Self-adaptive Volunteered Services Composition through Stimulus- and Time-awareness

Abdessalam Elhabbash, Rami Bahsoon, Peter Tino
School of Computer Science
University of Birmingham
Birmingham, United Kingdom
Email: {a.elhabbash, r.bahsoom, p.tino}@cs.bham.ac.uk

Peter R. Lewis
School of Engineering and Applied Science
Aston University
Birmingham, United Kingdom
Email: p.lewis@aston.ac.uk

***Abstract*—Volunteered Service Composition (VSC) refers to the process of composing volunteered services and resources. These services are typically published to a pool of voluntary resources. Selection and composition decisions tend to encounter numerous uncertainties: service consumers and applications have little control of these services and tend to be uncertain about their level of support for the desired functionalities and non-functionalities. In this paper, we contribute to a self-awareness framework that implements two levels of awareness, Stimulus-awareness and Time-awareness. The former responds to basic changes in the environment while the latter takes into consideration the historical performance of the services. We have used volunteer service computing as an example to demonstrate the benefits that self-awareness can introduce to self-adaptation. We have compared the Stimulus- and Time-awareness approaches with a recent Ranking approach from the literature. The results show that the Time-awareness level has the advantage of satisfying higher number of requests with lower time cost.**

Keywords*—*Volunteered Service, Stimulus-awareness, Time-awareness, Service Dependability

I. INTRODUCTION

Volunteer Computing (VC) is an emerging distributed computing paradigm in which users make their own resources available to others enabling them to do distributed computations and/or storage [1]. VC provides a multitude of advantages. Firstly, VC helps achieve efficient utilization of the computational and storage resources of the huge number of available computing devices like PCs, laptops, smart phones, etc. Secondly, VC has a positive impact on the environment by reducing the energy consumption as applications use idle resources instead of setting up new data centres or processing units. Finally, the premise of VC is to volunteer resources for little or no gain which reduces the costs of purchasing commercial resources for the users who will access those resources [2].

Many approaches have been proposed in the literature to enable volunteers to donate their resources for scientific projects, e.g. SETI@Home [3], Storage@Home [4], Folding@home [5], and others. Such projects are characterized by a requirement of large scale computation and/or storage. Taking into consideration that plenty of resources are available and idle on the users computing devices, purchasing resources for the large scale projects could be unreasonable, especially if the fund is not obtainable. Also, the increasing demand in resources may lead to some kind of 'monopoly' by few very big companies in the cloud market, according to [6]. So, utilizing the volunteered resources brings the advantages of large scale computing using inexpensive resources [7].

In Service-Oriented Computing (SOC), a service can be viewed as an elementary unit of the whole distributed application. That is, a distributed application consists of some number of distributed services that are composed together in order to provide the required functionality [8]. In VC, similar concepts can be applied on volunteered storage when considering this storage as a volunteered service (VS). Then, a composition approach is required to combine the VSs to form one composite service (CS) that satisfies the requester's required storage - a practice known as volunteered service composition (VSC).

The dynamism in the VC environment makes VSC a challenging task. Assume a volunteering environment in which users volunteer their idle storage. The VSC in such environment faces multiple challenges, including those listed below [9]:

- Storage-awareness: VSC should be able to compose the contributed storage achieving both maximum utilization and minimum waste with minimum computation time.
- Availability-awareness: the public contribute their resources during the time intervals in which they do not need these resources, i.e. the volunteered resources are not available permanently [10].
- Dilution of control: As volunteer services are offered on a voluntary basis by individuals and organizations willing to participate in the model, VC tends to exhibit 'dilution' of control increasing the level of uncertainty and the dynamism of the provision. This is because volunteered resources can be offered and withdrawn at any time [11]. The right without the symmetric obligation to participate in VC makes Service Level Agreements (SLAs) less stringent as when compared to commercial services.

The uncertainty associated with the volunteering environment makes the maintenance of the VSC beyond human capabilities. Consequently, self-adaptability becomes a vital requirement in VSC. A self-adaptive VSC should be able to take automatic and efficient adaptation decisions under the uncertainty associated with the behavior of the services. For example, the historical performance of the services can provide a significant input for self-adaptive service composition as it can be employed in learning the behavioral models of the services and predicting future performance which enables more intelligent adaptation decisions [12].

Recently, self-awareness and self-expression concepts are receiving more attention in computing [13]. Self-awareness is mainly the knowledge of the current state of the system. This knowledge is utilized in the self-expression to perform the adaptation actions. So, self-awareness and self-expression capabilities can provide self-adaptive systems with primitives for proactive management and control at runtime [14]. They can also improve both the accuracy and quality of adaptation. This may in turn converge the system towards more desirable stable states.

In previous work we developed a utility model for VSC, which enables a systematic approach for selecting and composing 'best' services to serve the user request [9] In this paper, we introduce a self-awareness framework for VSC. The framework enables acquiring knowledge on the services behaviour and adopting this knowledge VSC leading to reduce the violations of the users' requirements, i.e. leading to more stable states. The contribution of this paper is two-fold:

- A self-awareness architectural framework for informing self-adaptation in VSC. As part of the framework, two levels of awareness are defined, namely, Stimulus-awareness and Time-awareness, and two adaptation approached are proposed based on these levels.
- The evaluation of the VSC adaptation approaches, Stimulus-awareness-based adaptation, Time-awareness-based adaptation. The Stimulus-awareness-based approach is considered a baseline adaptation approach. The Time-awareness-based approach is based on taking into consideration the request time interval and the historical performance of the services in that time interval, which enables for fine grain representation of services historical performance. That is, the Time-aware approach can learn from the history the time intervals in which a service is more likely to perform well and the time intervals in which a service is more likely to perform poorly. In order to compare the approaches with the literature, we select the recent service selection Ranking approach presented in [15]. The motivation behind the selection of the ranking approach is that it can be comparable with the Time-aware approach as it considers the historical performance of the services by dividing the history age into a number of timeslots which helps to capture the variations of services performance over time.

The remainder of this paper is organized as follows: in the next section we present the self-awareness VSC framework. Section III introduces the VSC problem formulation. Section IV presents the self-aware approaches. Section V shows the evaluation results. Related works are outlined in section VI and we conclude in section VII.

