximise } (U_{Stg}(VS_i), U_{Time}(VS_i), Rep(VS_i)) \tag{10}$$

4) The Queue: Requests are selected from the queue according to the *Smallest-size Job First* (SJF) policy. The request size is defined as:

$$size \;=\; stg^R \;\times\; \left|T^R\right|$$

where stg^R is the required storage in megabytes $\left|T^R\right|$ is the length of the required time interval in hours.

5) Self-awareness levels: In this case study, the self-aware manager provides three levels of self-awareness in addition to the meta-self-awareness level: stimulus-, time-, and interaction-awareness.

Stimulus-awareness provides the basic knowledge on changes that occur internally (*e.g.,* a service violated the requirements) and externally (*e.g.,* a new service has been offered in the service repository). This knowledge supports the ability to adapt the composite service. When this level is activated, the self-expression responds to the notifications received from the stimulus-awareness level to adapt the composite services by re-executing the utility-model-based selection.

Time-awareness assumes the presence of stimulus-awareness and adds more self-awareness by considering the historical performance of the services to select dependable ones. The approach uses dynamic histograms (DHs), which are constructs that dynamically approximate data distributions in evolving datasets [31], such as service performance, and capture their variability including seasonal changes. When a request is submitted, DHs are used to predict service performance. However, this advantage is hampered by a computational overhead due to computations required to maintain the DHs. Obviously, this overhead results in longer waiting times for the requests in the queue to be processed.

Interaction-awareness extends the time-awareness level by considering the historical interactions between the services in pairs, *i.e.,* 2^{nd}-order relations. In other words, this level captures knowledge on the historical performance of each service when it is composed with other services. So, services that had good performance when composed together in the past will have a high chance of being composed together again. In order to realise this, we maintain a DH for each pair of services, as introduced in [32] where this approach shown to satisfy a higher number of requests compared to other forms of self-awareness. Nevertheless, it is accompanied with some overhead and faster queue growth.

To summarise, the stimulus-aware approach enables the basic and least informed adaptation with least overhead. More added self-awareness brings more advantages at the price of higher overhead and larger queue size.

C. Meta-self-awareness for VSC

At every checkpoint, the system passes the set of requests that have been processed by the real system and the models learned by the stimulus-, time- and interaction-awareness levels. Then the *What-if* simulator simulates the processing of

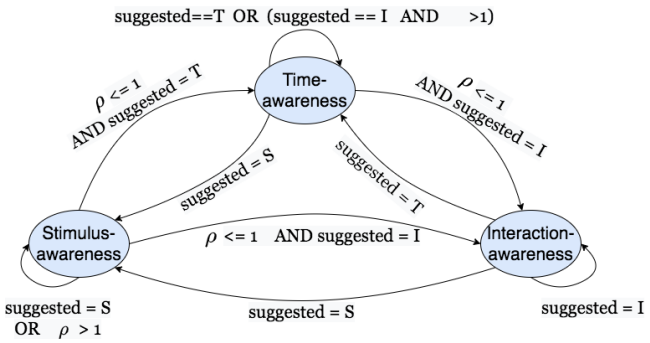

Fig. 4: The state transition diagram of awareness levels.

requests using the real learned models in order to investigate the alternative decisions which could have been made using the other awareness levels. The simulation results are passed to the *Analysis Process* in order to evaluate the expected performance of the different awareness levels. The simulation results are summarised in terms of the following criteria:

- Percentage of Satisfied Requests (PSR): the number of requests that the system could successfully satisfy divided by the total number of requests. This criterion is directly related to the efficiency of selecting services; selecting dependable services will lead to fewer violations and more satisfied requests.
- Resource Waste (RW): the average storage space over-provisioned to satisfy a request divided by the total storage space allocated to that request. Minimising RW corresponds to maximising resources utilisation.
- Waiting Time (WT): the average time that a subscriber needs to wait until the system finds or adapts a composite service for her request.

Upon receiving the simulation results at a checkpoint, the *Analysis Process* applies eq 1-7 to select a self-awareness level and suggest it to the *Decision Maker* component. The *Decision Maker* takes the final decision based on queue stability.

