

# ENGINEERING SELF-AWARENESS WITH KNOWLEDGE MANAGEMENT IN DYNAMIC SYSTEMS: A CASE FOR VOLUNTEER COMPUTING

by

ABDESSALAM ELHABBASH

A thesis submitted to  
The University of Birmingham  
for the degree of  
DOCTOR OF PHILOSOPHY

School of Computer Science  
College of Engineering and Physical Sciences  
The University of Birmingham  
May 2017

UNIVERSITY OF  
BIRMINGHAM

**University of Birmingham Research Archive**

**e-theses repository**

This unpublished thesis/dissertation is copyright of the author and/or third parties. The intellectual property rights of the author or third parties in respect of this work are as defined by The Copyright Designs and Patents Act 1988 or as modified by any successor legislation.

Any use made of information contained in this thesis/dissertation must be in accordance with that legislation and must be properly acknowledged. Further distribution or reproduction in any format is prohibited without the permission of the copyright holder.



# Abstract

The complexity of the modern dynamic computing systems has motivated software engineering researchers to explore new sources of inspiration for equipping such systems with autonomic behaviours. Self-awareness has recently gained considerable attention as a prominent property for enriching the self-adaptation capabilities in systems operating in dynamic, heterogeneous and open environments. This thesis investigates the role of knowledge and its dynamic management in realising various levels of self-awareness for enabling self-adaptivity with different capabilities and strengths. The thesis develops a novel multi-level dynamic knowledge management approach for managing and representing the evolving knowledge. The approach is able to acquire 'richer' knowledge about the system's internal state and its environment in addition to managing the trade-offs arising from the adaptation conflicting goals.

The thesis draws on a case from the volunteer computing, as an environment characterised by openness, heterogeneity, dynamism, and unpredictability to develop and evaluate the approach. This thesis takes an experimental approach to evaluate the effectiveness of the of the dynamic knowledge management approach. The results show the added value of the approach to the self-adaptivity of the system compared to classic self-adaptation capabilities.



*To my parents, wife, sons, brothers, sisters, parents-in-law  
with loyalty and love...*



# Acknowledgement

First, I am extremely grateful to my supervisor Dr Rami Bahsoon for his impressive support throughout my PhD and related research. Your guidance (during both the research and writing-up stages), patience, and motivation kept me continuously engaged in my research and made my time productive. Thank you Dr Rami. Also, I would like to sincerely thank my co-supervisor Prof. Peter Tino for his generous support and guidance.

I would like to thank my external supervisor Dr Peter Lewis and the members of my thesis monitoring group, Dr Eike Ritter and Prof. Joshua Knowles for their valuable comments and encouragement which triggered me to broaden my investigation.

I express a very special gratitude for the Islamic Development Bank (IDB) for their generous financial support for my PhD.

I owe enormous thankfulness and gratitude to my wife for her immeasurable moral and emotional encouragement and patience. I am also grateful to my children for their 'understanding' for my busyness.

Many thanks to my previous and current officemates who were the best I could have hoped for; thanks Christopher Novakovic, Olle Fredriksson, Mohammed Al-Wanain, Ahmed Al-Ajeli, Xiaodong Jia, Cory Knapp, Richard Thomas, Chris McMahon-Stone, Christopher Hicks, and Rajiv Singh for the fruitful and entertaining conversations.



# Table of Contents

CHAPTER 1	INTRODUCTION .....	1
1.1	Motivation .....	1
1.2	Problem Statement .....	6
1.2.1	Areas Requiring Improvement.....	7
1.2.2	Research Questions.....	9
1.3	Research Methodology.....	10
1.4	Thesis Contribution.....	12
1.4.1	Contribution in Brief.....	12
1.4.2	Thesis-related Publications .....	14
1.4.3	Thesis Roadmap.....	16
CHAPTER 2	SYSTEMATIC REVIEW OF SELF-AWARENESS IN SOFTWARE SYSTEMS..	20
2.1	Overview .....	20
2.2	Self-adaptation in Dynamic Software Systems.....	21
2.2.1	Limitations of Self-adaptive Systems.....	25
2.3	Self-Awareness in Software Engineering: A Systematic Literature Review .....	27
2.3.1	Review Protocol.....	27
2.3.2	Conducting the Review.....	35
2.3.3	Reporting the Review.....	42
2.3.4	Discussion .....	71
2.4	Gaps in Brief.....	75
2.5	Related Reviews.....	76
2.6	Summary .....	77
CHAPTER 3	MOTIVATING SELF-AWARENESS FOR VOLUNTEER COMPUTING.....	80
3.1	Overview .....	80
3.2	Volunteer Computing: Challenges and Characteristics .....	81
3.2.1	Performance Patterns of the Volunteer Hosts .....	84
3.3	Representative Volunteer Computing Systems .....	87

3.3.1	BOINC.....	88
3.3.2	Cloud@Home.....	90
3.3.3	Nebula.....	93
3.3.4	Cloud4Home .....	95
3.3.5	SocialCloud .....	96
3.4	Gaps Analysis .....	99
3.5	Summary .....	100
<b>CHAPTER 4 SERVICES SELECTION FOR VOLUNTEER COMPOSITE SERVICES: A UTILITY MODEL .....</b>		<b>103</b>
4.1	Overview .....	103
4.2	Volunteer Storage As a Service: A Steering Example.....	104
4.2.1	Volunteer Storage Scenario .....	106
4.2.2	Formulation of Volunteer Service Selection .....	108
4.3	Services Selection for Volunteer Composite Services .....	114
4.3.1	Exhaustive Search.....	115
4.3.2	Random Assignment.....	116
4.3.3	A Utility Model for Volunteer Composite Services.....	117
4.4	Experimental Evaluation .....	123
4.4.1	Experimental Results .....	125
4.5	Conclusion .....	130
<b>CHAPTER 5 SELF-AWARE FRAMEWORK FOR VC WITH DYNAMIC KNOWLEDGE MANAGEMENT .....</b>		<b>133</b>
5.1	Overview .....	133
5.2	Motivation for Self-awareness .....	135
5.3	General Architecture for Self-aware Volunteer Computing.....	136
5.4	Architecture of the Self-aware Framework.....	140
5.4.1	Overview of the EPiCS Framework .....	141
5.4.2	The Self-aware Framework for the Volunteer Computing.....	144
5.5	Dynamic Histograms for Dynamic Knowledge Management.....	147
5.5.1	Chebyshev's Inequality.....	149
5.5.2	Evolution Operations .....	150
5.6	Self-aware Selection and Adaptation Levels .....	153
5.6.1	Stimulus-aware Selection and Adaptation .....	153
5.6.2	Time-aware Selection and Adaptation.....	154

5.6.3	Interaction-aware Selection and Adaptation .....	158
5.7	Experimental Evaluation.....	160
5.8	Conclusion .....	166
<b>CHAPTER 6 SYMBIOTIC-BASED META-SELF-AWARENESS FOR SELF-AWARE SYSTEMS.....</b>		<b>169</b>
6.1	Overview .....	169
6.2	Symbiotic Simulations: A background.....	172
6.3	Symbiotic-based Meta-self-awareness Approach.....	176
6.4	Experimental Evaluation.....	184
6.4.1	Performance of Meta-self-awareness.....	184
6.4.2	Overhead of Meta-self-awareness .....	190
6.5	Conclusion .....	191
<b>CHAPTER 7 REFLECTIONS, CONCLUSION REMARKS, AND FUTURE WORK.....</b>		<b>195</b>
7.1	Reflections.....	195
7.1.1	Complexity.....	195
7.1.2	Scalability.....	198
7.1.3	Overhead .....	200
7.1.4	Practical Deployment.....	200
7.2	How the Research Questions are Addressed .....	202
7.3	Future Work .....	207
7.4	Closing Remarks .....	210
<b>APPENDIX A GLOSSARY.....</b>		<b>212</b>
<b>REFERENCES .....</b>		<b>213</b>

# List of Figures

Figure 1.1: Roadmap of the Thesis.....	18
Figure 2.1: Research Methodology.....	28
Figure 2.2: Quality Scores for the Primary Studies.....	40
Figure 2.3: Thematic Analysis of Self-Awareness in Software Engineering.....	41
Figure 2.4: Distribution of Primary Studies over Publication Types.....	43
Figure 2.5: Number of Publications per Year.....	44
Figure 2.6: Distribution of Publications by Affiliation Country.....	45
Figure 2.7: Distribution of Studies by Software Paradigms.....	55
Figure 2.8: Distribution of Studies by Engineering Practices.....	57
Figure 2.9: Distribution of Studies by Engineering Approaches.....	59
Figure 2.10: Distribution of Studies by Evaluation Approaches.....	66
Figure 3.1: Trace of A Random Subsample of the Hosts Availability. Source [5].....	84
Figure 3.2: Hosts Clustered by Availability. Source [111]. .....	85
Figure 3.3: Cloud@Home Architecture. Source [118] .....	91
Figure 3.4: Nebula System's Architecture. Source [121] .....	94
Figure 3.5: Cloud4Home Architecture. Source [122] .....	96
Figure 3.6: Social Storage Cloud Architecture. Source [108] .....	97
Figure 4.1: Motivating Scenario - Composition Request of $S_1$ .....	107
Figure 4.2: Volunteer Service Representation.....	110
Figure 4.3: Storage Request Representation.....	110
Figure 4.4: Example of a Composite Service.....	112
Figure 4.5: Waste Computation Example.....	113
Figure 4.6: Services Storage Utility.....	119
Figure 4.7: Services Time Utility Example.....	120
Figure 4.8: Services Security Utility.....	121

Figure 4.9: Average Waste in Small-Scale Experiment.....	126
Figure 4.10: Average Waste in High-Scale Experiment.....	127
Figure 4.11: Average PSR in Small-Scale Experiment.....	128
Figure 4.12: Average PSR in High-Scale Experiment.....	129
Figure 4.13: Average Waiting Time in Small-Scale Experiment.....	129
Figure 4.14: Average Waiting Time in High-scale Experiment.....	130
Figure 5.1: Architecture for Self-aware Volunteer Storage System .....	137
Figure 5.2: The Conceptual Self-Awareness Framework (Source [14]).....	142
Figure 5.3: The Self-Awareness Framework for The Volunteer Computing Case.....	145
Figure 5.4: Comparison in Resources Waste.....	162
Figure 5.5: Comparison in Percentage of Satisfied Requests.....	163
Figure 5.6: Comparison in Waiting Time.....	165
Figure 6.1: The Mutually Beneficial Symbiotic Simulation Paradigm. Adapted from [145] .....	172
Figure 6.2: Symbiotic Simulation Paradigm (a) Closed-Loop (b) Open-Loop.....	174
Figure 6.3: Architecture of The Meta-Self-Awareness Level.....	178
Figure 6.4: The State Diagram of The Awareness Levels Switching.....	183
Figure 6.5: Comparison in Percentage of Satisfied Requests.....	185
Figure 6.6: Comparison in Resources Waste.....	187
Figure 6.7: Comparison in Waiting Time.....	189
Figure 6.8: Comparison in Computation Time.....	191
Figure 7.1: Example Showing The Interface of A Subscriber Using A Composite Service of Two Volunteer Services.....	201

# List of Tables

Table 2.1: Manual Search Sources.....	31
Table 2.2: Studies Selection Criteria.....	33
Table 2.3: Quality Assessment Checklist.....	34
Table 2.4: Data Items Extracted from Primary Studies.....	35
Table 2.5: Search Execution (Search Strings and Settings) .....	37
Table 2.6: Automated Search Results.....	38
Table 2.7: Quality Assessment Total Results.....	40
Table 2.8: Active Research Communities within Self-Awareness.....	44
Table 2.9: Definitions of Self-Awareness.....	46
Table 2.10: Analysis of Self-Awareness Definitions.....	49
Table 2.11: Specific Motivations of Using Self-Awareness.....	52
Table 2.12: Source of Inspiration for Engineering Self-Awareness.....	53
Table 2.13: Software Paradigms Employing Self-Awareness.....	54
Table 2.14: Engineering Practices Realising Self-Awareness.....	56
Table 2.15: Engineering Approaches and Related Studies.....	58
Table 2.16: Evaluation Approaches and Related Studies.....	64
Table 2.17: Evaluation Criteria and Related Studies.....	67
Table 3.1: Summary of the Analysis of the Representative VC Systems.....	98
Table 4.1: Utility Model Attributes' and Parameters' Values.....	124
Table 5.1: The Values of the Parameters.....	160





# CHAPTER 1

## INTRODUCTION

### 1.1 Motivation

The complexity of current software applications is a consequence of the evolution of the computing paradigms. For instance, in 1990, the Grid Computing paradigm was proposed to enable High-Performance Computing (HPC) by using a collection of computing machines to obtain high computing power [1]. On top of this paradigm, the volunteer computing (VC) paradigm was proposed to employ donated idle computing power for solving complex problems. After that, recent advances in computing environments, such as services and cloud, had provided a natural platform for supporting the vision and operation of volunteer cloud computing in which volunteer resources can incubate a cloud [2]. This paradigm combines the cloud computing and volunteer computing to make the donated resources available for free<sup>1</sup> usage as services. For example, volunteer resources can be treated as services, which can be

---

<sup>1</sup> It is worth mentioning here that the free usage of the volunteer resources is in terms of the cost of purchasing computational and storage resources from the end users perspective. However, other aspects e.g. the inefficient energy consumption, is still a limitation of the volunteer computing paradigm that requires further investigating.

published and used on demand and benefit from modern computing paradigm including the Cloud.