II. Self-aware VSC Framework

A. Motivating Example

We motivate the need for integrating the concepts of self-awareness into VSC using a situation in which volunteered resources are offered as services. Assume a heterogeneous environment which consists of varied computing nodes like PCs, laptops, smart phones, etc., and these nodes are connected via a network. Individual people owning these nodes, known as publishers, offer their idle storage resources as services using a publish/subscribe model. Assume a subscriber may need to do some computations and store data temporarily but they have insufficient storage. To overcome this issue, subscribers can explore the network searching for volunteered storage services to use. If a subscriber finds the required storage while satisfying her requirements (e.g. location, security etc.), she can request it for her use. Otherwise, volunteered storages can be composed together to form a total storage that meets the subscriber needs. Fig.1 shows an example in which we use FindSpace4Me, a service which provides the subscribers the ability to find the required volunteered storage on one publisher node or any combination of volunteered storages that meet the subscriber's needs. The subscriber S1 invokes FindSpace4Me to search for storage of 40 GB. To make this volume available to S1, FindSpace4Me inspects the published storages and returns three possible composition strategies:

1) Using VS_1. In this case no composition is required.
2) Composing VS_2, VS_3, and VS_4.
3) Composing VS_2 and VS_5.

Now, which strategy should be selected to satisfy the 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, this adaptation helps to react to stimuli after they occur. But, if the system is able to anticipate the performance of the services, then it can select a strategy so that violations are less likely to occur, thus avoiding the violations. Also, the deeper the knowledge the system has on the services performance, the more intelligent the decision will be. For example, assume that S1 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 can avoid the selection of the third strategy. But, if we assume that S1 submitted a request at time t_2, and assume that the performance of VS_5 is anticipated to be well at t_2, then the system can select third strategy.

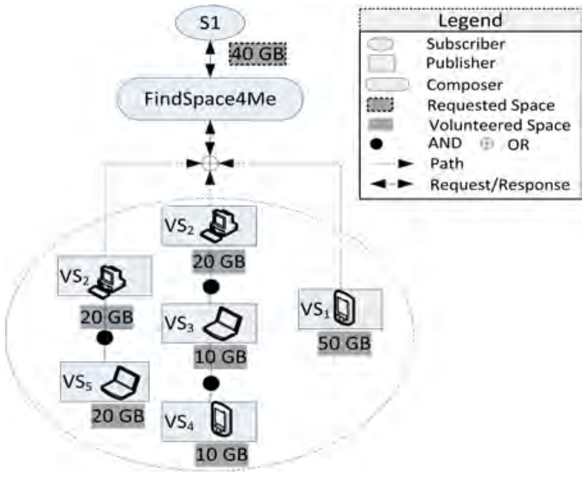


Fig. 1. Motivating Scenario - Composition request of S1 for 40GB using FindSpace4ME

B. Architectural Framework

In [11] we proposed a general framework for self-aware service composition which enables knowledge collection and representation for reasoning about adaptation in service composition. The framework is motivated by the self-awareness framework proposed by the EPiCS project in [12]. In this section, we overview a customized pattern of the framework. This pattern enables for considering the historical dependability of the services at the selection phase. The architectural diagram of the proposed framework is illustrated in Fig.2. The framework consists of the following basic components:

- Internal/external sensors: The sensors are responsible on collecting data on the services engaged in a composition (internal) and services available in the service repository (external). The data include any changes in the promised quality of service (internal) and the offering of new services in the service repository (external). Then the collected data are passed to the Stimulus-awareness and Time-awareness levels in the self-awareness component.
- Self-awareness: This component represents the knowledge collected by the sensors and passes the learnt models to the self-expression component. The Stimulus-awareness level represents the basic level of awareness i.e. this level enables the system to respond to the events received from the sensors. The Time-awareness level assumes the presence of the Stimulus-awareness and adds more awareness by considering the historical performance, in terms of dependability which is defined in the next section, of the services which enables the system to take more intelligent adaptation decision by selecting services which have high historical dependabilities.
- Self-expression: this component performs the actual adaptation action based on the learnt models received from the self-awareness component.

III. Volunteered Service Composition Model

In this section we briefly introduce the utility model and the composition approach we developed in [9]. This model is the base for our adaptive VSC. In this model, we focus on storage as a service, though our approach can readily be generalised to other types of service.

A. Model Definitions

Definition 1: (Volunteered Service). A Volunteered Service VS_i, is a 3-tuple (Stg_i, T_i, S_i) where Stg_i is the volunteered amount of storage, T_i is the time interval $[a_i, b_i]$ in which the VS_i is available, and S_i is the security level guaranteed by the service. A service repository (SR) is a set of disjoint volunteered services. We denote a SR with n services as $SR = \{VS_1, VS_2, \ldots, VS_n\}$. We denote the Stg_i,T_i, and S_i as the attributes of the service or the quality of the service.

Definition 2: (Subscriber's Request). A subscriber's request R is a 3-tuple (Stg^R, T^R, S^R), where Stg^R denotes the required storage, $T^R = [a^R, b^R]$ is the required time interval where $0 \leq a^R < b^R < 24$, and $a^R, b^R \in R$, and S^R is the required security level where $0 \leq S^R \leq S_{max}$ and $S^R, S_{max} \in \mathbb{N}$.

Definition 3: (Composite Service). Given a subscriber's request R, a Composite Service CS is a set of VSs, $\{VS_1, VS_2, \ldots, VS_k\}$, such that the following global constrains are satisfied (denoted as $CS \vdash R$):

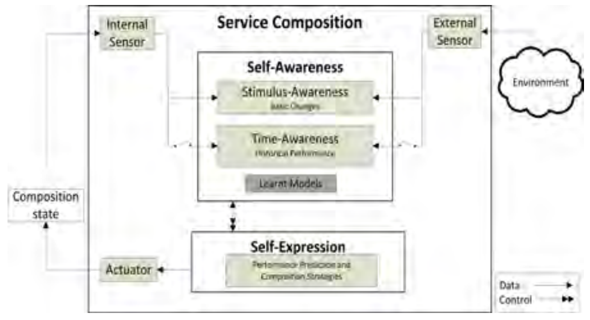


Fig. 2. Self-aware VSC framework

1) $VS_i \in SR, 1 \leq i \leq k \leq n$
2) $\sum_{i=1}^{k} Stg_i \geq Stg^R$, at any time instant in $[a^R, b^R]$.
3) $a^R \geq min[a_i]$ and $b^R \leq max[b_i]$, $\forall\ VS_i \in CS$
4) $S^R \geq S_i$, $\forall\ S_i \in CS$