Fig. 4 summarises the transitions that the VSS may perform where $suggested = S$, $= T$, *and* $= I$ denote the suggestion of the stimulus-, time-, or interaction-awareness levels, respectively. The transitions are the following:

- *The current state is stimulus-awareness:*
  - If $(\rho > 1)$, then the queue is unstable (*i.e.,* size is increasing). Therefore, no transition is performed in order to process the request as fast as possible using the stimulus-awareness level.
  - If $((\rho \leq 1)\ AND\ (suggested == S))$, then although the queue is stable, no transition is performed because no benefit is expected from the other levels of awareness.
  - If $((\rho \leq 1)\ AND\ ((suggested == T)\ OR\ (suggested == I)))$, then the system performs a transition to the suggested awareness level.
- *The current state is time-awareness:* If $(\rho > 1\ AND\ suggested == I)$, then the queue is unstable and hence the *Decision Maker* refuses the suggestion as that will worsen the stability of the queue. Therefore, no transition is performed. Otherwise, the suggestion will be approved.
- *The current state is interaction-awareness:* The *Decision Maker* will approve the suggested self-awareness level as that will at least benefit the queue stability.

TABLE I: Ranges of attribute values

Service and subscriber attributes		
Attribute	**Service**	**Subscriber**
Storage	[5 - 20]	[20 - 30]
Time	[1 Jan. 00:00 - 31 Dec. 23:59]	[1 Jan. 00:00 - 31 Dec. 23:59]
Reputation	[0 - 2]	[0 - 2]
Meta-self-awareness attributes		
Percentage of Satisfied Requests (W_{PSR})	0.4	
Resources Waste weight (W_{RW})	0.3	
Waiting Time weight (W_{WT})	0.3	

D. Experimental evaluation

We conduct experiments to evaluate the performance of meta-self-awareness for the VSC case study. The experiments were conducted on a PC with Intel Core i5-3570 3.5 GHz, 4GB RAM, Windows 7 and Java SE 1.7.0.

1) Context: We implement the scenario described in section V-A as a publish/subscribe model in which n services are published and m subscribers request their composition goals. A VS is represented as a tuple of the attributes: *storage*, *availability time*, and *reputation*. The attribute values of the n services are generated randomly. The values of the request attributes are generated randomly but with higher storage values so that a composition of services is needed to meet each request. Table I shows the ranges of service attribute values. In the experiments, we assume that 100 services are available and we vary the request arrival rate λ. For each case, the experiment is conducted 100 times.

Service performance is assumed to have a periodic daily or weekly pattern, based on the results of the long-term study by Lazaro et al. [33] who reported the presence of periodic patterns in the performance of volunteering hosts from the SETI@Home [34] system. The pattern lengths vary from one volunteer to another, ranging between hours and weeks.

2) Results: We compare the meta-self-aware version of the self-aware VSS with a non-meta-self-aware version, *i.e.,* each of the stimulus-, time-, and interaction-aware approaches. The experiments compare the four approaches in terms of the criteria identified in Section V-C.

PSR. Fig. 5 shows the average PSR for varying arrival rates λ. The average PSR of the interaction- and time-awareness cases are higher than that of stimulus-awareness in early simulation time μ. However, the PSR of the interaction- and the time-awareness drops as the request arrival rate increases. More importantly, the plots show that the average PSR of the meta-self-aware case matches the best PSR of the other three cases. For example, in Fig. 5(a), the PSR of the meta-self-awareness case is the same as with time-awareness, until the PSR of the interaction-awareness becomes higher (after μ = 300). At that point, the PSR of the meta-self-awareness will be the same as the interaction-awareness case. In Fig. 5(b) and (c) after

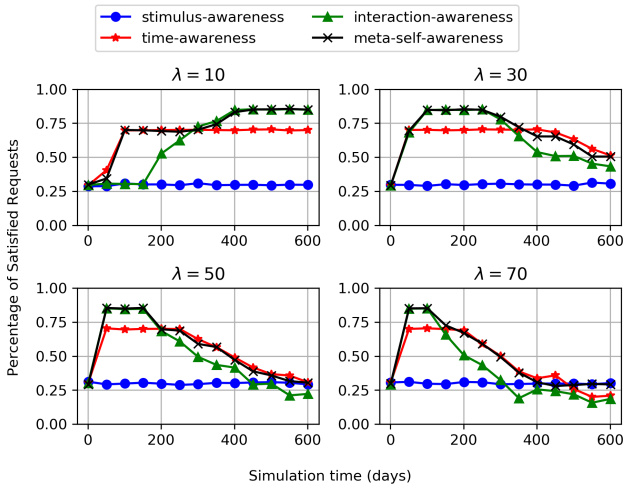

Fig. 5: Satisfying requests at different rates of request arrival (λ) is maximised through meta-self-awareness.