Heterogeneity, dynamism, and uncertainty are common characteristics across the current environments. Modern applications are composed of multiple entities, e.g. services that are supplied by different providers. Taking into consideration that different providers adopt different computing resources and continuously update their infrastructures, current applications should be able to combine heterogeneous and inter-organisational computing resources dynamically [3]. Furthermore, providers frequently exhibit changes to their infrastructures by publishing new services, modifying current services, or withdrawing provided services. Such dynamism necessitates the dealing with large space of configurations at runtime in order to preserve the goals of the system. Obviously, this is exacerbated by the evolution of the users' requirements and fluctuations of their demands. Under these circumstances, the assumption that the quality of the services' provision is deterministic cannot be valid [4] [5]. Furthermore, unpredictable changes (e.g. security attacks) that cannot be anticipated at design time can occur arbitrarily at runtime; resulting in disrupting the desired functionality and/or the quality of the system [6]. Consequently, the decisions made with regards to the management of current software applications need to take into account the associated uncertainties.

The complexity resulting from the above characteristics (i.e. heterogeneity, dynamism, and uncertainty) makes the management of software applications beyond human capabilities [7]. Consequently, self-adaptation has become a vital requirement to enable the systems to maintain their goals in dynamic and unpredictable environments.

In response to this requirement, considerable research efforts have been exerted to develop approaches for self-adaptive systems [8]. This has been demonstrated by proposing self-adaptive approaches that equip the system with autonomous and cost-effective management capabilities [9] [7]. Among several definitions mentioned in [10], one states that *“Self-adaptive software evaluates its own behaviour and changes behaviour when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible.”* [11].

Meanwhile, these characteristics contribute to the complexity of the modern computing paradigms with some varying degrees. In other words, although heterogeneity, dynamism, and uncertainty are common among the different computing environments, they can be more challenging in certain environments. This can be illustrated by this brief comparison between the cloud and the volunteer environments. In the Cloud, service providers agree to adhere to a certain level of service provision, or quality of service (QoS), encoded as terms of a service level agreement (SLA). Violating the SLA terms cannot be completely unavoidable and in some cases, service providers are obliged to pay penalties due to violating SLAs. From the client's perspective, violations cause dissatisfaction with the service as their requirements may not be satisfied. On the other hand, providers in the volunteer environment are volunteers. They offer their physical resources on a voluntary basis. There are no strict SLAs to oblige the volunteers to provide the promised resources [12], e.g. volunteers may not be able to continue offering their resources and they can withdraw their resources at any time without any consequent retribution. Developing applications in VC environments, thus, is more challenging and requires more attention to the uncertainties related to unpredictable changes in the provision of services to satisfy users' requirements.

Therefore, the adoption of volunteer computing can benefit from the progress that we make in making the paradigm more intelligent and its management more seamless. In this context, the consideration of data and its management is crucial to the operations of these systems.

Recent research efforts hold the view that current self-adaptive systems tend to provide limited adaptation capabilities. For example, it has been reported that most self-adaptive approaches lack the awareness about the implicit effects of the adaptation decisions taken at runtime, resulting in limited capabilities in facing the continuous changes and meeting the users' and system's quality requirements [13] [14] [15]. Therefore, this has motivated the need for the concept of *self-awareness* in computing systems, which has recently received considerable attention, with solutions drawing inspirations from Psychology, Cognitive Science, Natural Sciences, and others. The commonality in these solutions is that self-awareness has been motivated as an enabler for self-adaptation in dynamic and unpredictable environments. The purpose is to enrich the self-adaptation capabilities by gaining in-depth knowledge about the system and the environment while considering both current and future states.

The concept of self-awareness in computing systems has been investigated by some representative efforts under the EU Proactive Initiative Self-Awareness in Autonomic Systems [16] and road-mapping agenda of the Dagstuhl seminar [17]. Such efforts aim to develop a fundamental understanding of what self-awareness in computing means. However, although the research on self-awareness in computing has been ongoing for a decade, there is still a lack of common understanding of the concept. A specific definition of self-awareness in computing and a clear characterisation of self-aware software

systems still require further investigation. Moreover, although self-awareness has been viewed as an enabler for self-adaptation for the sake of realising better autonomy and more intelligent adaptation decision making, current self-aware approaches do not sufficiently demonstrate the claimed capabilities of self-awareness. Among the limitations, the environments that current research has looked at tend to exhibit little dynamism and divert from one of the core motivations behind introducing self-awareness in self-adaptive systems. Furthermore, the distinction between self-adaptive and self-aware systems is still unclear.

Meanwhile, recent research has proposed frameworks for engineering self-aware systems [18] [14] [19], contributing to advances in the state-of-art and practice of computational self-awareness. In [18] a system is considered self-aware if it can model the acquired knowledge at multiple levels of awareness, namely, stimulus-, time-, interaction-, goal-, and meta-self-awareness (definitions and discussion of these level are introduced in chapter2). Though the separation and coordination of these awareness levels were discussed, the attempt is still abstract and lack concrete demonstration. This observation is attributed to the absence of approaches and frameworks that explicate dynamic knowledge management in self-awareness. In particular, pending questions that relate to how the 'fine-grained' knowledge, which corresponds to the different levels of awareness, can be modelled and realised still exist. Invoking the levels of awareness is subject to knowledge adequacy for the said level. The exercise also entails understanding for the trade-offs between the benefits that a given level provides and the overhead of adopting that level. In this context, how these knowledge levels can be coordinated based on the availability and adequacy of the knowledge to enable the analysis is still a pending issue. Moreover, though self-awareness is realised through the

acquisition and representation of knowledge (which is dynamic in nature in open systems) there is still a general lack of principled knowledge management approaches, which are quantitative in nature to realise self-awareness and its engineering principles in support of self-adaptation.

## **1.2 Problem Statement**

Developing self-aware applications, which are (1) able to satisfy the users' requirements and (2) flexible enough to efficiently adapt to the various changes, is still a pending challenge in software engineering for dynamic software systems. In this context, knowledge capturing and management constitutes an essential requirement for achieving self-adaptivity. Recent research on software engineering for self-aware systems has illustrated that knowledge management is still an open and critical research challenge (e.g., [14] [20] [19]). For example, [20] argued that coarse-grained knowledge representation makes it difficult to capture the trade-offs that exist between the different knowledge concerns that relate to stimuli, time, goal, interaction, etc. That means, the limited attention given to the granularity of the knowledge can misguide the adaptation process as the knowledge can relate to different concerns (e.g. events, historical performance, interactions, changes in goals). Based on that, the work in [14] takes a fine-grained approach in architecting the knowledge according to the different concerns of the knowledge. The work proposes a conceptual framework to separate the different concerns of the captured knowledge where each concern can reason about a different type of adaptation. Such approach can be considered as a step towards gaining more in-depth knowledge about the system state, however, the problem is still open and many related questions are still not answered. For example, in a service-based

application, what type of knowledge is assumed to be available about the services and can such knowledge be used by the system for informing the selection and adaptation decisions? How can the knowledge be represented so that the system can discover the performance patterns of services? How can the system dynamically decide on the knowledge concerns that should be used?

### **1.2.1 Areas Requiring Improvement**

The following areas of current self-aware approaches require further investigation:

- *Limited dynamic knowledge management.* Self-awareness has been motivated by the high dynamism and heterogeneity of the environments of the current software systems. A contribution by this thesis's author [21] indicates that the issue of dynamic knowledge management in current self-aware approaches is considered in an abstract form, where contributions have provided reference and conceptual models without clear guidance on their implementation. These approaches explicitly declare their concern of adopting computational self-awareness as a way for informing the adaptation decision-making process and consequently improving self-adaptation capabilities of the system. However, they do not provide explicit and concrete approaches for dynamic knowledge management. Knowledge needs to be treated as moving targets that continuously arrive and evolve. That is, knowledge needs to be treated as a 'stream' that continuously arrive and evolve at runtime. Moreover, knowledge needs to be represented in a way that is able to capture the performance patterns and trends of the system entities. In the same context, knowledge management needs to be realised through concrete approaches. Such approaches should also take into

consideration that different concerns are attached to the collected knowledge and that the separation of the knowledge concerns is vital to support different self-adaptation capabilities. Although there are some attempts for separating the knowledge concern in the related literature, they are still abstract and lack realisation.

- *Lack of dynamic management of the adaptation capabilities.* The ability to separate the different knowledge concerns in self-aware systems equips the system with multiple adaptation capabilities. In other words, the system can have different self-adaptation capabilities due to the existence of multiple levels of self-awareness, each concerned with a certain type of knowledge. Each of these awareness levels can model a certain type of knowledge, thus, they can collectively provide in-depth knowledge that informs the decisions making process. Naturally, there will be overhead and usefulness for each of those levels. Based on that, the self-aware system needs to have the capability to manage the trade-off that exists between those levels, e.g. by switching between them based on the advantages and disadvantages of each of them.
- *Identifying applications and environments, which can better utilise computational self-awareness.* From our systematic literature review [21]; we also noticed that existing approaches tend to apply their proposals in environments that exhibit limited dynamism, which shows a lack of consistency with the purpose of using the computational self-awareness. The current attempts to demonstrate the need for self-awareness tend to use cases that exhibit limited dynamism, e.g. due to the existence of SLAs that drive the providers to fulfil the user's requirements. A case that exhibits high dynamism and dilution of control is required to illustrate that

the classic self-adaptive approaches do not sufficiently satisfy the users' and systems requirements and that the need for self-awareness is obvious. In addition, it can be mutually beneficial to adopt self-awareness in such environment. From one side self-awareness can be better demonstrated and, from the other side, the self-adaptation capabilities in that environment can be improved.

### **1.2.2 Research Questions**

This thesis addresses the following research questions:

RQ1) How to characterise self-awareness and what motivated the research and applications of self-awareness in software engineering for dynamic and scalable software systems? Answering this question can help us to lay down the foundation of the thesis in relation to existing work, to identify shortcomings of current solutions in relation to dynamic systems (such as volunteer computing), and to pin out areas for improvements that the thesis hopes to address.

RQ2) What are the requirements for enacting self-awareness in dynamic software systems and how can these requirements be addressed? Among the requirements that the thesis is interested in are the adequacy of knowledge, its representation, management, evolution, and separation of concerns.

RQ3) As different knowledge concerns enable different levels of self-awareness, how can the system seamlessly switch between various levels of awareness while considering the adequacy and quality of the knowledge in enabling self-awareness? What mechanisms can help in coordination?

RQ4) How does VC, as a representative paradigm, render itself as a sensible environment for understanding and demonstrating potential improvements related to knowledge management in self-aware systems?

### **1.3 Research Methodology**

This thesis adopts a classical research methodology, inspired by [22], for carrying out our research. The methodology defines five steps that are executed iteratively to guide the research process:

- *Problem identification.* The first step is to acquire knowledge about the problem domain, i.e. self-awareness in software systems. For this purpose, a systematic literature review (SLR) has been conducted. The SLR also identified the progress in this research topic along with the open problems. As the understanding of the problem domain is gained, the research direction converged to the most interesting problem related to knowledge management in self-aware systems, which is the pivotal problem handled in this thesis.
- *Objectives of the solution.* Driven by the aforesaid problem, the main objective of this thesis is to develop an approach for dynamic knowledge management in self-aware software systems. This main objective is twofold, on the one hand, it is related to separate the knowledge concerns and develop multiple levels of knowledge management where each level is concerned with a certain type of knowledge. On the other hand, it is related to dynamically manage the trade-offs that exist among these levels. In addition to that, other objectives emerged from the main objective. As self-awareness has been motivated as an imperative need in dynamic environments, a representative environment that exhibits high

dynamism and dilution of control is required for presentation of the proposed approach. This objective requires the development of a concrete decision-making scenario that is used as a basis for the development of the dynamic knowledge management approaches.

- *Design and development.* The results of the SLR revealed that there are many approaches that incorporate self-awareness in dynamic software systems. For the purpose of achieving the objectives mentioned above, one ‘flavour’ of these approaches, namely the EPiCS framework [18], is selected in this thesis to fundamentally improve it by introducing the knowledge management to it. This framework conceptually introduces multi-levels of self-awareness to enrich the self-adaptivity. By fundamentally improving this framework, this thesis proposes multiple knowledge management approaches along with the capability of autonomous switching between those approaches according to system state. For this objective, this thesis proposes leveraging a symbiotic simulation to perform *what-if* analysis in order to investigate the alternative decisions that could be taken by the different awareness levels using the collected knowledge.
- *Demonstration.* This thesis adopts the volunteer computing paradigm for demonstrating the usefulness of the proposed self-aware approach. A motivating scenario of volunteer storage services is developed in order to steer the development of the knowledge management approaches. This scenario requires the development of a selection and adaptation approach for volunteer services that is used as a basis for the development and steering the presentation of the dynamic knowledge management approaches. On the other hand, the proposed self-aware approach (with dynamic knowledge management) contributes to

enriching the adaptation capabilities of the VC paradigm, which are found to be limited as shown in chapter 3.