Definition 4: (Storage Utility). Given a volunteered service VS_i and a request R, the storage utility $U_{stg}(VS_i)$ defined in Eq.(1) measures the amount of storage contributed by VS_i to R. This utility function gives maximum value of 1 if $Stg_i = Stg^R$ and a value less than 1 otherwise. The parameters β and α are set by the system administrator in a way that gives utility values to services with $Stg_i > Stg^R$ higher than the utility values of services with $Stg_i < Stg^R$, if $|Stg_i - Stg^R|$ is equal in both cases. The reason is to enable higher selection chance for the services with higher storage if there are no services with storage equal to the required storage which helps to avoid composition as there will be one service that satisfies the storage constraint.

$$U_{stg}(VS_i) = \begin{cases} e^{-\beta(Stg_i - Stg^R)}, & \text{if } Stg_i \geq Stg^R \\ e^{\alpha(Stg_i - Stg^R)}, & \text{if } Stg_i < Stg^R \end{cases} \tag{1}$$

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

Definition 5: (Availability Time Utility). Given a volunteered service VS_i and a request R, the time utility $U_{time}(VS_i)$, defined in Eq.(2), measures the amount of time contributed by VS_i to R. Services that will be available in a time interval exactly equals to $[a^R, b^R]$, will be assigned a maximum utility value of 1. On the other hand, services that will be available partially during $[a^R, b^R]$ or those that will be available in a time interval greater than $[a^R, b^R]$, will be assigned a utility lower than 1, i.e. reducing their chance of being selected to satisfy R. Otherwise, a zero-utility will be assigned.

$$U_{time}(VS_i) = \begin{cases} 0, & \text{if } b_i \leq a^R \text{or } b^R \leq a_i \\ e^{\gamma(b_i - b^R)}, & \text{if } a_i \leq a^R \text{and } a^R < b_i \leq b^R \\ e^{\gamma(a^R - b_i)}, & \text{if } b_i > b^R \text{and } a^R < a_i \leq b^R \\ \frac{e^{\alpha(b_i - a_i)}}{e^{\alpha(b^R - a^R)}}, & \text{if } a_i \geq a^R \text{and } b_i \leq b^R \\ \frac{e^{-\beta(b_i - a_i)}}{e^{-\beta(b^R - a^R)}}, & \text{otherwise} \end{cases} \tag{2}$$

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

Definition 6: (Security Utility). Given a volunteered service VS_i and a request R, the security utility $U_{sec}(VS_i)$ defined in Eq.(3) compares between the security level provided by VS_i

and the requested level. The function in Eq.(3) gives a zero-utility to those services that have a security level lower than the requested level. Also, it gives maximum value of 1 if $S_i = S^R$ and a value less than 1 otherwise. Furthermore, the greater the security level of VS_i than the required level, the lower the security utility of VS_i which allows for keeping high security services for serving future high security requests.

$$U_{sec}(VS_i) = \begin{cases} 1 - \triangle u(S_i - S^R), & \text{if } S_i \geq S^R \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

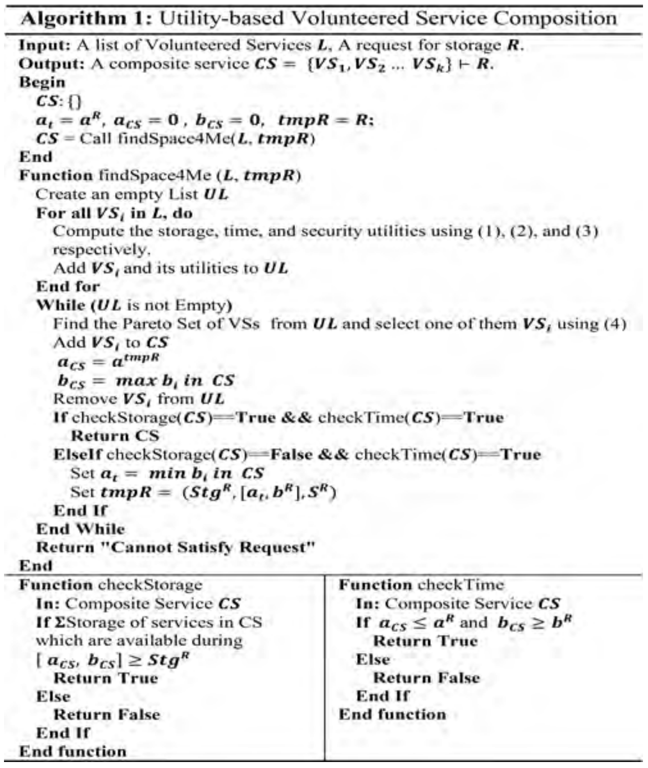
where $\triangle u = (1 - \epsilon)/S_{max}, 0 < \epsilon < 1$

B. Selection and Composition Algorithm

When a subscriber submits a composition request, the system retrieves the available services from the service repository and creates an empty CS set. Then the system computes the utility for each service in order to use these utilities as a selection criterion. After that the system finds the services which lie on the Pareto frontier and selects the highest utilities service using Eq.(4) and adds it to CS. Then, if the storage constraint specified in R is satisfied, the system returns CS to the requester, otherwise the process is repeated. If no composite service can be found to satisfy the subscriber's request, the system returns empty CS. The detailed composition algorithm is shown in Algorithm 1.

$$\begin{aligned} &\textit{maximize } (U_{Stg}(VS_i), U_{Time}(VS_i), U_{Sec}(VS_i)) \\ &\textit{subject to: } U_{Sec}(VS_i)) \geq 0, \\ &\qquad\qquad a_i \leq a_t \end{aligned} \tag{4}$$

where a_t initially equals a^R and is updated at run time.



Algorithm 1: Utility-based Volunteered Service Composition

```
Input: A list of Volunteered Services $L$, A request for storage $R$.
Output: A composite service $CS = \{VS_1, VS_2 \ldots VS_k\} \vdash R$.
Begin
  $CS$: {}
  $a_t = a^R$, $a_{CS} = 0$ , $b_{CS} = 0$,  $tmpR = R$;
  $CS$ = Call findSpace4Me($L$, $tmpR$)
End
Function findSpace4Me ($L$, $tmpR$)
  Create an empty List $UL$
  For all $VS_i$ in $L$, do
    Compute the storage, time, and security utilities using (1), (2), and (3)
    respectively.
    Add $VS_i$ and its utilities to $UL$
  End for
  While ($UL$ is not Empty)
    Find the Pareto Set of VSs  from $UL$ and select one of them $VS_i$ using (4)
    Add $VS_i$ to $CS$
     $a_{CS} = a^{tmpR}$
     $b_{CS} = max\, b_i\ in\ CS$
    Remove $VS_i$ from $UL$
    If checkStorage($CS$)==True && checkTime($CS$)==True
      Return CS
    ElseIf checkStorage($CS$)==False && checkTime($CS$)==True
      Set $a_t = min\, b_i\ in\ CS$
      Set $tmpR = (Stg^R, [a_t, b^R], S^R)$
    End If
  End While
  Return "Cannot Satisfy Request"
End
```

```
Function checkStorage
  In: Composite Service $CS$
  If ΣStorage of services in CS
  which are available during
  $[a_{CS}, b_{CS}] \geq Stg^R$
    Return True
  Else
    Return False
  End If
End function
```

```
Function checkTime
  In: Composite Service $CS$
  If $a_{CS} \leq a^R$ and $b_{CS} \geq b^R$
    Return True
  Else
    Return False
  End If
End function
```

IV. SELF-AWARE VSC APPROACHES

In this section, we present two self-adaptive VSC approaches, namely the Stimulus-aware and Time-aware VSC.