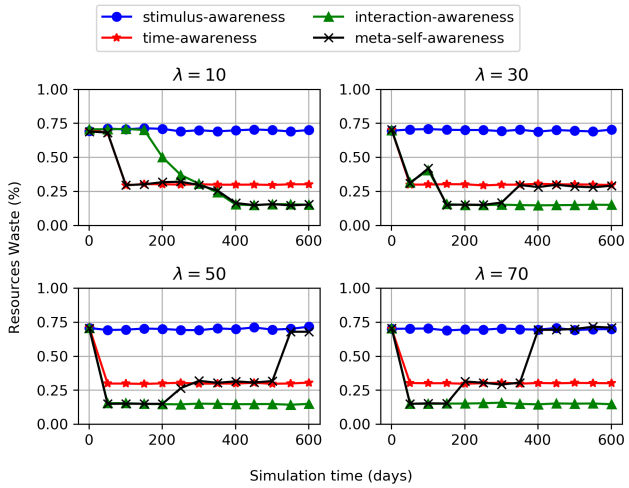

Fig. 6: Meta-self-awareness reduces resource waste. However, this is sensitive to the weights given to decision making criteria. In this experiment, higher weight is given to PSR.

the PSR of the interaction-awareness drops due to high arrival rate, the PSR of the meta-self-awareness will be the same as the time-awareness, which is the best one in that case. Another example is exhibited in Fig. 5(d), where the PSR of the meta-self-awareness case is close to that of the stimulus-awareness case (after μ = 400) which has the best PSR at that point.

Finding 1. *Switching between awareness levels through the meta-self-awareness approach results in obtaining the best possible percentage of satisfied requests.*

RW. Fig. 6 depicts the average RW for different values of λ. Initially, the stimulus-awareness RW is higher those of the interaction- and time-awareness cases. However, the latter two RWs rise as λ increases. More importantly, the plots demonstrate that the meta-self-aware RW matches the RW of the adopted awareness approach. For example, in Fig. 6(a), the RW of the meta-self-awareness case is the same as for time-awareness until μ = 300, as it is the adopted awareness level during this period. The adoption of the time-awareness level in this period, not the stimulus-awareness which has the least RW, is due to the higher weight assigned to the PSR metric in equation 3 which biases towards suggesting the level that is expected to have higher PSR. Fig. 6(a) also shows that the RW of the meta-self-awareness is the same as the interaction-awareness case after μ = 300, which is the least RW among the other approaches. In Fig. 6(b) and (c) the meta-self-awareness RW is the same as that of the interaction-awareness, which is the best until μ 300 and 200, respectively. In Fig. 6(d), the meta-self-awareness RW is close to the stimulus-awareness case (after μ = 400) even though it is not the least RW at that point. The reason again is the higher weight assigned to PSR in the experimentation setup of the what-if simulations.

Finding 2. *Switching between awareness levels allows the reduction of RW if the difference in the expected PSR of the levels is not significant (in case the PSR has a higher weight than RW).*

WT. Fig. 7(a), (b), (c), and (d) portray the average WT for different arrival rates λ. The average WT under stimulus-awareness case is always smaller than under interaction- and time-awareness. The average WT of the interaction- and time-awareness cases rises more rapidly, but linearly, as the request arrival rate increases. More importantly, the figures show that the average WT of the meta-self-aware case changes to the average WT of the adopted awareness approach. For example, in Fig. 7 (a), the WT of the meta-self-awareness case is the same as the time-awareness until μ = 300. This is because the adopted awareness level in this period is the time-awareness level. Fig. 7(a) shows also that the WT of the meta-self-awareness is the same as the interaction-awareness case after μ = 300, the period in which is the interaction-awareness level is adopted. In Fig. 7(c), the WT of the meta-self-awareness case will drop close to the stimulus-awareness case (after μ = 500) which is adopted in that case. Another example in Fig. 7(d), the WT of the meta-self-awareness case will be close to the stimulus-awareness case (after μ = 400) which is adopted in that case.

Finding 3. *Meta-self-awareness reduces waiting time when request arrival rate is high.*

Overhead. Meta-self-awareness comes at the cost of additional overhead that stems mainly from re-computing the selection and adaptation decisions using the different levels of self-awareness during the what-if analysis. Fig. 8 compares the computational time of the meta-self-aware and non-meta-self-aware approaches. The computational time of the meta-self-awareness approach is much higher compared to other approaches. Moreover, overhead increases with request arrival rate as more computation is needed to serve increasing number of requests to search for and adapt composite services. Furthermore, overhead increases over time due to the increase in the size of the knowledge accumulated about service performance.