- *Evaluation.* Experimental quantitative evaluation is used to compare the performance of the self-aware framework in the cases of the presence and the absence of the dynamic knowledge management approach. The evaluation is based on simulations-based experiments run in a controlled environment.

## **1.4 Thesis Contribution**

### **1.4.1 Contribution in Brief**

The thesis treats evolving knowledge as moving targets that need to be dynamically managed in order to improve self-adaptivity in dynamic software systems. The thesis investigates the problem of dynamically managing knowledge in self-aware software systems and its implications on improving self-adaptivity and its dependability. The main contributions of this thesis are:

- *Systematic Literature Review of Self-awareness in Software systems.* Dynamic knowledge management is the fundamental characteristics that we call for improving the self-adaptivity of software systems. That is, the self-aware system should be able to acquire in-depth knowledge about its state and the environment. A systematic literature review (SLR) was conducted to build the required understanding of the computational self-awareness and the related latest developments. The review aimed also at identifying the gaps in the current approaches and motivated the need for knowledge management approaches. This contribution deals with the research question RQ1.

- *Novel Approach for Dynamic Knowledge Management.* This thesis proposes a novel multi-level knowledge management approach for self-aware software systems. The presentation of the approach is steered by the VC services selection scenario. The approach uses dynamic histograms [23] to store the evolving knowledge so that the performance patterns of the volunteer services are captured. The multi-level knowledge management approach is able to separate the knowledge related to stimulus, historical performance of the services, and the historical interactions between the services. We develop the algorithms related to the dynamic histograms update and maintenance as well as services selection and adaptation approaches. This contribution deals with the research question RQ2.
- *A novel approach for meta-level adaptation.* This thesis proposes a novel approach for self-adaptation at the meta level. Such adaptation is represented as the switching between the different levels of awareness based on the advantages and disadvantages of the levels taking into consideration the current state of the system in terms of the stability of the system's queue. The meta-level adaptation approach makes novel use of a symbiotic simulation – in the heart of the self-adaptive process – to provide the basis for what-if analysis in cases where the knowledge for adaptation is stringent. This contribution deals with the research question RQ3.
- *Application of the self-awareness to a timely computing paradigm, the VC.* The results of the SLR revealed that although computational self-awareness has been introduced as an approach for fertilising self-adaptivity in highly dynamic environments, the scenarios which have been used to demonstrate this 'claim'

tend to show limited dynamism. Therefore, this thesis builds on the volunteer computing (VC) paradigm, which is a highly dynamic environment and lacks the existence of stringent SLAs. We review the representative VC systems. The objective of the review is to identify the self-adaptation capabilities of the systems and to draw conclusions on the need for self-awareness. The review results show limited adaptation capabilities of the current VC systems. In addition, the review identified gaps related to the services selection problem in the case of storage services. Thus, we contribute to a novel utility-based approach for informing the selection of the volunteer storage services, which is necessary to build the motivating scenario. This contribution deals with the research question RQ4.

- *Evaluation.* Using non-trivial simulations, we evaluate our approaches with different scales, e.g. the number of services and number of requests. The results demonstrate the advantage of our approach in terms of the ability to satisfy the user's requirements. This advantage is accompanied with computational overhead, which needs further investigation of the cost of accepting such overhead.

#### **1.4.2 Thesis-related Publications**

Publications arising from this thesis are [21], [24], [25], [26], [27], [28], and [29], which are respectively the following:

- Abdessalam Elhabbash, Maria Salama, Rami Bahsoon, and Peter Tino, "Self-Awareness in Software Engineering: A Systematic Literature Review", 2017, submitted to ACM TAAS.

- Abdessalam Elhabbash, Rami Bahsoon, and Peter Tino, and Peter R. Lewis "Symbiotic-based Meta-self-awareness for Self-adaptive Systems: A Case for Volunteer Services", 2017, submitted.
- Abdessalam Elhabbash, Rami Bahsoon, and Peter Tino. "Self-awareness for dynamic knowledge management in self-adaptive volunteer computing", *The 24<sup>th</sup> IEEE International Conference on Web Services (ICWS 2017)*, (Full paper in Research Track), Honolulu, Hawaii, USA, 2017, (Acceptance rate: 21%), to appear.
- Abdessalam Elhabbash, Rami Bahsoon, and Peter Tino. "Interaction-awareness for self-adaptive volunteer computing", *The IEEE 10th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, Augsburg, 2016, pp. 148-149. (Acceptance rate: 26%)
- Abdessalam Elhabbash, Rami Bahsoon, Peter Tino, and Peter Lewis. "Self-adaptive Volunteered Services Composition through Stimulus- and Time-awareness", *The 22<sup>nd</sup> IEEE International Conference on Web Services (ICWS 2015)*, (Full paper in Research Track), New York, USA, 2015. (Acceptance rate: 17.4%)
- Abdessalam Elhabbash, Rami Bahsoon, Peter Tino, and Peter Lewis. "A Utility Model for Volunteer Service Composition". *The 7<sup>th</sup> ACM/IEEE International Conference on Utility and Cloud Computing (UCC 2014)*. Full paper, London, UK, 2014. (Acceptance rate: 19%)
- Abdessalam Elhabbash, Rami Bahsoon, and Peter Tino. "Towards Self-aware Service Composition". *In Proceedings of the 16th IEEE International Conference on High Performance Computing and Communications (HPCC 2014)*, Paris, France, 2014. (Acceptance rate: 25%)

### 1.4.3 Thesis Roadmap

Figure 1.1 shows the roadmap of this thesis. The structure of the rest of this thesis is as the following:

- *Chapter 2* conducts a systematic literature review on self-awareness in software engineering. It aims at building the required background and understanding of the field, exploring the current approaches, and identifying the gaps in the literature. This chapter is partially derived from [21].
- *Chapter 3* presents a background about the volunteer computing paradigm and reviews the representative VC systems. The review identifies the limitations of the autonomic behaviour of the systems and motivates the need for self-awareness. Also, the review reveals inadequacies in the selection approaches of the volunteer resources in terms of resources utilisations and requests satisfaction. This chapter is partially derived from [28].
- *Chapter 4* draws on the conclusions of chapter 3. The chapter develops a volunteer storage scenario and proposes a utility model that informs the selection of the volunteer storage resources. The chapter compares the utility-based selection approach with two basic approaches. This chapter is derived from [28].
- *Chapter 5* proposes the multi-level self-aware approach for dynamic knowledge management. Benefiting from existing analysis of the performance of the volunteer hosts, the volunteer services selection scenario is used to steer the presentation of the approach. An experimental evaluation is conducted to show

the pros and cons of the proposed approach compared to the basic adaptation approach used in VC. This chapter is derived from [29] [25], [27], and [26].

- *Chapter 6* proposes an approach, namely *meta-self-awareness*, for switching between the levels of self-awareness according to the system state and the usefulness of each of the levels. The chapter introduces the symbiotic simulation paradigm and reasons about using it in the meta-self-awareness approach. The chapter compares the meta-self-awareness with the non-meta-self-awareness approaches. The results show that the meta-self-awareness approach selects the optimal awareness level. This chapter is derived from [24].
- *Chapter 7* reflects the findings of the thesis on the research questions, concludes the outcome of the thesis, and discusses the open issues.

In addition, we find it useful to highlight the following terms to make them clear to the reader, as they frequently appear in the thesis:

- *Fine-grained vs coarse-grained* knowledge. Treating knowledge at fine-grained levels means taking into consideration that different insights can be extracted from the collected data. That means, considering different concerns of the collected data can support structuring the knowledge into multiple levels where each level can provide a different type of knowledge (e.g. stimuli, historical performance, and interaction). The ignorance of the potential structure of the knowledge is considered *coarse-grained* treatment.
- *Highly-dynamic environment*. In general, current computing environments (e.g. the Cloud, services computing, and VC) are dynamic. However, the dynamism of environments like VC can be more challenging due to the lack of strict SLAs that

oblige the providers to provide the resources. We use the term high-dynamic to highlight such cases, which can benefit from self-awareness.

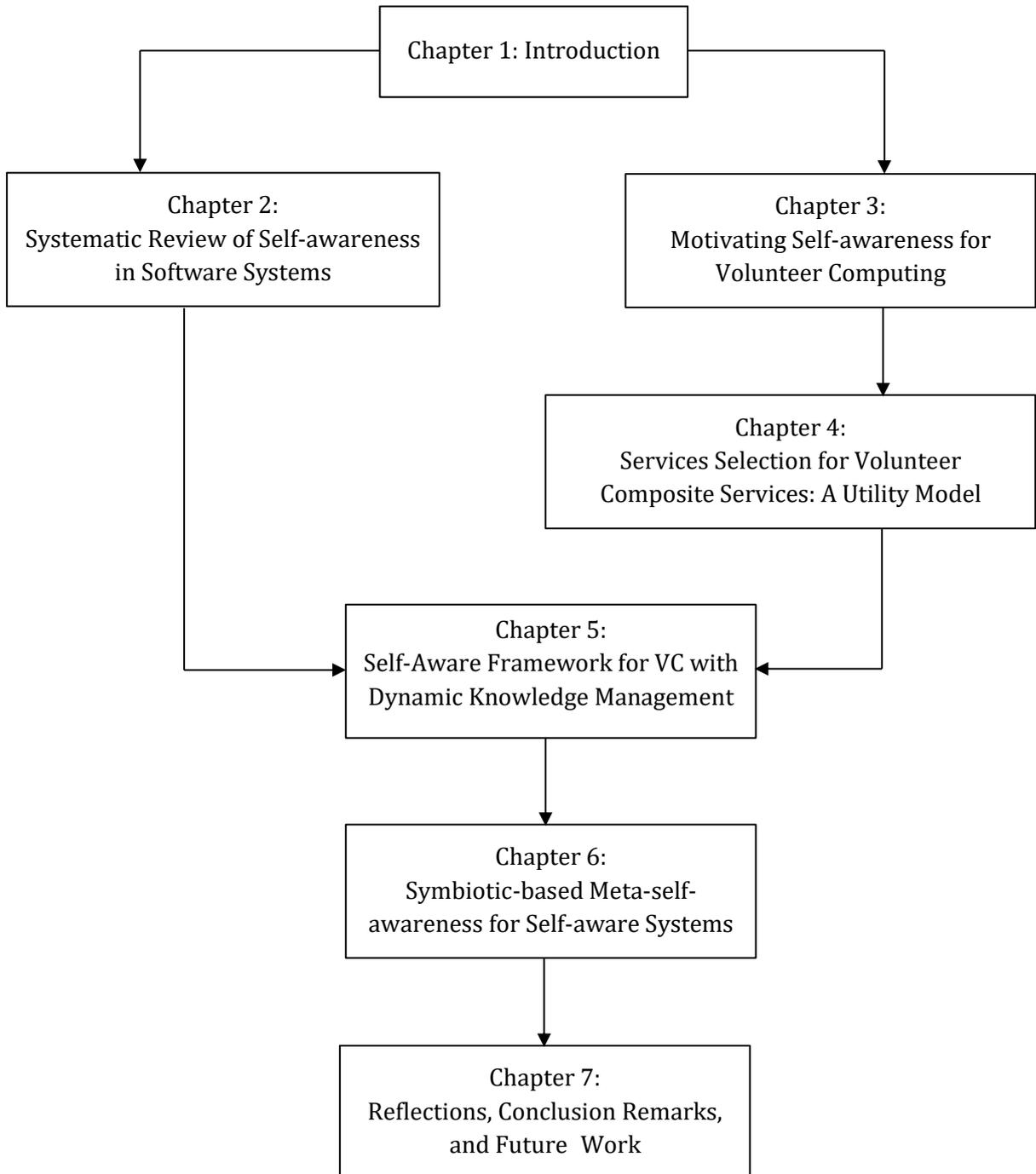


Figure 1.1: Roadmap of the Thesis



# CHAPTER 2

## SYSTEMATIC REVIEW OF SELF-AWARENESS IN SOFTWARE SYSTEMS

### 2.1 Overview

This chapter introduces a background on self-adaptive systems and investigates the motivations behind adopting self-awareness to empower self-adaptivity in software systems. After that, the chapter surveys the landscape of self-aware software systems to review the state of the art and identify challenges and open problems. The objective of the review is to verify and provide a better understanding of the research questions that have been posed in chapter 1. More specifically, the chapter reviews the solutions and advances that have been done in self-awareness and how they can benefit the case of dynamic open systems such as VC. Another objective is to investigate the limitations of the current solutions so that we can attempt to provide enhancements that benefit our case.

We follow the guidelines for conducting SLRs proposed by Kitchenham [30]. The SLR guidelines aim at documenting the steps of the review process. This enables assessing the search protocol and results by re-executing the search protocol, e.g. by an independent assessor.