A. Stimulus-awareness VSC

In this approach, for each request R, the composition method searches for a composite service that satisfies R using Algorithm 1. When a change in the promised storage, availability, or security, of a service VS_i occurs, the self-expression initiates an adaptation action in order to replace the violating service VS_i. If the adaptation process is successful, then the violating service is replaced, otherwise the subscriber is notified that the violation cannot be treated. The Stimulus-aware adaptation approach is shown in Algorithm 2.

B. Time-aware VSC

The aim of Time-aware approach is to use the historical performance of the services to select the most appropriate services, i.e. services that provide what they promise. In our approach, we express the services performance in terms of *dependability*. We consider a service VS_i to be dependable if VS_i provides the storage, availability, and security it promises. In this section, we introduce a formal definition of *dependability* then the Time-awareness VSC approach.

1) VS dependabilities: The dependability evaluation provides a useful method for examining the behaviour of the service provider i.e. the volunteer. We use the *Dependability* measure to express the extent to which a selected service fulfils the promised resources and quality of service. As the deviation from the promised quality can be in any attributes, there will be a *Dependability* measure for each service attribute. Here we introduce the definition of *Dependability* as follows:

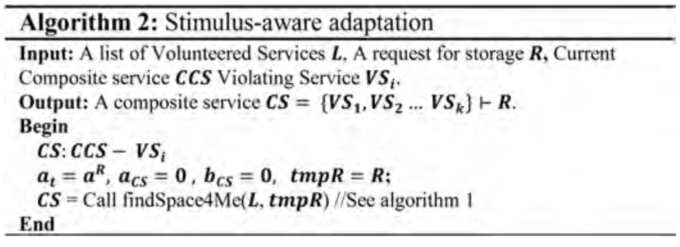
Definition 7: (Service Dependabilities). Given that a volunteered service VS_i has been selected in a composite service CS to serve the request R. Assume that $U^P_{stg}(VS_i)$ is the storage utility promised by the volunteer of VS_i. Assume also that the actual storage utility provided by VS_i during serving R is $U^A_{stg}(VS_i)$. Then the storage dependability of VS_i, $D_{stg}(VS_i)$, is defined as in Eq.(5):

$$D_{stg}(VS_i) = \begin{cases} \frac{U^P_{stg}(VS_i) - U^A_{stg}(VS_i)}{U^P_{stg}(VS_i)}, & \text{if } U^P_{stg}(VS_i) < U^P_{stg}(VS_i) \\ 1, & \text{otherwise} \end{cases} \tag{5}$$

Similarly, the availability time dependability ($D_{time}(VS_i)$), and the security dependability ($D_{sec}(VS_i)$) are defined as in Eq. (6) and Eq.(7).

$$D_{time}(VS_i) = \begin{cases} \frac{U^P_{time}(VS_i) - U^A_{time}(VS_i)}{U^P_{time}(VS_i)}, & \text{if } U^P_{time}(VS_i) < U^P_{time}(VS_i) \\ 1, & \text{otherwise} \end{cases} \tag{6}$$

$$D_{sec}(VS_i) = \begin{cases} \frac{U^P_{sec}(VS_i) - U^A_{sec}(VS_i)}{U^P_{sec}(VS_i)}, & \text{if } U^P_{sec}(VS_i) < U^P_{sec}(VS_i) \\ 1, & \text{otherwise} \end{cases} \tag{7}$$



Algorithm 2: Stimulus-aware adaptation

```
Input: A list of Volunteered Services $L$, A request for storage $R$, Current
Composite service $CCS$ Violating Service $VS_i$.
Output: A composite service $CS = \{VS_1, VS_2 \ldots VS_k\} \vdash R$.
Begin
  $CS$: $CCS - VS_i$
  $a_t = a^R$, $a_{CS} = 0$ , $b_{CS} = 0$,  $tmpR = R$;
  $CS$ = Call findSpace4Me($L$, $tmpR$) //See algorithm 1
End
```

2) History manipulation: Our aim is to acquire in-depth knowledge on the services' performance, in terms of dependabilities, so that the system can use that historical knowledge to determine the time intervals in which a service is most likely to fulfil the request requirements and the time intervals in which that service is most likely to violate the request requirements. To achieve that, our approach observes the services' historical dependabilities records, not only according to the request time spot but also according to the request time interval, unlike some approaches in the literature which observe the history only according to the request time spot. As we assume that the VS are volunteered on a daily basis during specific hours, we divide the day timeline into timeslots and record the dependabilities of each VS during each timeslot For simplicity, we assume the length of the timeslot to be one hour. Based on that method of history observation, each history record is stored according to the timeslot in which the service has been used. Figure 3 shows an example on how the dependability data points can be re-plotted according to the timeslots. Assume that a service VS_i was used six times in different days and assume that two out of these six times were during the timeslot [6-7], two during [7-8], one during [16-17], and one during [17-19]. Figure 3(a) shows how the data points are plotted in the 'classical'. They are plotted sequentially according to the time instance of usage. In Fig.3(b) the data points are re-plotted according to the timeslots in which the service has been used. The advantage here is that Fig.3(b) shows VS_i is most likely to be dependable if it is selected to serve requests in the timeslots [6-7] and [7-8] and undependable in the timeslots [17-18] and [18-19]. So, if a request time is e.g. [6-8] then VS_i is an appropriate service to be involved in a CS that server the request. But it is not the case if the request time is e.g. [17-19].