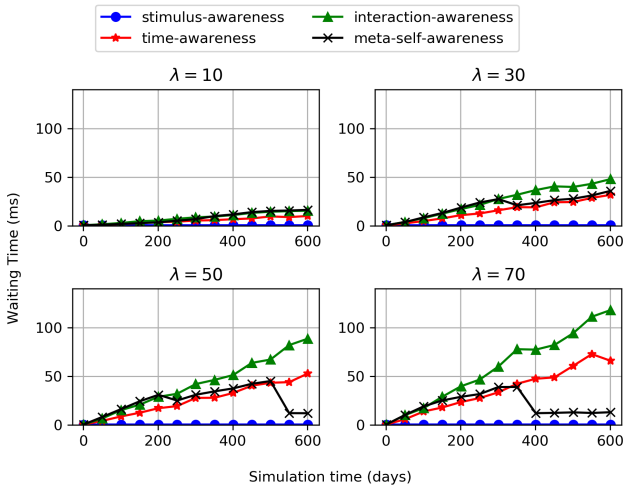

Fig. 7: Meta-self-awareness minimises request waiting times, but this again is sensitive to the weights given to decision making criteria.

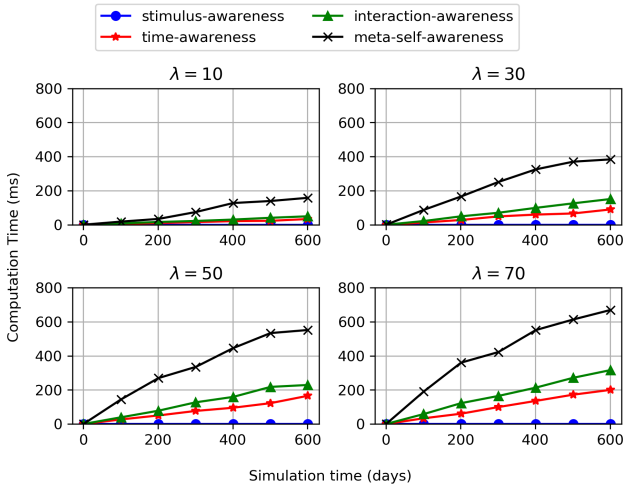

Fig. 8: The advantages of meta-self-aware are accompanied by some added computational overhead, albeit acceptable: peaking at ≈ 0.65 seconds for a very high value of λ.

Finding 4. *Additional computational overhead is the price of using symbiotic simulation to support self-adaptation at the meta-level. The acceptability of this overhead is subjective based on the criteria of the meta-self-awareness context.*

VI. Discussion

A. Assessment of QoN

The case study shows that the meta-self-awareness approach enables the switching between different levels of self-awareness based on the expected performance of each level and on the queue state. The *What-if* simulations use the knowledge captured by the different levels of self-awareness to evaluate the decisions that could have been taken by each level to satisfy the requests. This involves evaluating the QoN captured by the self-awareness levels which results in adopting the 'most promising' level. It also considers the overhead of knowledge representation and balances between benefits and overheads. Though this approach is evaluated using a case study of VC, it is general to any self-adaptive system. This is because self-adaptive systems should be able to collect knowledge and use it to reason about selecting an adaptation action from a set of potential actions. Then our approach can be used to model the quality of knowledge that is relevant to each adaptation action in order to reason about the selection of a certain action.

B. Evaluation

The quantitative evaluation of the case study is based on conducting experimental controlled simulations. This has the advantage of performing scalable experiments which are otherwise expensive to conduct on real systems. Undoubtedly, real deployments can show important differences and findings. However, the simulation results can be utilised to guide the real application of the meta-self-awareness approach.

On another observation, the results show that the switching from one level to another is sensitive to the values of the weights which are assigned to the metrics in the symbiotic simulation for suggesting a self-awareness level (see section IV-C and equation (3)). As we mentioned in regards to Fig. 6(a), the adopted level in the period where simulation time $<= 300$ was time-awareness although the stimulus-awareness exhibited the minimum waste. That is because a higher weight is assigned to the PSR metric in this experiment. Indeed, the weights assignment in the symbiotic simulations should get along with the specification of the system goals. However, a sensitivity analysis using weight space and how different weights can affect the decision making in meta-self-awareness would be an interesting aspect to investigate in the future.

The results show that the advantage of meta-self-awareness comes with a price of overhead in terms of computation time. However, the overhead increase exhibits a linear trend and reaches an albeit acceptable value of 0.65 seconds in a high-scale case. Nevertheless, further research that investigates the acceptance of this overhead compared to the benefits of the meta-self-awareness level is desirable from the research community. In relation to that, exploring the optimal placement of symbiotic simulator, *i.e.,* on the same machine of the self-aware framework or on dedicated computing machine to parallelise the computation and reduce the computation overhead, is an area of research for further work.