The main findings of the review show that there is a growing attention to incorporate self-awareness for better reasoning about the adaptation decision making in autonomic systems. The motivation behind leveraging computational self-awareness is to provide the system with the required knowledge to handle the complexity of the systems. Yet, the current works have given little attention to realising self-awareness with dynamic knowledge management. In addition to that, the review indicates that the usefulness of self-awareness needs to be demonstrated more sufficiently. The reason can be that the environments in which self-awareness has been applied possess some controls that, to some extent, restrain the environment dynamics. Addressing these pending issues is likely to better inform the self-adaptivity and provide better seamless management of the system, taking into consideration the core properties which relate to the heterogeneity, uncertainty, and dynamism.

## **2.2 Self-adaptation in Dynamic Software Systems**

Self-adaptive software systems are empowered by the capabilities of autonomously adjusting their attributes and/or structures in response to the internal and external changes. While internal changes arise from the system itself, e.g. unexpected failure, external changes arise from the operating environment, e.g. a sudden increase in the workload. In order to be able to adapt, a self-adaptive system incorporates a feedback loop system to continuously feed information about the current state of the system and

the environment. The fed information is analysed at runtime to assess whether the system is able to meet the intended requirements. If it is not the case, then an adaptation action will be taken to adapt the system behaviour [31].

There is a unanimity that the increased cost and complexity in dealing with such situations at runtime by humans raised the need for self-adaptive systems. Adaptation based on humans' supervision is unreliable, slow, expensive, and error-prone due to the imperfect nature of humans and their limited ability to react in a timely manner, and also due to the complexity of the system itself [32]. Based on that, self-adaptation has received a considerable level of interest in different research areas like Cloud Computing, Service Computing, Robotics etc. Researchers have made significant efforts to design self-adaptive systems which resulted in a considerable literature, which has been explicitly discussed in many surveys, e.g., [33] [9] [34] [35]. The representative works are presented in the following.

In [10] [33] the authors introduced six questions to address the requirements of self-adaptive systems, namely, "*Who has to perform the adaptation?*", "*When to adapt?*", "*Where do we have to implement change?*", "*What kind of change is needed?*", "*Why do we have to adapt?*", and "*How is the adaptation performed?*". The answer of the *Who* question is obvious; the system needs to adapt autonomously, ideally, with no human interventions. The *When* question addresses the temporal aspects of adaptation in terms of the time of applying the change decisions. The mentioned types of adaptation are reactive adaptation and proactive adaptation. Each of the types has its pros and cons in terms of benefits and overheads. The reactive adaptation is to apply the adaptation decisions after a need for adaptation arises, e.g. after a violation of the SLA occurs. The

proactive adaptation is to apply the adaptation decision before the need arises in order to avoid undesirable situations, which requires the forecast of the system and the environment states. The *Where* question addresses the level at which the adaptation applied. For example, in a cloud-based system, this can be the application level, architecture level, or the infrastructure level. The *What* is concerned with the kind of the required change [33], which can be adapting parameters, components, or context. The *Why* question is concerned with the feasibility of applying the adaptation action - in terms of its necessity to achieving the system's and users' goals. The *How* question is about the plan of adaptation. That is, which actions should be taken to apply the adaptation; taking into consideration the benefits and costs of each adaptation actions.

In 2001, IBM released a conceptual architecture to engineering self-adaptive systems, namely, the MAPE-K [7]. According to this architecture, a self-adaptive system contains basically of two elements, namely, the *managing* element and the *managed* element. The managing element is the autonomous part of the self-adaptive system, which is responsible for adapting the managed element behaviour. For achieving such autonomous behaviour, the managing element basically adopts a closed-loop controller that involves five main components, namely, *Monitor*, *Analyse*, *Plan*, *Execute*, and *Knowledge*. The *Monitor* component is responsible for gathering information about the system and the environment states. Typically this is achieved by implementing different sensors as the need entails. The gathered information is passed to the *Analysis* component which analyses the received information in order to investigate whether the system and/or the environment states call for adapting the managed element. In the case of detecting a change that calls for adaptation, the *Analysis* component reports to the *Plan* component which constructs the system strategy in the new situation. The

planned strategy involves the actions needed to be undertaken to alternate the managed element so that the system goals are achieved. After that, the *Execute* component executes the recommended plan in order to change the managed element. The above four components share access to the *Knowledge* component which stores the information (e.g. topology, available services, performance logs, etc.) fed and updated by the sensors and the *Execute* component.

The MAPE-K architecture [7], introduced above, represents the conceptual architecture which inspired many other works that built on and extended the MAPE-K. In [36] Garalan et al. proposed the RAINBOW framework for engineering software systems. The framework addresses the problems of generality and adaptation cost reduction by dividing the software system into three layers, namely, the *architecture* layer, the *system* layer, and the *translation* layer. The *architecture* layer contains reusable components that provide common functionalities and hence can be reused across different systems to define the adaptation plans. The *system* layer provides system-specific mechanisms to monitor and report the system state to the translation layer. The *translation* layer controls the mapping of the monitored state and the adaptation plans among the system and the architecture layers.

In [37] a three-layer architecture has been introduced to self-adaptive software systems. The bottom layer, *Component Control*, monitors the current state of the system and effects actions in response to known detected changes. When the changes are new, the *Component Control* layer reports the changes to the middle layer, the *Change Management*. The *Change Management* inspects the predefined adaptation plans that can be applied in response to the new changes and effects the plan to the *Component*

*Control* to apply the corresponding actions. In the case of no plans exist to treat the situation; the *Change Management* reports the case to the upper layer, the *Goal Management*. The *Goal Management* then produces new plans which are injected to the *Change Management*.

Another framework inspired by the control theory is the Observe-Decide-Act (ODA) loop [38]. The *Observe* component is concerned with monitoring both the system and the environment. The *Decide* component is concerned with making adaptation decisions using a set of available actions. Then the *Act* component executes the selected adaptation action.

In [39] Elkhodary et al. present a learning-based framework called 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. FUSION realises the self-adaptability of the system through two cycles, namely *Learning* cycle and *Adaptation* cycle. In the former, the framework observes the behaviour of the features and estimates the impact of the feature's selection, and then stores the learnt models in a knowledge base. In the later, the system is capable of detecting the violation of goals, planning adaptation strategies using the knowledge base, and effecting the adaptation.

### **2.2.1 Limitations of Self-adaptive Systems**

In spite of the existence of the considerable literature of self-adaptive systems, these systems exhibit limited self-adaptation capabilities due to the following reasons [40] [41]:

- Most self-adaptive systems lack the awareness about the implicit effects of the adaptation decisions taken at runtime resulting in limited capabilities in facing the continuous changes and meeting the users' and system's quality requirements.
- Proactivity and the anticipation of adaptation action have been limited in self-adaptive systems due to the lack of treatment of knowledge; limiting the capability of avoiding violations and reducing interruptions.
- Knowledge representation has been considered at a coarse-grained level which limits the quality of adaptation. Gaining in-depth knowledge on system performance and representing that knowledge at a fine-grained level helps taking more intelligent adaptation decisions. For example, in service-based applications, representing knowledge on the performance of the services helps to select the services that perform well. However, representing the knowledge on the interactions between the services helps to select services that perform well when composed together.

Such limitations of the self-adaptive systems resulted in the emergence of the concept of 'self-awareness', which has been receiving more attention in computing systems. Over the past ten years, researchers have been proposing approaches for adopting self-awareness as an enabler for self-adaptation in the dynamic environments. The purpose is to enrich the self-adaptation capabilities by gaining 'richer' knowledge of the system and the environment's current and future states. The concept of self-awareness in computing systems has been demonstrated by some representative efforts, e.g. the EU Proactive Initiative Self-Awareness in Autonomic Systems [16] and

road-mapping agenda of the Dagstuhl seminar [17], to develop a fundamental understanding of the computational self-awareness. In the next section, we conduct a systematic review on self-awareness in software engineering in order to compile the related studies.

## **2.3 Self-Awareness in Software Engineering: A Systematic Literature**

### **Review**

A systematic literature review is a methodical process to build a body of knowledge on a certain subject or topic [42]. In this section, we aim at building knowledge on the computational self-awareness in software systems. We focus on the studies that have an explicit claim of using self-awareness in software engineering. The review explores and investigates (i) how the studies have defined and characterised self-awareness, (ii) what sources have inspired researchers and motivated them to use self-awareness, (iii) what software engineering practices and software paradigms have employed self-awareness, (iv) how computational self-awareness is engineered to encode self-awareness within software systems, and (v) the evaluation approaches of the proposed self-aware systems to examine the accompanied benefits and overheads.

#### **2.3.1 Review Protocol**

In this section, we describe the research method used to conduct this systematic review. The procedure of this study followed the guidelines for conducting systematic literature reviews [42]. The process has also been informed by other reviews relevant to software engineering [30] [43] [44]. The review process includes three main phases: (i) planning, (ii) conducting, and (iii) reporting the review. We first define the research questions that

drive our research, and then we describe the planned protocol to be followed for conducting the review. The research methodology is depicted in Figure 2.1.

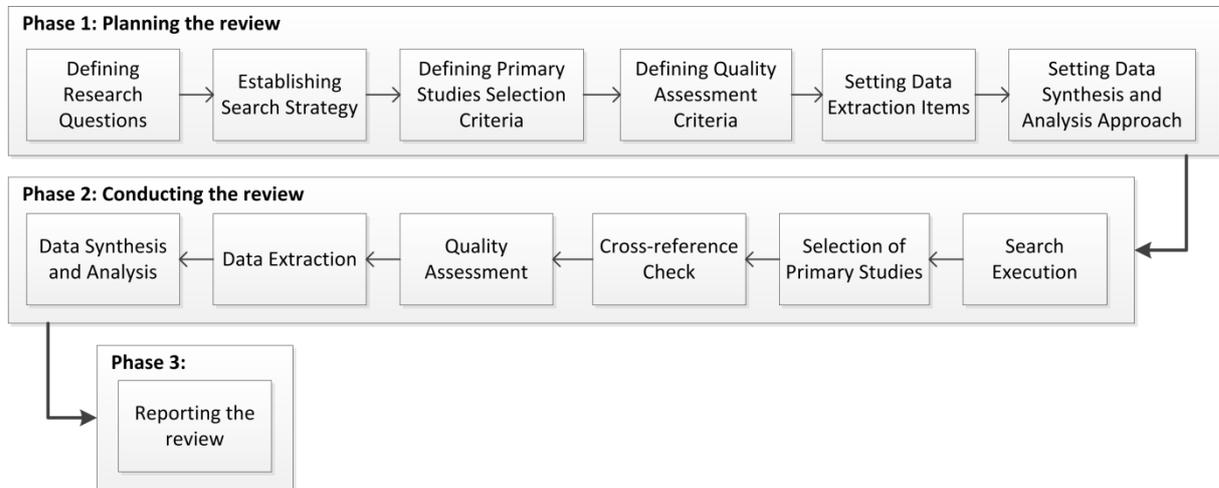


Figure 2.1: Research Methodology

### 2.3.1.2 The Review Research Questions

The overall research objective of the review is to give an overview of the current state-of-the-art related to self-awareness in software engineering in research and practice. The review surveys the existing approaches and paradigms to investigate how those works tackled self-awareness. It also aims to identify the potential issues and limitations in current research. In this context, in order to inform our work and obtain a better understanding of the thesis research questions, we conduct the review using the following questions:

- **Q1. How to define and characterise self-awareness?**

This question is motivated by the need for defining and consequently characterising self-awareness in software systems; i.e. how to consider that a

system is self-aware. Answering this question supports understanding the thesis RQ1 by investigating the characteristics that the self-aware system should possess.

- **Q2. What did motivate the application of self-awareness in software engineering and what are the sources of inspiration for its engineering?**

This question aims to identify the motivations and the sources of inspiration that stimulated the adoption of computational self-awareness in software engineering. The sources of inspiration for engineering self-awareness are also identified, in order to investigate how these sources helped to advance self-aware software systems. Answering this question supports understanding the thesis RQ1 by investigating the reasons behind adopting self-awareness.

- **Q3. In which software engineering practices and software paradigms is self-awareness employed?**

The goal of Q3 is to find the software paradigms that employed self-awareness and to explore the characteristics of the environments that can benefit from computational self-awareness. This question is related to the thesis RQ4, as we leverage from the existing application of self-awareness to inform the case of VC that we are using to steer the presentation of our approach.

- **Q4. What are the approaches for engineering self-awareness?**

This question identifies the approaches of engineering self-aware software systems and investigates how computational self-awareness has been realised. In other words, this question investigates the engineering of knowledge management in current self-aware approaches. This question supports the understanding of the thesis RQ2 and RQ3 as it helps to find guidance to engineer

a framework for introducing self-awareness with dynamic knowledge management.

- **Q5. How are self-aware software systems evaluated?**

This question explores the significance of adopting self-awareness, in terms of performance evaluation, in order to inform the evaluation of our approach.

### **2.3.1.3 Search Strategy**

Our search strategy includes determining the data sources and the search string, as they appear in the sub-sections below.