3) Dependabilities estimation: As there will be a certain number of historical dependabilities per timeslot, we need to estimate the distribution of the dependabilities data points of each service in each time slot. To do that, we adopt the following histogram-based method:

a. Divide the dependabilities range of values [0.0-1.0] into a series of small equal non-overlapping bins m_k, $k = 1, 2 \ldots J$, then count the number of data points that fall into each bin, $|m_k|$. Here we set J to 10, so we have the bins [0.0-0.1], (0.1-0.2] …(0.9, 1.0].
b. Find V, the set of bins with their lower limit greater than the dependability threshold, D_{th}, which is provided by the subscriber. For example, in our case if the threshold is 0.8, then the bins will be (0.9-1.0] and (0.8-0.9].
c. Compute the cumulative histogram CH for the intervals in the set V; $CH = \sum_{i=1}^{|V|} |m_i|$, where $m_i \in V$ and $|V|$ is the size of V.
d. The CH in each timeslot will be the representative dependability measure in that time slot.

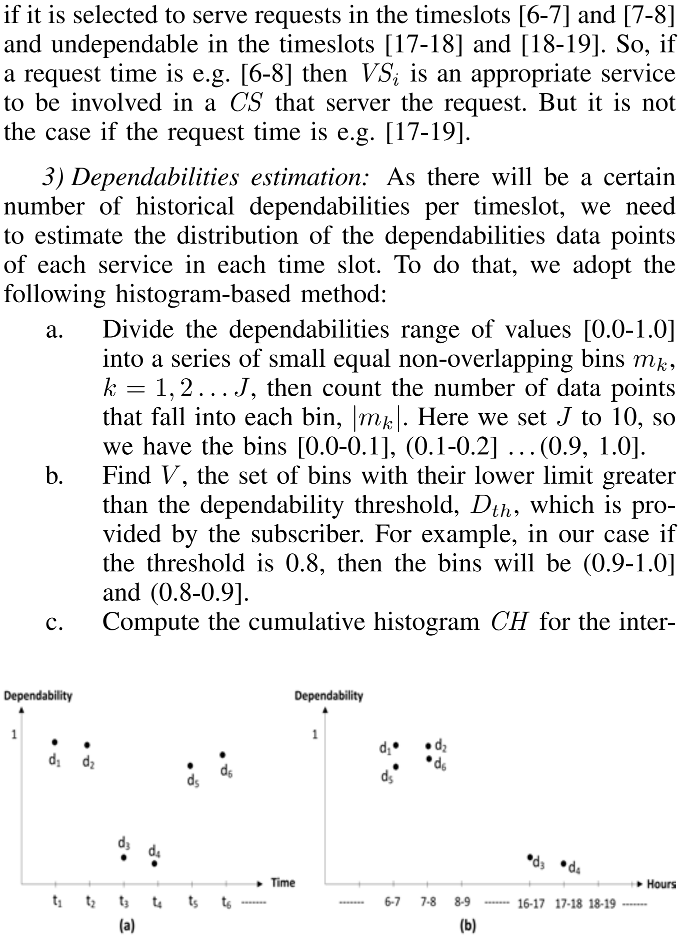


Fig. 3. Dependability plotting in (a) Classical (b) Time-aware approach

The above steps are repeated in each time slot for each service and for each dependability measure, i.e. the storage, time, and security dependabilities.

4) Time-aware service selection: Now, when a subscriber submits a request, the system executes the following key steps in order to satisfy the request:

a. Find the time slots that form the request interval. For example, if the request interval is $T^R = [9-12]$, then the time slots are [9-10], [10-11], [11-12].
b. For each $VS_i \in SR$, compute the U_{stg}, U_{time}, and U_{sec} using (1), (2), and (3) respectively.
c. For each time slot, estimate the representative D_{stg}, D_{time}, and D_{sec} for each $VS_i \in SR$ using (5), (6), and (7) respectively.
d. Find the average storage dependability, AVD_{stg}, for each VS_i by summing the representative storage dependability of each time slot and dividing over the number of time slots. Similarly find AVD_{time} and AVD_{sec}.
e. Find the Pareto-optimal services using (8), select one of them randomly, and add it to CS.

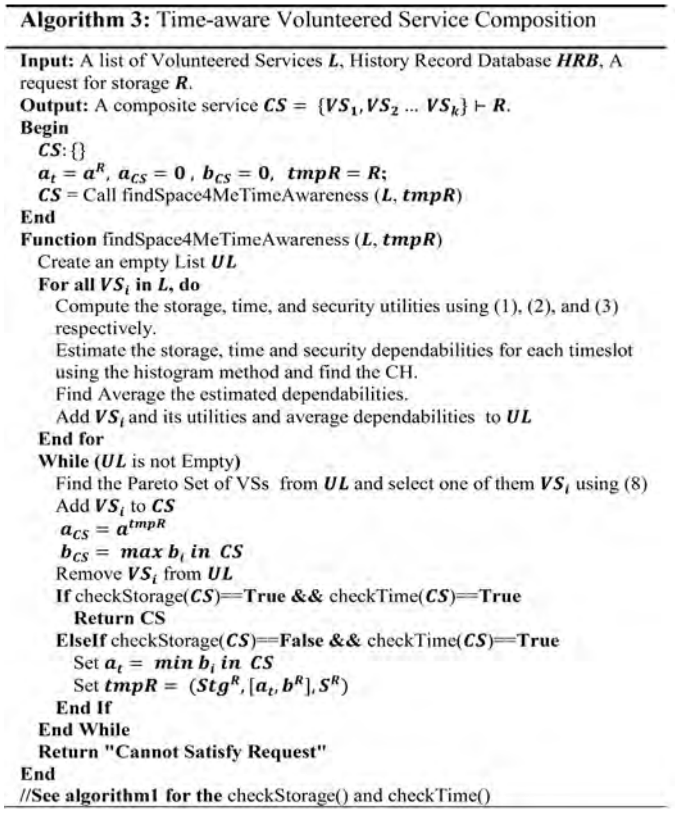
$$\begin{aligned} &\textit{maximize} \begin{pmatrix} U_{stg}(VS_i), & U_{time}(VS_i), & U_{sec}(VS_i), \\ AVD_{stg}(VS_i), & AVD_{time}(VS_i), & AVD_{sec}(VS_i) \end{pmatrix} \\ &\textit{subject to: } U_{sec}(VS_i) \geq 0, \\ &\qquad\qquad\quad a_i \leq a_t \end{aligned} \tag{8}$$

where a_t initially equals a^R and is updated at run time.

After executing the above steps, the subscriber request will be partially satisfied, then the request will be re-calculated and the above steps will be repeated to select the next service. After selecting each service, the global constraints (see section II.A) will be checked. If they are satisfied, the composite service CS will be returned; otherwise the above steps will be repeated. If all the services are visited and the global constraints are still not satisfied, an empty CS is returned and the subscriber will be notified that the request cannot be satisfied. The detailed greedy composition algorithm is shown in Algorithm 3.