VII. Related Work

The architectural approach taken in this paper is based on the core idea of self-aware computing of explicitly designing systems in run-time knowledge concerns, and adapting these at run-time. Camara et al. [35] explore the relationship between the self-aware computing perspective and other related fields, such as self-adaptive software systems and autonomic computing. They show that in some respects, self-aware computing systems embody similar ideas to more advanced, proactive self-adaptive software. However, approaches falling under that umbrella do not explicitly consider the need for different forms

of run-time knowledge concerning the system, nor do they necessarily lead to the system itself containing knowledge of what run-time knowledge it possesses.

Recent works (*e.g.,* [36], [11]) highlight that there are many notable architectures that embody (often layered) control loops in the context of self-adaptive, -managing and -organising systems. These include MAPE-K [37], Observer/Controller [38], multi-agent systems [39], [40], and those inspired by robotics such as the three-layered architecture [41]. The literature on self-adaptive software systems [42], [43], [44] has focused in particular on the increasing complexity of service-oriented applications. Dominant approaches are surveyed in [45], [46].

Machine learning techniques have been recently used to learn the effect of each adaptation policy on the system goal and thus find correlation between the environment changes and the performance of the system, *e.g.,* [47], [48], [49]. This leads to greedily learn the right adaptation action and mitigate the uncertainty resulting from dynamism and unpredictability of the environment. However, what is still missing is the systematic and dynamic assessment of the quality of knowledge and the adaption of knowledge acquisition and representation.

Architectures emerging from research in self-aware computing differ from these in that they facilitate an engineering perspective that explicitly considers different levels of self-awareness that may be present in computing systems, supporting the analysis of knowledge concerns. This is what is seen in this work, where the experimental analysis allows us, and ultimately the system itself, to reason about the benefits and overheads of maintaining different knowledge sets. Ultimately, Self-adaptive systems make adaptation decisions based, at least partially, on run-time knowledge. Therefore, knowledge acquisition and representation are prerequisites for self-adaptation, and self-awareness can be considered as an enabler for effective self-adaptation.

One benefit of taking this knowledge-driven approach is that it allows not just for self-adaptation, but also for self-explanation. In the case of a self-aware system, knowledge it possesses concerning itself is subdivided into separate concerns, for example regarding its history, relations with other systems, predictions for the future, etc. Furthermore, in a meta-self-aware system, the system also holds knowledge about which of these concerns are present. For example, in the system described in this paper, the system would be able to report that it had made an adaptation decision based on historical information (in the case that its time-awareness strategy had been used), or that it had made a decision based on very little information (in the case that only stimulus-awareness had been used) and might, therefore, be less able to provide evidence to justify the adaptation decision to others. Ultimately, this approach can be expected to make the black box that is typically associated with adding learning to a self-adaptive system more transparent. This is important when dealing with adaptation decisions that impact on humans.

Early works on self-awareness [50], [51], [52] outlined the vision for designing systems with built in self-aware systems. Afterwards, many approaches were proposed to extend and realise the self-awareness vision. For example, in [53] Bicocchi et al. proposed an awareness framework for knowledge collection and classification in urban environments to reason on adaptation and to improve the energy efficiency in pervasive scenarios. Kounev et al. [54] presented a model-based approach to designing self-aware systems using an architecture-based modelling language [55]. The approach enables modelling the system's interactions to reasoning on the adaptation on the model-level. In [56] Salama et al. presented a generic approach for correlating QoS tactics with self-aware architecture patterns. Their approach tends to enrich self-aware patterns with QoS self-management capabilities to respond to QoS run-time requirements. Applications of self-aware computing have more recently been found in fog and mist computing [57], virtualised data centers [58], heterogeneous multi-core platforms for on-the-fly computing [58], interactive music devices [59], sensor networks [60], and the Internet of Things [61], as well as elsewhere in the cloud computing space [62]. In all of these, self-awareness has been used to help deal with the issue of run-time resource constraints, and it is notable that self-awareness is providing benefits particularly in time- and space-constrained systems. An overview of recent work in self-aware computing can be found in [63].

Finally, while there is plenty of work on meta-cognition (*e.g.,* [64]) and computational reflection (*e.g.,* [65]), none addresses the run-time meta-adaptation of a system's self-awareness capabilities by the system itself. Our work proposes a novel structure and quantitative approach for such self-adaptation at the meta-level, which is an important step towards the vision of effective and explainable self-adaptive behaviour based on run-time knowledge.

VIII. Conclusion

In this paper, we proposed an approach for dynamically and programmatically assessing the quality of knowledge that is learned and utilised to inform self-adaptation. In particular, meta-self-awareness utilises different types of knowledge that have been captured by the system in a *What-if* simulator to investigate alternative decisions that could have been taken using those types of knowledge. This assesses and compares the quality of knowledge and enables adapting the usage of the knowledge type that is expected to best inform the decision making. We believe this work to be a strong foundation for future work related to the assessment the of the self-awareness property of software-intensive self-adaptive systems. Our future work will focus on quantifying the presence and degree of self-awareness a system posses during its operation, which would help to investigate how to increase the system intelligence for better quality of decision making.