**Data sources** The search process for this study is based on automated search in the following digital libraries and indexing systems that are considered as the largest and most complete scientific databases for conducting literature reviews in computer science [45] [46]:

- IEEE Xplore (<http://ieeexplore.ieee.org/>)
- ACM Digital Library (<http://dl.acm.org/>)
- ScienceDirect (<http://www.sciencedirect.com/>)
- Web of Science (<http://www.webofknowledge.com/>)
- SpringerLink (<http://link.springer.com/>)
- Wiley InterScience (<http://onlinelibrary.wiley.com/>)

As technical reports do not appear in these libraries, we considered Google Scholar (<http://scholar.google.co.uk/>) for this type of publications only.

Furthermore, we identified the most relevant conference proceedings and journals in the field of software engineering for manual search, according to our previous experience and the results obtained during trial searches. The full list of sources considered for the manual search is presented in Table 2.1.

Table 2.1: Manual Search Sources

Type	Data Source	Publisher
Journal	ACM Transactions on Software Engineering and Methodology	ACM
	ACM Transactions on Autonomous and Adaptive Systems	ACM
	IEEE Transactions on Software Engineering	IEEE
	Journal of Software and Systems	Elsevier
	Information and Software Technology	Elsevier
	Software and System Modelling	Springer
Conference	ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE)	ACM
	International Conference on Software Engineering (ICSE)	IEEE
	International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)	IEEE
	IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)	IEEE
Magazine	Software	IEEE
	Computer	IEEE
	Awareness Magazine: Self-Awareness in Autonomic Systems	European Commission (FP7)
	Dagstuhl Reports	Schloss Dagstuhl - Leibniz Center for Informatics

**Search String**      The aim of the search string is to capture all results related to self-awareness in the context of software engineering. Trial searches were performed in each database with the intention of checking the number of returned papers and their

relevance. The objective of the trial searches is to check the feasibility of the search string and adjust it accordingly.

The general search string used on all databases is: **(self-aware\*) AND (software)**. The first term captures the different ways self-awareness could be used, i.e. self-aware or self-awareness. The second term makes it explicit for software. The keywords **system(s)** and **computing** have returned a huge number of results related to computing systems, hardware, robots and networks. Other combined keywords, such as **software engineering and software systems** - when tried - had led to a vast wide set of irrelevant results. The simplicity and generality of the search string help in maximising the number of returned relevant papers, as it places as few restrictions as possible on the search string. We used the search string in the automated search engines of the data sources defined earlier, searching by meta-data (i.e. title, abstract and keywords).

Regarding the search in the specialised data sources, we first checked whether the papers published in these venues are retrieved in the databases included in the automated search. We found that manual search is needed only for the Awareness Magazine (<http://www.awareness-mag.eu/index.php>) and the Dagstuhl Reports ([http://drops.dagstuhl.de/opus/institut\\_dagrep.php?fakultaet=07](http://drops.dagstuhl.de/opus/institut_dagrep.php?fakultaet=07)).

#### **2.3.1.4 Studies Selection Criteria**

After the search has been executed, the study selection has been performed on the resulting set of studies. During the screening of search results, the title, abstract, introduction and conclusion for each candidate paper has been examined closely to determine the relevance of the paper. In some cases when these do not provide enough

information to decide the relevance of the paper, the whole paper has been read. The selection is to be performed with respect to the inclusion and exclusion criteria defined in Table 2.2. It is worth mentioning here that the inclusion of non-peer-reviewed (e.g. technical reports) is motivated by our understanding that the field of computational self-awareness is emergent. This supports our goal of understanding the advances made in engineering self-awareness by capturing the relevant outcomes of the relevant projects that addressed self-awareness.

When similar studies are reported in several papers as work-in-progress, the most comprehensive version is to be considered, unless significant details were reported in the earlier version.

Table 2.2: Studies Selection Criteria

<b>Inclusion Criteria</b>	
<b>I1.</b>	Papers published in conferences and journals, as full research paper, short and position paper presenting new and emerging ideas, as well as doctoral symposiums
<b>I2.</b>	Literature published as books, book chapters and technical reports
<b>I3.</b>	Papers employing self-awareness concepts in engineering software systems (e.g. cloud-based, service-oriented)
<b>I4.</b>	Papers implementing or extending self-awareness concepts
<b>I5.</b>	Papers discussing general or particular aspects of self-awareness
<b>Exclusion Criteria</b>	
<b>E1.</b>	Papers not in the form of a full research paper, i.e. in the form of abstract, tutorials, presentation, or essay.
<b>E2.</b>	Papers with abstract not available.
<b>E3.</b>	Papers not written in the English language.
<b>E4.</b>	Papers focusing on awareness or context-, situation-awareness or any other form of awareness (i.e. not self-awareness), or not explicitly addressing self-awareness.
<b>E5.</b>	Papers not focusing on self-awareness in software engineering; i.e. other computer science fields, such as networking or robotics or hardware.

### **2.3.1.5 Cross-references Check**

In order not to miss any relevant studies, we designate the cross-referencing technique to find potentially relevant studies, by applying the ‘snow-balling’ search method [47]

[43]. This is performed by tracking the references contained in the ‘References’ section in each selected primary study [47] [43].

### 2.3.1.6 Quality Assessment Criteria

Primary studies are evaluated according to the quality assessment criteria shown in Table 2.3, in order to assess the quality of the studies under consideration. The quality assessment checklist is based on the assessment method for research studies proposed in [48] [44].

The scoring procedure is 1 if the quality item is present, 0.5 if it is partially present, 0 if not present or unknown. Based on that, the quality assessment score (maximum of 7) for a study is calculated by summing up the scores for all the quality items.

Table 2.3: Quality Assessment Checklist

<b>Quality item</b>	
<b>QA1.</b>	Problem definition of the study
<b>QA2.</b>	Reporting on background and context
<b>QA3.</b>	Description of the research method
<b>QA4.</b>	Evaluation of the research method
<b>QA5.</b>	Contributions of the study
<b>QA6.</b>	Reporting on the insights derived from the study
<b>QA7.</b>	Reporting on the limitations of the study and threats to validity

### 2.3.1.7 Data Extraction Items

For each selected primary study, the whole paper has been read to extract the data items, which will help in answering the research questions. Data items to be extracted and their relevant research questions are listed in Table 2.4.

### 2.3.1.8 Data Synthesis and Analysis Approach

Data synthesis involves collating and summarising data extracted from primary studies. In this stage, statistics are also extracted and the results are further analysed. For the

data synthesis, the extracted data should be inspected for similarities in order to define how results could be encapsulated. Our approach for synthesising findings will be based on the synthesis method ‘thematic analysis/synthesis’ [49], with the difference that instead of identifying themes derived from the findings reported in each primary study, we consider the synthesis and analysis targeted to answer the research questions.

Table 2.4: Data Items Extracted from Primary Studies

<b>Data item</b>	<b>Description</b>	<b>Relevant SLR question</b>
BibTeX key	a unique key identifying the study for reference	Documentation
Title	title of the study	Documentation
Year	publication year	Demographics
Authors’ affiliations	the affiliations of all authors as appearing in the study	Demographics
Affiliation Countries	the countries of the authors’ affiliations	Demographics
Definition	definition of the self-awareness concept	Q1
Characteristics	characteristics to consider a software as a self-aware one	Q1
Motivation	the motivation for employing self-awareness	Q2
Source of inspiration	what inspired the self-aware approach	Q2
Software Engineering Practices	the software engineering practices that employed self-awareness; i.e. requirements engineering, architecture design	Q3
Software paradigm	which types of software considered self-awareness; i.e. cloud-, mobile-, service-based	Q3
Engineering Approach	what is the approach used to realise self-awareness; i.e. prediction, machine learning	Q4
Evaluation tool	how the self-aware system is evaluated; i.e. simulation, experiments	Q5
Performance	how the self-aware system performed compared with non-self-aware	Q5
Overhead	what is the overhead caused by self-awareness	Q5

### 2.3.2 Conducting the Review

This section summarises the execution of the review protocol.

### **2.3.2.1 Search Execution**

The search was executed according to the search strategy defined in section 2.3.1.3. In practice, particular settings were built for each search engine (details in Table 2.5), since each digital library works in a specific manner. This was attempted to minimise duplications and rejections by setting the appropriate options in each search engine. Particularly, filters were applied - when available - for setting the search engine to retrieve only studies published by its own engine or to retrieve documents in English language only. Minimising results by excluding irrelevant disciplines was also used, whenever available. In cases where the search engine does not imply enough filters and a large number of irrelevant results were retrieved, we used the first sets of search results sorted by relevance. This decision was made after a careful checking of the next set of search results to ensure their complete irrelevance.

During the course of executing the search, we used a spreadsheet to keep track of the search execution process and perform quantitative analysis on the results. This spreadsheet contains:

- Data source - the name of the data source;
- URL - the URL of the data source;
- Search Query - the query string as entered to the search engine;
- Search filters - further filters used to refine the search results (e.g., language, discipline);
- Search results - the total number of search results retrieved;
- Considered results - the total number of search results considered for primary studies selection;

- Search results file - the bibliography file of the search results;

Table 2.5: Search Execution (Search Strings and Settings)

Database	Search string	Search settings
ACM Digital Library	“self-aware*” AND software	N/A
IEEE Xplore	“self-aware*” AND software	refined by Publisher: IEEE
ScienceDirect	“self-aware*” AND software	Publications titles: - Procedia Computer Science - Journal of Systems and Software - Future Generation Computer Systems - Expert Systems with Applications - Science of Computer Programming - Computer Standards & Interfaces - Decision Support Systems - Journal of Network and Computer Applications
Web of Science	“self-aware*” AND software	Language: English Categories: - Computer Science Theory Methods - Computer Science Information Systems - Computer Science Software Engineering - Computer Science Interdisciplinary Applications
SpringerLink	“self-aware*” AND software	Discipline: Computer Science SubDiscipline: SWE Language: English
Wiley InterScience	self-aware* AND software	N/A
Google Scholar	“self-aware*” AND software	N/A

Search results were extracted as a bibliography in BibTeX format, having a final collection of bibliographies for each data source. We have also created a spreadsheet listing the search results with their meta-information. We, then, used JabRef [50] to merge the search results files into one *.bib* file after detecting and removing duplicates.

As a result of the automated search execution, we found 47,787 studies in total, without the Awareness Magazine and Dagstuhl Reports (where we performed a manual

search for all articles). In case a large number of results was retrieved, we used the first sets of search results sorted by relevance. More specifically, we considered the first 200 results from SpringerLink and the first 100 results from Google Scholar sorted by relevance. Table 2.6 shows the total results of automated search execution in each data source and the considered results.

Table 2.6: Automated Search Results

<b>Database</b>	<b>Search results</b>	<b>Considered results</b>
ACM Digital Library	48	48
IEEE Xplore	73	73
ScienceDirect	80	80
Web of Science	56	56
SpringerLink	30,430	200
Wiley InterScience	0	0
Google Scholar	17,100	100
Total	47,787	557
Total after removing duplicates	-	532

The end results of the automated search execution are 557 studies to be considered with 25 duplicates. We then performed primary studies selection on the 532 candidate studies and all the articles published in the two specialised data sources considered for manual search; i.e. 51 articles in the Awareness Magazine and 6 volumes (with 12 issues each) of the Dagstuhl Reports.

### **2.3.2.2 Selection of Primary Studies**

Selection of primary studies was performed using the inclusion and exclusion criteria (defined earlier in Table 2.2). We used a spreadsheet to collect data related to this stage.

This spreadsheet contains:

- Selection - whether the study is selected or not;
- BibTeX key - a unique key identifying the study for reference;

- Title - title of the study;
- Year - publication year;
- I1 - I5 - whether the study fulfils the inclusion criteria;
- E1 - E5 - whether the study fulfils the exclusion criteria;

This step results in 33 selected primary studies.

### **2.3.2.3 Cross-references Check**

Cross-references check was performed on the References section of each selected primary study. Bibliography data about every cited reference was collected, similar to the search results. We collected 712 new studies. Then, we performed the same study selection process as described in section 2.3.1.4. This results in 13 more studies after removing duplicates. The final set of primary studies includes 46 studies.

### **2.3.2.4 Quality Assessment**

We performed quality assessment check on the 46 collected studies, according to the criteria defined earlier in section 2.3.1.6. Figure 2.2 shows the number of studies with different scores for each quality assessment criterion. The results show that researchers explicitly provide descriptions of the problem they tackle (QA1), report on background and context (QA2), as well as a description of the research method (QA3). Evaluation of the research method (QA4) and insights (QA6) are also reported, but not always explicitly. This reflects that the evaluation of self-aware software systems needs to receive more attention, as we will discuss in the next section. Not the majority of the studies reported significant contributions to self-awareness (QA5). This reflects the need for clear vision and roadmap for self-awareness in the community. Most studies ignore reporting limitations of the results and threats to validity (QA7). This could

reflect some weakness to the studies addressing self-awareness in software systems. However, reporting the limitations deserves attention, as this should be part of any research study.