5) Time-aware adaptation: Since Time-awareness involves the presence of the Stimulus-awareness, the adaptation in the Time-aware approach is two-fold, reactive adaptation and proactive adaptation.

- Reactive adaptation: This type of adaptation is similar to the Stimulus-awareness adaptation actions. When a change in the promised quality of service is reported to the Time-awareness component the actual utilities will be computed and subsequently the dependabilities using (5), (6), and (7). Then the dependabilities will be stored according to the timeslot. After that, an adaptation action will be carried out by the self-expression component in order to replace the service(s) that violated the requirements.
- Proactive adaptation: In this adaptation type, the system will predict the dependability of each service involved in a composition. During each timeslot, each dependability attribute of each VS_i involved in a CS



Algorithm 3: Time-aware Volunteered Service Composition

```
Input: A list of Volunteered Services $L$, History Record Database $HRB$, A
request for storage $R$.
Output: A composite service $CS = \{VS_1, VS_2 \dots VS_k\} \vdash R$.
Begin
  $CS$: {}
  $a_t = a^R$, $a_{CS} = 0$ , $b_{CS} = 0$,  $tmpR = R$;
  $CS$ = Call findSpace4MeTimeAwareness ($L$, $tmpR$)
End
Function findSpace4MeTimeAwareness ($L$, $tmpR$)
  Create an empty List $UL$
  For all $VS_i$ in $L$, do
    Compute the storage, time, and security utilities using (1), (2), and (3)
    respectively.
    Estimate the storage, time and security dependabilities for each timeslot
    using the histogram method and find the CH.
    Find Average the estimated dependabilities.
    Add $VS_i$ and its utilities and average dependabilities  to $UL$
  End for
  While ($UL$ is not Empty)
    Find the Pareto Set of VSs  from $UL$ and select one of them $VS_i$ using (8)
    Add $VS_i$ to $CS$
    $a_{CS} = a^{tmpR}$
    $b_{CS} = max\ b_i\ in\ CS$
    Remove $VS_i$ from $UL$
    If checkStorage($CS$)==True && checkTime($CS$)==True
      Return CS
    ElseIf checkStorage($CS$)==False && checkTime($CS$)==True
      Set $a_t = min\ b_i\ in\ CS$
      Set $tmpR = (Stg^R, [a_t, b^R], S^R)$
    End If
  End While
  Return "Cannot Satisfy Request"
End
//See algorithm1 for the checkStorage() and checkTime()
```

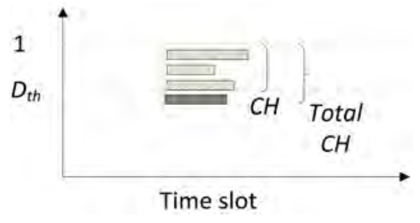


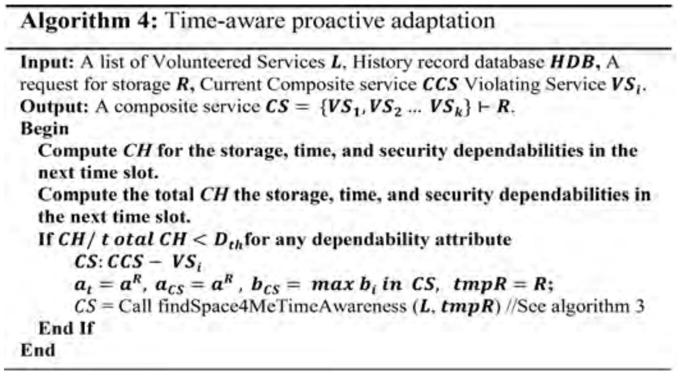
Fig. 4. CH in the interval [D_{th},1.0] and total CH

in the next timeslot is estimated using the histogram method. If the CH between in the interval [D_{th},1.0] divided by the total CH is less than D_{th} (see Fig.4) then that service is most likely to violate the constraints in the next time slot. Therefore, the system will search the service repository for an alternative VS in order to avoid a possible violation. Algorithm 4 shows the steps of the proactive adaptation.

V. Experimental Evaluation

In this section, we conduct experiments in order to evaluate the performance of the proposed Time-aware, and Stimulus-aware approaches against the Ranking [15] approach. The experiments were conducted on a desktop PC with an Intel core i5-3570 3.5 GHZ processor, 4G RAM, Windows 7, Java Standard Edition V1.7.0.

We implement the example described in Section II as a publish/subscribe model in which n services are published and m subscribers request their composition goals. A service is represented as a tuple of the attributes: Storage, Availability Time, and Security Level. The attributes' values of the n services were generated randomly. A subscriber's request is represented in the same way as services in addition to the Dependability threshold D_{th}. The values of the requests were generated randomly also but with higher Storage values so that a composition of services is needed to meet each the request. Table I shows the ranges of the services attributes values. The history dataset has been generated such each service shows variations in the dependabilities over the different timeslots. The threshold D_{th} has been set to 0.9. In the experiments we assume that we have 100 services, along with their corresponding history, which is a reasonable number to indicate scalability compared with the literature, e.g. the number of services in [15] is 5. We also vary the number of requests m and the number of history records in each time slot. For each test case, the experiment was conducted 100 times and the average was computed.



Algorithm 4: Time-aware proactive adaptation

```
Input: A list of Volunteered Services $L$, History record database $HDB$, A
request for storage $R$, Current Composite service $CCS$ Violating Service $VS_i$.
Output: A composite service $CS = \{VS_1, VS_2 \dots VS_k\} \vdash R$.
Begin
  Compute $CH$ for the storage, time, and security dependabilities in the
  next time slot.
  Compute the total $CH$ the storage, time, and security dependabilities in
  the next time slot.
  If $CH/\ total\ CH < D_{th}$for any dependability attribute
      $CS: CCS - VS_i$
      $a_t = a^R$, $a_{CS} = a^R$ , $b_{CS} = max\ b_i\ in\ CS$,  $tmpR = R$;
      $CS$ = Call findSpace4MeTimeAwareness ($L$, $tmpR$) //See algorithm 3
  End If
End
```

TABLE I. Range of Attributes' Values

	Service		Subscriber	
Attribute	min	max	min	max
Storage	5	20	20	30
Availability Time	8	19	9	20
Security	0	4	0	4

The experiments compare the Stimulus-awareness, the Time-awareness, and the Ranking approaches in the time cost and the average percentage of requests satisfaction:

- Time Cost. Defined as the average summation of the time needed to generate the composite services and the time needed to adapt to the constraints violations (whether reactively or proactively in the Time-awareness approach) in milliseconds.
- % Requests Satisfied. Defined as the number of requests that the composition algorithm could successfully generate composite services for divided by the total number of requests. This metric is related to the efficiency of selecting services. That is, selecting dependable services will lead to fewer violations and hence more requests will be satisfied.