References

[1] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proceedings of the Tenth European Conference on Computer Systems*, ser. EuroSys. ACM, 2015.

[2] C. Curino, S. Krishnan, K. Karanasos, S. Rao, G. M. Fumarola, B. Huang, K. Chaliparambil, A. Suresh, Y. Chen, S. Heddaya, R. Burd, S. Sakalanaga, C. Douglas, B. Ramsey, and R. Ramakrishnan, "Hydra: a federated resource manager for data-center scale analytics," in *16th Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX, Feb. 2019, pp. 177–192.

[3] B. Maurer, "Fail at scale," *Queue*, vol. 13, no. 8, pp. 30–46, Sep. 2015.

[4] H. S. Gunawi, M. Hao, T. Leesatapornwongsa, T. Patana-anake, T. Do, J. Adityatama, K. J. Eliazar, A. Laksono, J. F. Lukman, V. Martin, and A. D. Satria, "What bugs live in the cloud? a study of 3000+ issues in cloud systems," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC. ACM, 2014, pp. 1–14.

[5] 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 on Foundations of Software Engineering*, ser. ESEC/FSE. ACM, 2015, pp. 307–319.

[6] F. Samreen, G. S. Blair, and Y. Elkhatib, "Transferable knowledge for low-cost decision making in cloud environments," *Transactions on Cloud Computing*, vol. tbc, p. tbc, May 2020.

[7] D. Chou, T. Xu, K. Veeraraghavan, A. Newell, S. Margulis, L. Xiao, P. M. Ruiz, J. Meza, K. Ha, S. Padmanabha, K. Cole, and D. Perelman, "Taiji: Managing global user traffic for large-scale internet services at the edge," in *Proceedings of the 27th Symposium on Operating Systems Principles*, ser. SOSP. ACM, 2019, pp. 430–446.

[8] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 350–361, Aug. 2011.

[9] G. Wang, L. Zhang, and W. Xu, "What can we learn from four years of data center hardware failures?" in *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2017, pp. 25–36.

[10] S. Kounev, X. Zhu, J. Kephart, and M. Kwiatkowska, "Model-driven algorithms and architectures for self-aware computing systems (dagstuhl seminar 15041)," *Dagstuhl Reports*, vol. 5, no. 1, 2015.

[11] S. Kounev, J. O. Kephart, A. Milenkoski, and X. Zhu, *Self-Aware Computing Systems*, 1st ed. Springer, 2017.

[12] T. Chen, F. Faniyi, R. Bahsoon, P. R. Lewis, X. Yao, L. L. Minku, and L. Esterle, "The handbook of engineering self-aware and self-expressive systems," *arXiv preprint arXiv:1409.1793*, 2014. [Online]. Available: http://arxiv.org/abs/1409.1793

[13] F. Faniyi, P. R. Lewis, R. Bahsoon, and Y. Xin, "Architecting self-aware software systems," in *IEEE/IFIP Conference on Software Architecture (WICSA)*, 2014, pp. 91–94.

[14] A. Elhabbash, R. Bahsoon, and P. Tino, "Self-awareness for dynamic knowledge management in self-adaptive volunteer services," in *International Conference on Web Services (ICWS)*, June 2017, pp. 180–187.

[15] EPiCS, "FET proactive initiative: Self-awareness in autonomic systems (awareness)," 2010. [Online]. Available: https://cordis.europa.eu/project/rcn/95042/factsheet/en

[16] H. Hoffmann, "SEEC: Creating systems that understand high-level goals and automatically adapt to meet those goals online," 2013. [Online]. Available: http://people.cs.uchicago.edu/~hankhoffmann/seec/

[17] ASCENS, "Ascens project," 2010. [Online]. Available: http://www.ascens-ist.eu/index-2.html

[18] Dagstuhl, "Dagstuhl seminars on software engineering for self-adaptive systems (sefsas)," 2018. [Online]. Available: https://www.hpi.uni-potsdam.de/giese/public/selfadapt/dagstuhl-seminars/

[19] R. Fujimoto, W. H. Lunceford, E. H. Page, and A. M. Uhrmacher, "Grand challenges for modeling and simulation," Dagstuhl report 350, 2002. [Online]. Available: http://www.dagstuhl.de/en/program/calendar/semhp/?semnr=02351

[20] C. Blum, A. F. T. Winfield, and V. V. Hafner, "Simulation-based internal models for safer robots," *Frontiers in Robotics and AI*, vol. 4, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/frobt.2017.00074

[21] H. Umme and S. Asghar, "A survey on multi-criteria decision making approaches," in *International Conference on Emerging Technologies (ICET)*. IEEE, oct 2009, pp. 321–325.