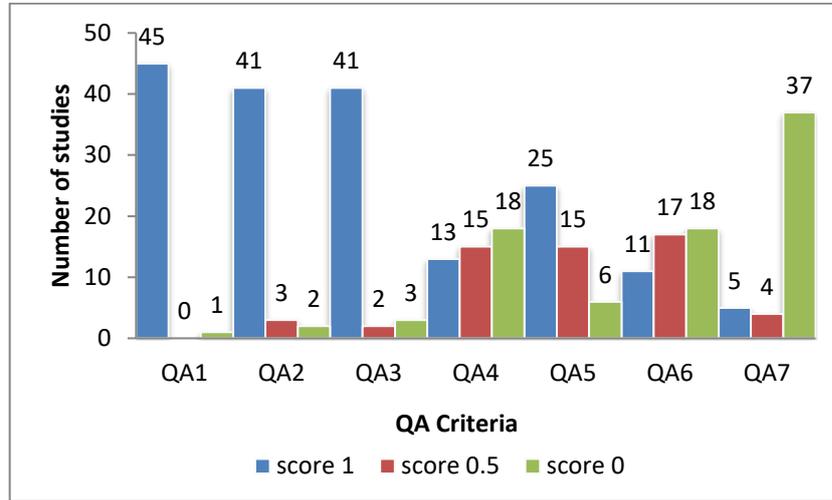


Figure 2.2: Quality Scores for the Primary Studies

The results of the quality assessment show that the average quality score is 4.54. The number of studies with respect to different score ranges is shown in Table 2.7. The quality score for the majority of studies (33 studies) ranges between 4.0 and 6.0 with an average of 4.71. A small percentage of studies highly scored with an average of 6.6 and another small percentage had a score of 3.5 or lower with an average score of 2.56.

Table 2.7: Quality Assessment Total Results

	Number of studies	Percentage	Mean score
score $\geq$ 6.5	5	17.4	6.6
score 4.0-6.0	33	71.7	4.71
score $\leq$ 3.5	8	10.9	2.56
<b>Average QA score</b>			<b>4.54</b>

Generally, the mean quality score of the majority of studies ranging between 4.0 and 6.0 could be attributed to the quality criteria that were scored low for a large number of studies. These include the lack of strong evaluation (QA4), reporting on future insights (QA5), and reporting on limitations and threats to validity (QA7).

### 2.3.2.5 Data Extraction

Finally, we performed data extraction and synthesis on the primary studies. For each study, data items defined earlier in section 2.3.1.7 were extracted and recorded in a spreadsheet.

### 2.3.2.6 Data Synthesis

We constructed a thematic map from the data extracted from the primary studies. The thematic map is depicted in Figure 2.3. Further results of data synthesis and analysis are reported in the next section with respect to each of the SLR research question.

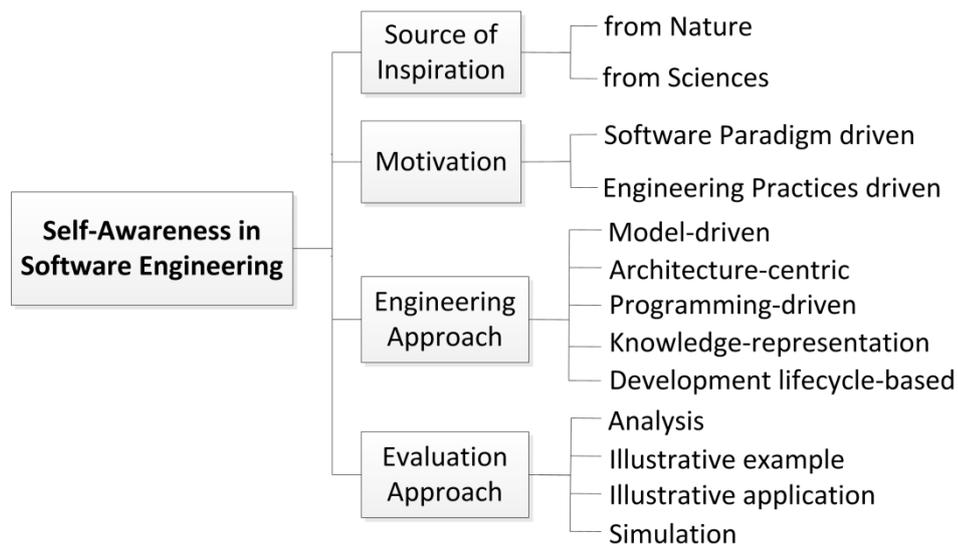


Figure 2.3: Thematic Analysis of Self-Awareness in Software Engineering

### 2.3.3 Reporting the Review

In this section, we present an overview of the selected studies, as well as the findings and answers to the SLR research questions.

#### 2.3.3.1 Overview of Primary Studies

This section describes the primary studies with respect to their publication types and year. Research communities that are active in the field of self-awareness are also presented.

All the primary studies were published in journals, conferences or seminal books that belong to well-established data sources in the software engineering community, as defined in the search strategy in section 2.3.1.3. Most of the studies fulfilled the criteria for quality assessment above average as described in section 2.3.2.4. These represent the degree of high quality and potential impact of the selected studies and provide confidence in the overall quality of the systematic review.

**Publication Types** As shown in Figure 2.4, a significant number of studies were published in conference proceedings (63%), followed by a smaller number of publications (19%) in journals. A limited number of publications were published as book chapters (9%) and technical reports (11%). Ideas and solutions are still being proposed in conferences, and some of them have matured and reported through journals and books. This indicates that research in this area is still considered maturing. The presence of a number of technical reports reflects the transition between research and practice, as well as the technical work done in this area.

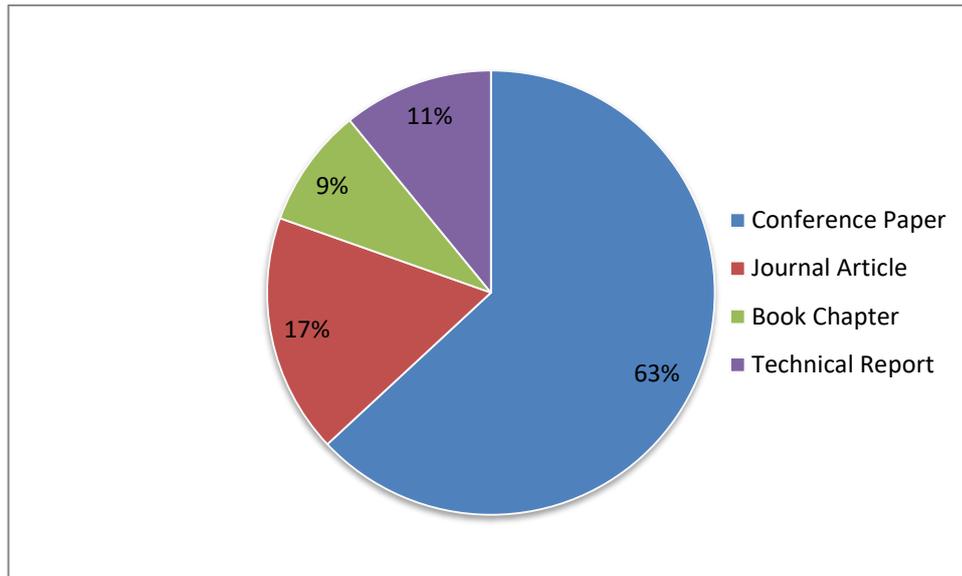


Figure 2.4: Distribution of Primary Studies over Publication Types

**Publication Years** Checking the distribution of publications over years as shown in Figure 2.5, it is noticed that the interest on self-awareness has started on 2005, with the exception of very few studies scattered over years starting 1997. As defined in the search strategy in section 2.3.1.3, we did not set filters on the publication year, yet the time frame of the studies reflects the time frame of interest and advancements in self-awareness. Following the year of 2005, the number of publications is increasing, though it is not constant increase over years. Note that the search process has covered only publications for the first two quarters of 2016. The increase in the number of publications indicates that self-awareness has taken its place as the next property among self-\* properties, as self-adaptivity and software systems are becoming more complex.

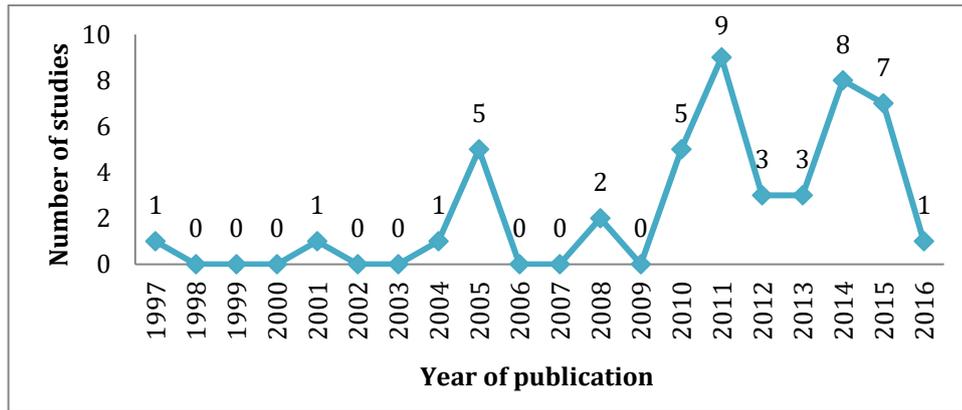


Figure 2.5: Number of Publications per Year

**Active Research Communities** To identify the active research communities within the area of self-awareness, we look at the affiliations that appeared in the publications. Table 2.8 summarises the active research communities (with at least two publications in self-awareness) along with the number of publications. Publications are mostly dominated by University of Birmingham UK, University of Modena and Reggio Emilia Italy and Polytechnic University of Milan, Italy (note that we follow the authors' affiliations as appeared at the time of publication, and some studies appear multiple times under different affiliations).

Table 2.8: Active Research Communities within Self-Awareness

Affiliation	Number of studies
University of Birmingham, UK	9
University of Modena and Reggio Emilia, Italy	7
Polytechnic University of Milan, Italy	5
Karlsruhe Institute of Technology, Germany	3
Massachusetts Institute of Technology, USA	3
Irish Software Engineering Research Centre, Ireland	3
University of Oslo, Norway	2
Volkswagen AG, Germany	2
Aston University, UK	2
University of Würzburg, Germany	2
Vienna University of Technology, Austria	2

Analysing the demographic distribution of the researchers by their affiliation countries, Figure 2.6 illustrates the distribution of this analysis. This shows that self-awareness research is receiving the highest attention in Italy, USA, UK, and Germany.

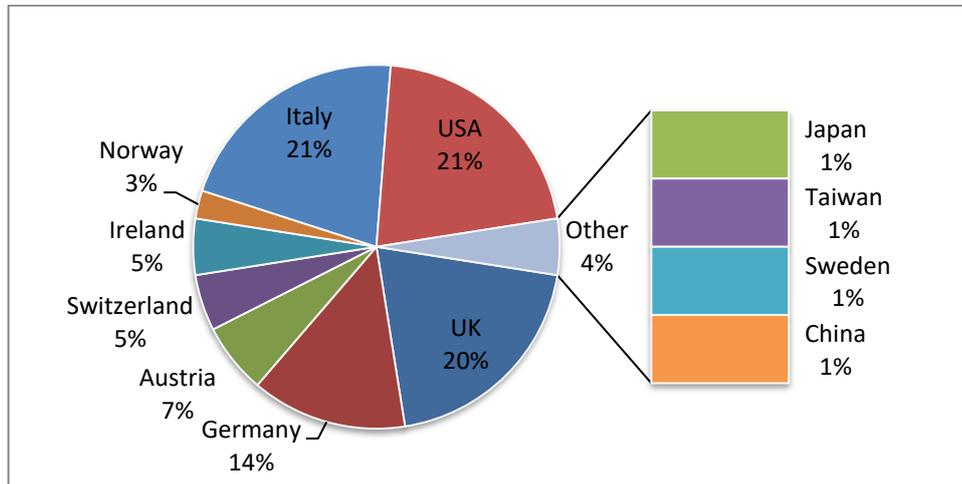


Figure 2.6: Distribution of Publications by Affiliation Country

### 2.3.3.2 Defining and Characterising Self-Awareness in Software Engineering (Q1)

This question looks at the definition of self-awareness as provided in the primary studies. We found that there is no general agreement on the definition of the computational self-awareness. Some authors provided an explicit definition for self-awareness based on how they view this concept in software engineering or computation in general. Others used the term interchangeably with the term 'self-adaptive', i.e. according to their view, a self-aware system is a self-adaptive system and vice versa. Table 2.9 lists the explicit definitions of self-awareness found in the primary studies.