A. *Comparison in % Satisfied Requests*

The first experiment evaluates the number of requests that each approach can satisfy proportional to the total number of requests submitted. Fig. 5 compares the percentage of satisfied requests using the three composition approaches. The experiment is repeated 100 times and the percentage here is computed as the average of satisfied requests. The results show that the percentage of satisfied requests is the least in

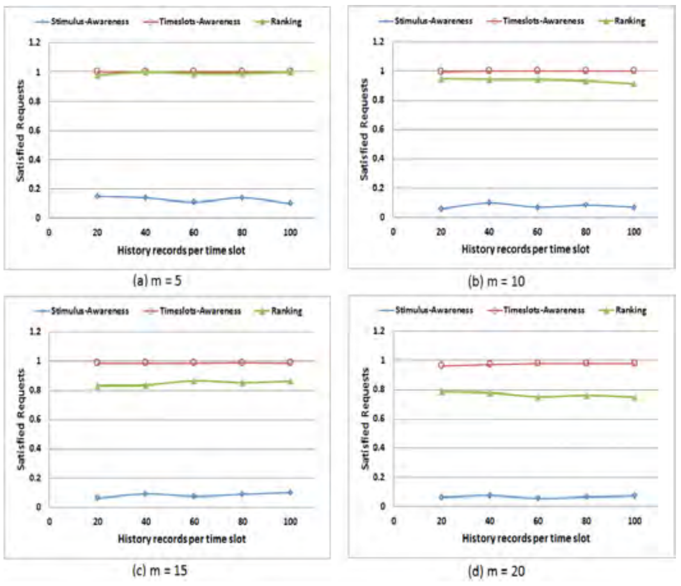


Fig. 5. Comparison of the three approaches in percentage of satisfied requests.

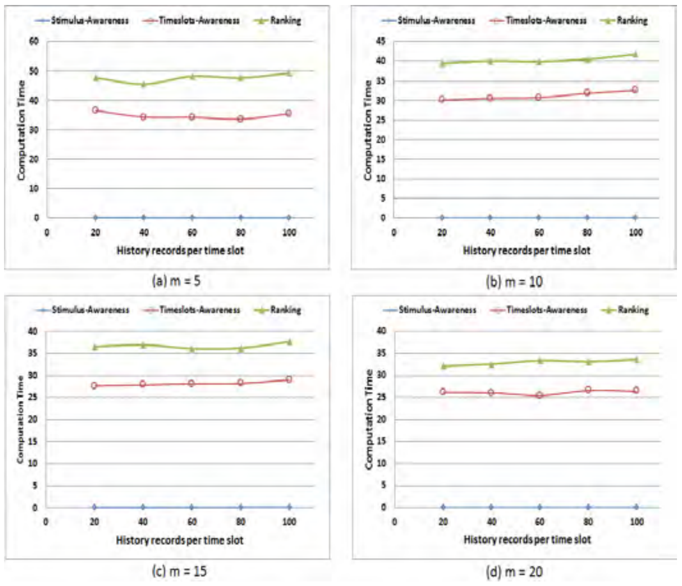


Fig. 6. Comparison of the three approaches in Time cost.

the Stimulus-aware case whereas it is the best in the Time-aware case especially when the number of requests increases i.e. in Fig. 5(c) and Fig. 5(d). The figures show also that the percentage of satisfied requests in the three approaches decreases when m increases; the reason is that the higher the number of requests while the fixed number of services raises the probability of the inability of finding the composite services for some requests. However, the Time-aware approach has the advantage of satisfying higher number of requests compared to the other approaches.

B. *Comparison in Time Cost*

The second experiment evaluates the time of generating and adapting the composite services of the three approaches. Figure 6 plots computation cost (in milliseconds) of selecting services for VSC and adapting to violations. The figure shows that the Stimulus-aware approach has the least time cost whereas the Ranking approach has the highest. It is notable also that the time cost in the Time-aware and the Ranking approaches increases slowly with the increase of the number of history records, whereas the time cost in the Stimulus-aware approach is not affected as it does not use the services history.

C. *Discussion*

The experimentation results show that leveraging history in service selection helps satisfying more requests which is a result of decreasing the violation of the constraints since learning from history enables the selection of the highly dependable services. However, this improvement is accompanied with an overhead in terms of the computational cost. In the comparison between the Time-aware approach and the Ranking approach, the results show that the Time-awareness is more efficient. The reason is that the Time-aware approach is based on acquiring in depth knowledge. In other words, the Time-aware approach is able to discriminate between the times of good performance and those of poor performance whereas the Ranking approach can only benefit from the history if the service performance changes over time. For example, the Time-awareness approach can inform that a certain VS is dependable if we use it in the morning but not in the evening, whereas the Ranking approach can only inform that a certain VS was undependable in the past but now it is becoming dependable.

An important threat of validity of the Time-aware approach is related to the quality of the historical data, specifically, the absence of a pattern of the services performance. That is, if the services performance is highly variable and intervals of good or poor performance cannot be discriminated then the Time-awareness efficiency may decrease.

VI. Related Works

A. *Volunteer Computing Paradigms*

In this section, we present an overview of the deemed volunteer computing frameworks. BOINC is the earliest VC middleware [2]. It enables for creating public-resource computing projects. Through this middleware, users can participate their PCs and specify their contributions to the projects. SETI@home [3] is one of the earliest volunteer computing projects that use BOINC. It uses volunteered resources to analyse radio signals from space instead of special purpose supercomputers. Folding@home [5] project benefits from the huge computational power of volunteered processing resources to simulate a biological process, the protein folding. Storage@home [4] is a project that has been developed to enable backing up, storing, and sharing huge amounts of scientific results using volunteered resources. The following projects are also other examples of BOINC-based projects; the UCB/Intel study of Internet resources [16], Climateprediction.net [17], Climate@home, CERN projects, and Predictor@home [16]. Cloud@Home [6] is a computing paradigm that has been proposed to enable resource sharing on either voluntary or commercial basis. Contributing resources for scientific research projects or commercial data centres can be possible application for the Cloud@Home paradigm. Social Cloud [18] is a paradigm that takes advantage of pre-existing social networks trust relationships to share resource among users. In this paradigm, users can discover and use volunteered storage shared by their friends in an online social network.