[22] M. N. Durrani and J. A. Shamsi, "Volunteer computing: requirements, challenges, and solutions," *Journal of Network and Computer Applications*, vol. 39, pp. 369–380, 2014.

[23] S. Abar, P. Lemarinier, G. K. Theodoropoulos, and G. M. OHare, "Automated dynamic resource provisioning and monitoring in virtualized large-scale datacenter," pp. 1–10, may 2014.

[24] F. Darema, "Dynamic data driven applications systems: A new paradigm for application simulations and measurements," in *International Conference on Computational Science*. Springer, 2004, pp. 662–669.

[25] H. Aydt, S. J. Turner, W. Cai, and M. Y. H. Low, "Symbiotic simulation systems: An extended definition motivated by symbiosis in biology," in *22nd Workshop on Principles of Advanced and Distributed Simulation*, 2008, pp. 109–116.

[26] ——, "Research issues in symbiotic simulation," in *Proceedings of the Winter Simulation Conference (WSC)*, 2009, pp. 1213–1222.

[27] D. Ardagna and B. Pernici, "Adaptive service composition in flexible processes," *IEEE Transactions on Software Engineering*, vol. 33, no. 6, pp. 369–384, 2007.

[28] J. Vilaplana, F. Solsona, I. Teixidó, J. Mateo, F. Abella, and J. Rius, "A queuing theory model for cloud computing," *The Journal of Supercomputing*, vol. 69, no. 1, pp. 492–507, Jul 2014.

[29] R. M. Loynes, "The stability of a queue with non-independent inter-arrival and service times," in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 58. Cambridge Univ Press, 1961, pp. 497–520.

[30] C. Hwang and K. Yoon, "Lecture notes in economics and mathematical systems," *Multiple Objective Decision Making, Methods and Applications: A State-of-the-Art Survey*, vol. 164, 1981.

[31] V. Vu, G. Park, G. Tan, and M. Ben-Akiva, "A simulation-based framework for the generation and evaluation of traffic management strategies," in *Proceedings of the Annual Simulation Symposium*, ser. ANSS. Society for Computer Simulation International, 2014, pp. 9:1–9:10.

[32] A. Elhabbash, R. Bahsoon, and P. Tino, "Interaction-awareness for self-adaptive volunteer computing," in *10th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. IEEE, sep 2016, pp. 148–149.

[33] D. Lázaro, D. Kondo, and J. M. Marquès, "Long-term availability prediction for groups of volunteer resources," *Journal of Parallel and Distributed Computing*, vol. 72, no. 2, pp. 281–296, 2012.

[34] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@Home: An experiment in public-resource computing," *Commun. ACM*, vol. 45, no. 11, pp. 56–61, Nov. 2002.

[35] J. Càmara, K. L. Bellman, J. O. Kephart, M. Autili, N. Bencomo, A. Diaconescu, H. Giese, S. Götz, P. Inverardi, S. Kounev, and M. Tivoli, *Self-aware Computing Systems: Related Concepts and Research Areas*. Springer International Publishing, 2017, pp. 17–49.

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

[37] A. Computing, "An architectural blueprint for autonomic computing," *IBM White Paper*, vol. 31, 2006. [Online]. Available: http://www-03.ibm.com/autonomic/pdfs/ACBlueprintWhitePaperV7.pdf

[38] C. Müller-Schloer, H. Schmeck, and T. Ungerer, *Organic computing – A paradigm shift for complex systems*, C. Müller-Schloer, H. Schmeck, and T. Ungerer, Eds. Springer Science & Business Media, 2011.

[39] R. A. Brooks, "Intelligence without representation," *Artificial Intelligence*, vol. 47, no. 1, pp. 139–159, 1991.

[40] S. J. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach*, 3rd ed. Pearson Education, 2010.

[41] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," in *Future of Software Engineering (FOSE)*. IEEE, 2007, pp. 259–268.

[42] A. Elkhodary, N. Esfahani, and S. Malek, "Fusion: a framework for engineering self-tuning self-adaptive software systems," in *Proceedings of the 18th SIGSOFT international Symposium on Foundations of Software Engineering*. ACM, 2010, pp. 7–16.

[43] N. M. Villegas, H. A. Müller, G. Tamura, L. Duchien, and R. Casallas, "A framework for evaluating quality-driven self-adaptive software systems," pp. 80–89, 2011.