Table 2.9: Definitions of Self-Awareness

Study	Definition
[51]	<i>"Self-awareness is the ability of an element to autonomously detect deviations in its behaviour that are meaningful."</i>
[52]	<i>"Systems are self-aware if they have an information subsystem which generates an adaptive self-model of the system providing reference for identity check communications. In other words, self-awareness implies the evolution of an information sub-system in the first place, and evolution of particular properties of this information subsystem."</i>
[53]	<i>"Self-awareness is information contained in a system about its global state that feeds back to adaptively control the system's low-level components."</i>
[54]	<i>"By self-awareness I am referring to an awareness of one's own thought processes along with the insight that those thought processes can be captured, conceptualised, and named — and when applied to software, externalised as code."</i>
[55]	<i>"Self-aware computer systems will be able to configure, heal, optimise and protect themselves without the need for human intervention."</i>
[56]	<i>"A self-aware Cloud market is a market has the ability to change, adapt or even redesign its anatomy and/or the underpinning infrastructure during runtime in order to improve its performance."</i>
[13]	<i>"To be Self-Aware a node (component of a software system) must contain total information about its internal state along with enough knowledge of its environment to determine the current state of the system as a whole. It may either be focused on its own state or the environments state at any time, but not both at once."</i>
[57]	<i>"By self-awareness, we mean the ability of each node in the Cloud infrastructure to monitor the level of compliance to SLAs associated with the tasks under its control."</i>
[58]	A component or an ensemble of components is self-aware if it is <i>"able to recognize the situations of their current operational context requiring self-adaptive actions."</i>
[59]	<i>"Awareness is a product of knowledge and monitoring."</i>
[60]	<i>"The SOTA model identifies an n-dimensional virtual-state space in which the execution of a system situates. In the SOTA space, a system is self-aware if it can autonomously recognize its current position and direction of movement in the space, and self-adaptation means that the system is able to dynamically direct its trajectory."</i>
[61]	A self-aware computational node is defined <i>"as one that possesses information about its internal state and has sufficient knowledge of its environment to determine how it is perceived by other parts of the system."</i>
[62]	A self-aware system is defined as <i>"a system has detailed knowledge about its own entities, current states, capacity and capabilities, physical connections and ownership relations with other (similar) systems in its environment."</i>
[63]	<i>"A system that can be called aware should be able to sense or store at least some information about its environment or itself."</i>

Besides the definitions in Table 2.9, some research works ([64] and [65]) intended to characterise a self-aware software system using a set of sub-properties and they provided more comprehensive definitions.

According to [64], *“To be self-aware a node must:*

- *Possess information about its internal state (private self-awareness).*
- *Possess sufficient knowledge of its environment to determine how it is perceived by other parts of the system (public self-awareness).*

*Optionally, it might also:*

- *Possess knowledge of its role or importance within the wider system.*
- *Possess knowledge about the likely effect of potential future actions/decisions.*
- *Possess historical knowledge.*
- *Select what is relevant knowledge and what is not.”*

According to [65], self-awareness is considered by *“the combination of three properties that a system should possess:*

- *Self-reflective: Aware of its software architecture, execution environment, and hardware infrastructure on which it is running as well as of its operational goals,*
- *Self-predictive: Able to predict the effect of dynamic changes (e.g., changing service workloads) as well as predict the effect of possible adaptation actions (e.g., changing system configuration, adding/removing resources),*
- *Self-adaptive: Proactively adapting as the environment evolves in order to ensure that its operational goals are continuously met.”*

We extracted the different aspects and characteristics of self-awareness from the definitions cited above. We, then, analysed these definitions to show how each of the definitions found in the primary studies characterises self-awareness and how comprehensive they are. The characteristics are defined as follows:

- *Domain-specific*: determines whether the definition is restricted to the problem domain or is general to cross-cut different domains.
- *Behaviour*: determines whether the definition considers the behaviour of the system implicitly or explicitly.
- *Knowledge*: determines the aspects that the definition considers regarding the treatment of the knowledge, i.e. at a coarse- or fine-grained level. Fine-grained knowledge treatment means that the knowledge is structured into levels, which allows for different levels of adaptation. On the contrary, coarse-grained knowledge treatment does not consider such structure.
- *Internal State*: determines how the definition considers modelling the internal state of the system; implicitly or explicitly.
- *Environment*: determines how the definition considers modelling the environment state of the system; implicitly or explicitly.
- *Adaptation time*: determines what type of adaptation is supported by the self-aware system (according to the definition) in terms of the time to perform the adaptation; reactively (reacting to the incident after detecting it) or proactively (attempting to avoid the occurrence of an incident).

Table 2.10 shows the analysis of the definitions. It is worth noting here that the absence (declared using '-') of any of the characteristics in this table does not mean that

the corresponding work does not support that characteristic. It means that the definition or characterisation of self-awareness in that work does not explicitly mention that characteristic. That is, in such cases, inconsistency between the work and the definition may exist.

Table 2.10: Analysis of Self-Awareness Definitions

Study	Aspects of Self-Awareness					
	Domain	Behaviour	Knowledge	Internal State	Environment	Adaptation time
[51]	General	Explicit	Coarse	Explicit	-	-
[52]	General	Implicit	Coarse	Implicit	-	-
[53]	General	Implicit	Coarse	Explicit	-	-
[54]	General	Implicit	Coarse	Implicit	-	-
[55]	General	Implicit	Coarse	Implicit	-	-
[56]	Cloud	Implicit	Coarse	Explicit	Explicit	-
[13]	General	Implicit	Coarse	Explicit	Explicit	-
[57]	Cloud	Implicit	Coarse	Implicit	-	Reactive
[58]	General	Implicit	Coarse	Implicit	-	-
[59]	General	Implicit	Coarse	Implicit	Implicit	-
[60]	General	Implicit	Coarse	Implicit	-	-
[61]	General	Implicit	Coarse	Explicit	Explicit	-
[62]	General	Implicit	Coarse	Explicit	Explicit	-
[63]	General	Implicit	Coarse	Explicit	Explicit	-
[64]	General	Implicit	Fine	Implicit	Explicit	-
[65]	General	Explicit	Coarse	Explicit	Explicit	Proactive

It is notable from the surveyed literature that there is no comprehensive definition that covers all aspects of self-awareness. It is also worth to note that there is no unified distinction between self-aware and self-adaptive systems. Based on the studies considered in this review, most of the researchers use the two terms interchangeably. Among the studies considered in this review, the works of [64] and [65] are the only works that have clearly differentiated between the two terms. Both of them view self-aware systems as a sub-category of self-adaptive systems. The former considers a self-

adaptive system to be self-aware if the system defines multi-levels of knowledge modelling and representation and correspondingly supports different levels of self-adaptation. The latter considers the self-adaptive system to be self-aware if the system supports proactive adaptation.

It is also noteworthy that the majority of the definitions mention explicitly the act of obtaining knowledge. This indicates that knowledge management is central to self-awareness. However, more attention should be given to the in-depth knowledge acquisition and dynamic knowledge management, which will distinguish self-awareness systems from self-adaptive systems.

Based on the above, we propose the following definition of self-awareness in software systems; adapted from the definitions and characterisations of [64] and [65]:

*A software system is self-aware if it:*

- *possesses knowledge about its internal state and its environment,*
- *supports fine-grained knowledge management,*
- *able to capture the performance patterns of its components (internal and external),*
- *supports both autonomic reactive and proactive adaptation at different levels, and*
- *is able to predict the likely effect of the adaptation actions/decisions.*

### **2.3.3.3 Motivation and Inspiration for Employing Self-Awareness (Q2)**

This question looks at the motivation that derived the studies in employing self-awareness, as well as what inspired the self-awareness engineering process.

The majority of studies have clearly identified the motivation behind having self-awareness as a capability in software systems. Extracted from all studies, we have found

that the general motivation that has directed researchers towards self-awareness is the complexity, heterogeneity, scale of modern software systems, evolving functionality and quality requirements during run-time, emergent behaviours, and unpredictable changes in the highly dynamic operating environment [66] [67] [61] [68] [63].

More specifically, the motivation of employing self-awareness in software systems varied between a general one related to realising better autonomy for software systems and others that are more specific. With respect to the former, researchers considered self-awareness for: (i) reasoning and engineering better adaptations with guaranteed functionalities and quality of service during runtime [52] [69] [38] [64] [60] [66] [70] [71] [72] [73] [67] [63] [74], (ii) managing complex systems without human intervention [40] [75], (iii) dealing with real-world situations, operational contexts and dynamic environments of modern software systems to respond to such fluctuating environment and associated uncertainty [76] [66] [58] [77] [68] [63] [78], (iv) managing complex trade-offs arising from adaptation conflicting goals [79] and the heterogeneity of the system [61], and (v) realising intelligent software systems with sophisticated abilities [53] [64] [80] [59].

Specific motivations varied between domain-specific according to the software paradigm (e.g. ubiquitous applications, pervasive services, cloud-based services) and others driven by software engineering practices (e.g., formal specification, performance management, data access). Table 2.11 summarises these motivations.

Unlike in the case of motivation, a few number of studies have clearly identified their source of inspiration in engineering self-awareness. Generally, nature and sciences inspired from nature are the main sources of inspiration in all studies. Examples of

Table 2.11: Specific Motivations of Using Self-Awareness

<b>Study</b>	<b>Motivation</b>
<b>Driven by Software Paradigm</b>	
[81]	Autonomous adaptations of hardware/software functionalities in ubiquitous computing applications to meet the dynamic requirements of various environmental situations and provide better QoS
[56]	Creating cloud markets platforms with self-* properties harmoniously working together in order to be able to adapt effectively to dynamic changes in user requirements, services, and variability in resources.
[82]	Modelling integrated pervasive services and their execution environments, in a way that diverse issues of context-awareness, dependability, openness, flexible and robust evolution, can be addressed
[83]	The need for runtime self-adaptive interactions between pervasive computing services
[84]	Achieving parallelism within a reasonable cost and time range for data streaming applications operating in distributed environments
<b>Driven by Engineering Practices</b>	
[85]	The motivation of including the notion of self within object-oriented formal specification languages is to facilitate reasoning about object interaction.
[51]	The detection of anomalies in the functioning of internet-based services and fault localisation (i.e., locating the responsible sub-services) is easier if service elements are aware of their own health status, determined by whether the current observed behaviour is consistent with expectations.
[86]	The need to access distributed and dynamic high-dimensional data about resources heterogeneity in a timely fashion in large, decentralised, resource-sharing environments
[54]	The invention of new abstractions as conceptualisation necessary to determine the behaviour of a software needed by users and the implementation details.
[87]	Enabling change at run-time for evolution purposes
[88], [89], [65]	The need to predict the performance of running services at run-time and related resources management.
[55]	Balancing resources usage in order to improve performance, utilisation, reliability and programmability
[68]	Solving problems caused by QoS interference in shared resources environment to achieve auto-scaling for cloud-based services
[19]	The complexity of managing end-to-end application performance

nature's inspiration include: biological systems [52] [53] [64], natural ecosystems [55] [82] [83] and human beings [52] [54] [80]. Sciences inspiring self-awareness are control

theory [38], biology [64], psychology [13] [66] [61] [18] and cognitive science [64].

Table 2.12 summarises inspirations cited in primary studies.

Table 2.12: Source of Inspiration for Engineering Self-Awareness

<b>Study</b>	<b>Inspiration</b>
<b>Inspiration from Nature</b>	
[52]	biological cell and the system of a human organisation (e.g., a company or government department)
[53]	biological systems: the immune system and ant colonies
[54]	human beings
[55]	biological organic nature
[80]	human wisdom
[82], [83]	natural ecosystems
<b>Inspiration from Sciences</b>	
[38]	Control Theory
[64]	Biology and cognitive science
[66]	Psychology, philosophy and medicine
[13], [61], [18]	Psychology

Within the studies mentioning their source of inspiration, we have found that the majority of studies named only their source of inspiration. More details, albeit in an abstract form, on how self-awareness approaches are inspired by nature or sciences are found in a few number of studies; such as [52] [53] [80] [13] [66] [83]. The exception that could be found is [64], where the authors have explicitly mentioned how self-awareness have been inspired from biology and cognitive science. The mapping between the source of inspiration and the research work conducted in the study is expected to be clearly communicated. Further, studies investigating how self-awareness could be inspired from nature and other sciences can help advance self-aware software systems.

### 2.3.3.4 Software Paradigms and Engineering Practices Realising Self-Awareness (Q3)

This question looks at the software paradigms that employed self-awareness and the software engineering practices that realised it.

Regarding the software paradigms found in the primary studies, we have found that the majority of studies considered self-awareness for autonomous computing; i.e. engineering self-adaptive software systems as a general software paradigm. Service-oriented systems and cloud-based services also received attention in a good number of studies, and less attention to ubiquitous computing. Regarding distributed systems, some studies considered a certain type of applications operating in decentralised environments; such as artificial intelligence systems [53], distributed smart cameras [66] [18]. Single works focused on software-intensive systems [59], and stream programming [84]. Table 2.13 summarises software paradigms found in the studies (note that some studies appear multiple times under different categories, which interprets the total number of studies appearing in the table is greater than the number of primary studies).

Table 2.13: Software Paradigms Employing Self-Awareness

Software Paradigms	Studies
Self-adaptive software systems	[90], [69], [52], [54], [87], [55], [38], [80], [79], [60], [66], [70], [75], [67], [73], [72], [62], [63], [19]
Service-oriented Systems	[51], [15], [88], [58], [77], [29], [65], [74]
Cloud-based services	[89], [56], [57], [66], [61], [18], [68], [78]
Distributed Systems	[86], [53], [66], [71], [18]
Ubiquitous Computing	[76], [81], [64], [82], [83]
Software-Intensive Systems	[59]
Stream Programming	[84]

Figure 2.7 shows the number of studies by software paradigms. The observation that the majority of the proposed work tends to be generic and not explicitly designed for a particular paradigm or application type implies that generality can come with advantages and disadvantages. Generality can imply application and evaluation of the proposed work under different contexts and applications, reflection on their strengths and weaknesses in dealing with the said paradigm. This can consequently provide inputs for further improvements and extensions. On the other hand, employing self-awareness can take simplistic assumptions, or tend to be limited when addressing the requirements of some paradigms, where speciality and customisation are desirable for more effective adaptations. Self-awareness that considers characteristics of particular software paradigms will result in advancing these paradigms. Yet, the validity of these observations can be subject to further empirical studies.