None of the existing approaches address the VSC problem and the dynamism of the volunteering environment. They do not provide answers on how to deal with the changes in the environment and how to adapt to those changes.

B. Self-adaptive Frameworks

Many frameworks have been proposed to architect self-adaptive systems. IBM MAPE-K [19] enables representing knowledge about the environment and using this knowledge to adapt the system behaviour at runtime through Monitor, Analyse, Plan, and Execute cycle to meet user's objectives. Another reference model was introduced in [20]. The model defines three layers; Component Control which reports the current state to higher layers, Change Management which responds to the state change by executing some predefined actions, and Goal Management which produces plans in response to change in the goals. These two architectures [19] [20] were presented as reference models rather than as a guide to implement self-adaptive systems. In [21] an architecture has been proposed to monitor the state of a system in order to find any runtime problems, e.g. constraints violation. Using these frameworks as a foundation, other frameworks were proposed to extend its capabilities by adopting machine learning techniques. For example, [22] presents a learning based framework entitled FUSION in which adaptation decisions are taken based on monitoring the system at runtime to learn the system's runtime behaviour unforeseen at design time. The framework defines a particular system capability as a feature. One drawback of this framework is that it depends on the user's knowledge in reducing the features' space, an assumption that is not practical in the service composition problem as the size of services is too large for human to process. A recent architecture has been proposed in [12] for adopting self-awareness as a prerequisite for self-adaptability in computing systems. The proposed architecture defines five levels of self-awareness for knowledge acquisition and representation in addition to self-expression component that make use of the obtained knowledge to take appropriate actions in response to the system and environment dynamics. In this paper, we applied a custom pattern of the self-awareness framework to reason on the reactive and proactive adaptation in VSC problem.

VII. Conclusion

We have proposed a self-awareness framework to enhance the self-adaptation capabilities of volunteered service composition. One of the core fundamentals of the proposed framework is that it considers the services historical performance in the selection of the services that will composed to server the users' requests. The historical performance is also used to reason on the reactive and proactive adaptation of the composite services. We have experimentally evaluated and compared three approaches of adaptation; the Stimulus-aware, the Time-aware, and the Ranking approach. The results show the one of the benefits that Time-awareness capabilities can bring to a self-adaptive application is satisfying more requests since it tends to select services that exhibit high dependability and low probability of violating the requests constraints. A scenario of volunteer storage is introduced to evaluate the framework and approaches. In the future, we will work on implementing other levels of awareness of the framework and evaluate the potential benefits and overheads of these also.

References

[1] O. Nov, D. Anderson, and O. Arazy, "Volunteer computing: A model of the factors determining contribution to community-based scientific research," in *Proceedings of the 19th International Conference on World Wide Web*, ser. WWW '10. New York: ACM, 2010, pp. 741–750.

[2] D. Anderson, "Boinc: a system for public-resource computing and storage," in *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, Nov 2004, pp. 4–10.

[3] D. Werthimer, J. Cobb, M. Lebofsky, D. Anderson, and E. Korpela, "Seti@home&mdash;massively distributed computing for seti," *Computing in Science and Engg.*, vol. 3, no. 1, pp. 78–83, Jan. 2001.

[4] A. Beberg and V. Pande, "Storage@home: Petascale distributed storage," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, March 2007, pp. 1–6.

[5] A. Beberg, D. Ensign, G. Jayachandran, S. Khaliq, and V. Pande, "Folding@home: Lessons from eight years of volunteer distributed computing," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, May 2009, pp. 1–8.

[6] V. Cunsolo, S. Distefano, A. Puliafito, and M. Scarpa, "Volunteer computing and desktop cloud: The cloud@home paradigm," in *Network Computing and Applications, 2009. NCA 2009. Eighth IEEE International Symposium on*, July 2009, pp. 134–139.

[7] M. N. Durrani and J. A. Shamsi, "Volunteer computing: requirements, challenges, and solutions," *Journal of Network and Computer Applications*, vol. 39, no. 0, pp. 369 – 380, 2014.

[8] M. Moghaddam and J. Davis, "Service selection in web service composition: A comparative review of existing approaches," in *Web Services Foundations*, A. Bouguettaya, Q. Z. Sheng, and F. Daniel, Eds. Springer New York, 2014, pp. 321–346.

[9] A. Elhabbash, R. Bahsoon, P. Tino, and P. R. Lewis, "A utility model for volunteered service composition," in *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*, Dec 2014, pp. 337–344.

[10] S. Dustdar, C. Dorn, F. Li, L. Baresi, G. Cabri, C. Pautasso, and F. Zambonelli, "A roadmap towards sustainable self-aware service systems," in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '10. New York, NY, USA: ACM, 2010, pp. 10–19.

[11] A. Elhabbash, R. Bahsoon, and P. Tino, "Towards self-aware service composition," in *In Proceedings of the 16th IEEE International Conference on High Performance and Communications, to appear*, 2014.

[12] F. Faniyi, R. Bahsoon, X. Yao, and P. R. Lewis, "Architecting self-aware software systems," in *Proceedings of the 11th Working IEEE/IFIP Conference on Software Architecture (WICSA 2014)*, Sydney, 2014.

[13] 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 *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2011 Fifth IEEE Conference on*, Oct 2011, pp. 108–113.

[14] 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 *Computational Science and Engineering (CSE), 2012 IEEE 15th International Conference on*, Dec 2012, pp. 353–360.

[15] Z. Rehman, O. Hussain, and F. Hussain, "Parallel cloud service selection and ranking based on qos history," *International Journal of Parallel Programming*, vol. 42, no. 5, pp. 820–852, 2014.

[16] M. Taufer, C. An, A. Kerstens, and C. Brooks, "Predictor@home: A "protein structure prediction supercomputer' based on global computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 17, no. 8, pp. 786–796, Aug 2006.

[17] Climateprediction.net. [Online]. Available: http://www.climateprediction.net/

[18] R. Chard, K. Bubendorfer, and K. Chard, "Experiences in the design and implementation of a social cloud for volunteer computing," in *IEEE 8th International Conference on E-Science*, Oct 2012, pp. 1–8.

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

[20] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," in *Future of Software Engineering*, 2007.

[21] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, pp. 46–54, Oct 2004.

[22] A. Elkhodary, N. Esfahani, and S. Malek, "Fusion: A framework for engineering self-tuning self-adaptive software systems," in *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE '10. New York, NY, USA: ACM, 2010, pp. 7–16.