[44] R. de Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, D. Weyns, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cukic, R. Desmarais, S. Dustdar, G. Engels, K. Geihs, K. M. Göschka, A. Gorla, V. Grassi, P. Inverardi, G. Karsai, J. Kramer, A. Lopes, J. Magee, S. Malek, S. Mankovskii, R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. Pezzè, C. Prehofer, W. Schäfer, R. Schlichting, D. B. Smith, J. P. Sousa, L. Tahvildari, K. Wong, and J. Wuttke, *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*. Springer Berlin Heidelberg, 2013, pp. 1–32.

[45] F. D. Macías-Escrivá, R. Haber, R. del Toro, and V. Hernandez, "Self-adaptive systems: A survey of current approaches, research challenges and applications," *Expert Systems with Applications*, vol. 40, no. 18, pp. 7267–7279, 2013.

[46] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker, "A survey on engineering approaches for self-adaptive systems," *Pervasive and Mobile Computing*, vol. 17, pp. 184–206, 2015.

[47] S. Shevtsov, D. Weyns, and M. Maggio, "SimCA*: A control-theoretic approach to handle uncertainty in self-adaptive systems with guarantees," *ACM Trans. Auton. Adapt. Syst.*, vol. 13, no. 4, Jul. 2019.

[48] B. Porter and R. R. Filho, "Distributed emergent software: Assembling, perceiving and learning systems at scale," in *13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, June 2019, pp. 127–136.

[49] R. de Lemos and M. Grześ, "Self-adaptive artificial intelligence," in *14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, May 2019, pp. 155–156.

[50] S. Kounev, F. Brosig, N. Huber, and R. Reussner, "Towards self-aware performance and resource management in modern service-oriented systems," in *Conference on Services Computing (SCC)*, 2010, pp. 621–624.

[51] F. Zambonelli, N. Bicocchi, G. Cabri, L. Leonardi, and M. Puviani, "On self-adaptation, self-expression, and self-awareness in autonomic service component ensembles," in *Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW)*. IEEE, 2011, pp. 108–113.

[52] T. Becker, A. Agne, P. R. Lewis, R. Bahsoon, F. Faniyi, L. Esterle, A. Keller, A. Chandra, A. J. Refsum, and C. Stilkerich, "EPiCS: Engineering proprioception in computing systems," in *15th International Conference on Computational Science and Engineering (CSE)*. IEEE, 2012, pp. 353–360.

[53] N. Bicocchi, D. Fontana, and F. Zambonelli, "A self-aware, reconfigurable architecture for context awareness," in *Symposium on Computers and Communication (ISCC)*, 2014, pp. 1–7.

[54] S. Kounev, N. Huber, F. Brosig, and X. Zhu, "Model-based approach to designing self-aware IT systems and infrastructures," *IEEE Computer Magazine*, 2016.

[55] S. Kounev, F. Brosig, and N. Huber, "The Descartes modeling language," Department of Computer Science, University of Wuerzburg, Tech. Rep., 2014. [Online]. Available: http://www.descartes-research.net/dml/

[56] M. Salama and R. Bahsoon, "Quality-driven architectural patterns for self-aware cloud-based software," in *8th International Conference on Cloud Computing (CLOUD)*, 2015, pp. 844–851.

[57] 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," *Computer*, vol. 48, no. 7, pp. 37–45, 2015.

[58] V. Beek, J. Donkervliet, T. Hegeman, S. Hugtenburg, and A. Iosup, "Self-expressive management of business-critical workloads in virtualized datacenters," *Computer*, vol. 48, no. 7, pp. 46–54, 2015.

[59] K. Nymoen, A. Chandra, and J. Torresen, *Self-awareness in active music systems*. Springer, 2016, pp. 279–296.

[60] B. Rinner, L. Esterle, J. Simonjan, G. Nebehay, R. Pflugfelder, G. D. Fernández, and P. R. Lewis, "Self-aware and self-expressive camera networks," *Computer*, vol. 48, no. 7, pp. 21–28, 2015.

[61] M. Möstl, J. Schlatow, R. Ernst, H. Hoffmann, A. Merchant, and A. Shraer, "Self-aware systems for the internet-of-things," in *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2016, pp. 1–9.

[62] T. Chen and R. Bahsoon, "Toward a smarter cloud: Self-aware autoscaling of cloud configurations and resources," *IEEE Computer*, vol. 48, no. 9, pp. 93–96, 2015.

[63] P. R. Lewis, "Self-aware computing systems: From psychology to engineering," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2017, pp. 1044–1049.

[64] M. T. Cox, "Metacognition in computation: A selected research review," *Artificial Intelligence*, vol. 169, no. 2, pp. 104–141, 2005.

[65] P. Maes and D. Nardi, Eds., *Meta-Level Architectures and Reflection*. Elsevier Science Inc., 1988.