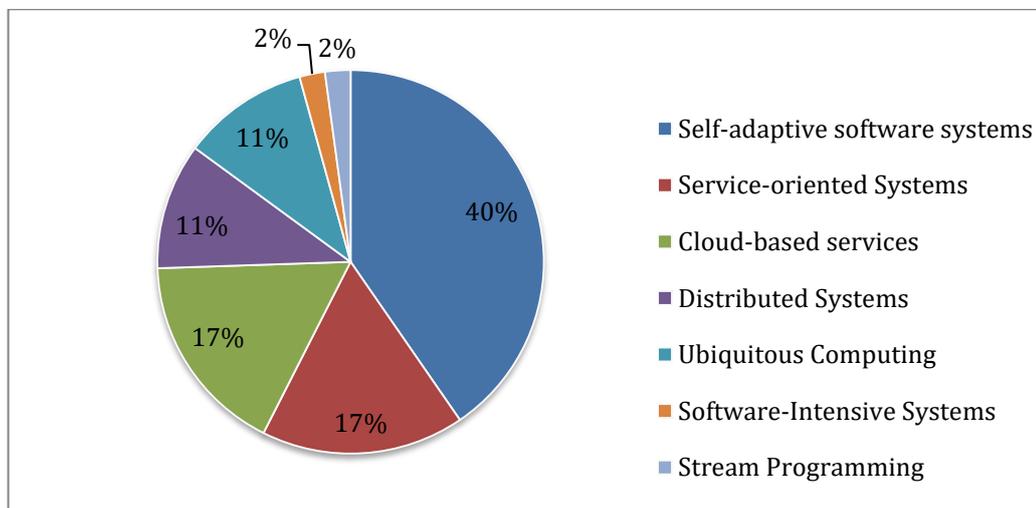


Figure 2.7: Distribution of Studies by Software Paradigms

With respect to the software engineering practices that addressed self-awareness, Table 2.14 summarises practices found in the primary studies. The results reflect that

architecture design is the practice that most contributed in realising self-awareness, as well system design and engineering adaptations. A number of studies also contributed in realising self-awareness for QoS resources management (with some explicitly focusing on performance), system specification (including formal methods), as well as knowledge representation and reasoning. Operation management during runtime and service composition also received some attention. Single research efforts also considered various practices; such as system development for stream programming [84] and language semantics for Object-Z [85]. These studies are domain-specific, which interprets their minimal number.

Table 2.14: Engineering Practices Realising Self-Awareness

<b>Engineering Practices</b>	<b>Studies</b>
Architecture design	[51], [90], [81], [38], [60], [70], [77], [61], [18], [78]
System design	[76], [53], [73], [72], [65], [19]
Engineering adaptations	[87], [79], [66], [71], [83], [63]
QoS and resources management	[88], [89], [57], [75], [68]
System specification	[54], [67], [74]
Knowledge engineering	[86], [80], [62]
Operation management	[82], [56]
Service composition	[58], [29]
System development	[84]
Language semantics	[85]

Meanwhile, some studies investigated the concept of self-awareness in software engineering. For instance, the works of [69] [13] [64] [58] reviewed the concept of self-awareness and its applications in computing systems. Other studies presented roadmaps for realising self-awareness in software systems [52], developing sustainable systems [40] and software-intensive systems [59], as well as realising service

composition [58] and enabling change and evolution [87]. The work of [55] discussed related technologies for enabling self-awareness. Though these studies did not explicitly consider certain engineering practices, yet they are contributing in formalising the concept of self-awareness and guiding the research community. Figure 2.8 shows the number of studies by engineering practices (also note that some studies appear multiple times under different categories, which explains why the total number of studies appearing in the figure is greater than the number of primary studies).

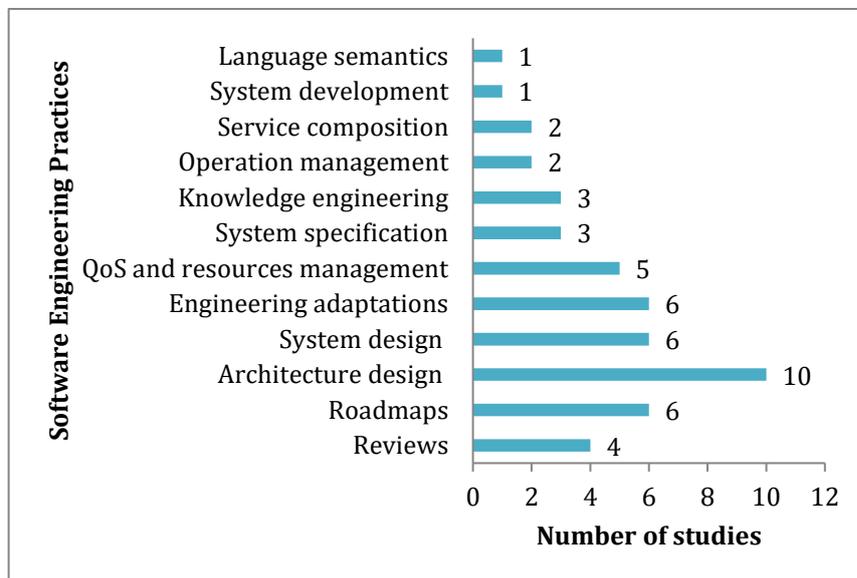


Figure 2.8: Distribution of Studies by Engineering Practices

### 2.3.3.5 Approaches for Engineering Self-Awareness (Q4)

Engineering self-awareness aims for encoding self-aware properties within the software systems in an attempt to provide systematic treatment for managing the software system state, knowledge, and execution environment. This question looks for the approaches that have been used to engineer self-aware software systems and categorises these approaches.

In literature, different approaches for engineering self-awareness in software engineering are found. On one hand, we have observed that 8 out of the 46 primary studies did not provide any engineering approaches for self-awareness in software engineering. These works have presented visions, outlined challenges, and raised questions. On the other hand, the remaining 38 studies claimed to provide engineering approaches for self-awareness. We have categorised these approaches into model-driven, architecture-centric, programming-driven, knowledge-representation, and development lifecycle-based approaches. Table 2.15 lists the engineering approaches categories and their related studies.

Table 2.15: Engineering Approaches and Related Studies

<b>Engineering Approach</b>	<b>Studies</b>
Model-driven	[88], [89], [56], [60], [65], [74], [63], [19]
Architecture-centric	[51], [90], [76], [55], [81], [38], [82], [57], [79], [58], [66], [70], [75], [77], [29], [61], [73], [72], [18], [83], [68], [78]
Programming-driven	[85], [84]
Knowledge-representation	[86], [80], [59], [62], [67]
Development lifecycle-based	[71]

Figure 2.9 shows the distribution of studies with respect to the classification of engineering approaches. Architecture-centric and model-driven approaches are found the most dominant approaches in the current literature. Other categories of approaches have taken less attention in the research community.

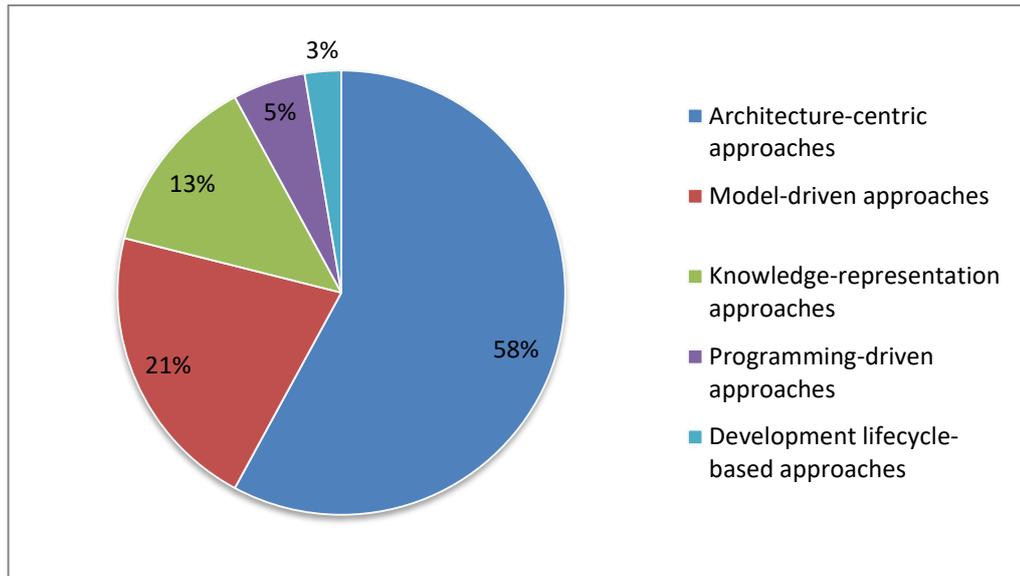


Figure 2.9: Distribution of Studies by Engineering Approaches

### **Model-driven approaches**

In general, the model-driven approaches attempt to create abstract models that represent the software system and its execution environment [91]. In dynamic systems, environments are characterised by changing behaviours and a demanding need for self-adaptation. This calls for runtime model-driven approaches [92] which capture the runtime system state, in order to help the system to decide when and how to adapt to accommodate changes.

In the literature of self-aware software systems, few model-driven approaches have been proposed. In [56], the authors have proposed a model-driven monitoring methodology to enable self-awareness in cloud platforms. The methodology presents a model for mapping the low-level metrics to the cloud market goals, in order to evaluate the performance of the goals. The work of [74] has presented a model for expressing self-adaptive behaviour of service-oriented applications using the SCA-ASM modelling language [93]. This extension of the SCA-ASM offers mechanisms to monitor the environment and the system itself and to perform adaptation actions. In [63], authors

have introduced a novel graphical language, namely, Extended Behaviour Trees (XBTs), for modelling adaptive and self-aware agents. The approach introduces a new reinforcement learning strategy that allows the interleaving of reasoning, learning and actions.

The works of Kounev et al. [88], [89] and [19] have taken the model-driven self-aware systems a step further by introducing the dynamic performance models as a 'mind' that controls a self-aware system. Such models enable predicting changes in the system workload and the execution environments leading to proactively adapting the system in order to avoid the requirements violation. In [65], the authors designed the Descartes Modelling Language (DML) as a tool for modelling QoS and resource management aspects of self-aware systems.

**Architecture-centric approaches** The architecture-centric approaches introduce reference architectures for representing the system's design decisions and constraints [94]. In general, the architecture-centric approaches have similar design trends, i.e. they consist of the following five components (which are basically the MAPE-K [7] or extended versions of the MAPE-K) [95]. The architectural framework proposed by [18] brings forward the MAPE-K by extending the knowledge modelling component, to enrich the self-adaptation capabilities by adopting the computational self-awareness principles. Inspired by the concept of self-awareness in Psychology, the architecture introduces five levels of self-awareness:

- *Stimulus-awareness*: This level is related to the knowledge about the basic events affecting the system. It does not support any ability of learning or prediction.

Hence, it provides knowledge for basic levels of adaptation, e.g. replacing a failed service in SOA-based applications.

- *Goal-awareness*: This level models the knowledge about the goals and objectives of the system and the extent to which the goals are being achieved.
- *Interaction-awareness*: This level is able to model the knowledge about the interactions among the different systems components and the interactions of the system with the environment. This enables to anticipate of how adaptations decisions can affect the interacted components and the environment.
- *Time-awareness*: This level enables modelling the knowledge about the past performance of the different system components, e.g. the historical performance of the services in an SOA-based application.
- *Meta-self-awareness*: This level acts as a 'brain' that reasons about the adoption of any of the other levels of awareness, based on the benefits and overhead of each of them.

Also, different architecture-centric approaches have been proposed to monitor the system and environment states to reason about the autonomous adaptation decisions at different levels; the infrastructure level, the architecture level, or the application level. The work of [68] is an example of adopting self-awareness in cloud computing at the infrastructure level, where self-awareness is used in the process of auto-scaling the physical resources in the cloud based on the changes in the workload. Introducing a set of quality-driven architectural patterns [78], one of the patterns, namely the Meta-self-awareness pattern is an example of using self-awareness to adapt at the architectural level, where an architecture adaptation manager manages the trade-offs between

different QoS requirements to switch between different architectural patterns. The work of [79] is an example of a self-aware architecture-centric approach for adaptation at the application level, where the approach presents an architectural framework that enables automatic scheduling of adaptation actions to react to the changes and fluctuations in the available resources.

In general, the architecture-centric approaches provide conceptual frameworks to engineer self-awareness. These frameworks are in abstract form and lack the demonstration of how self-awareness can be concretely and quantitatively achieved. Furthermore, the majority of these frameworks treat knowledge as a coarse-grained element, rather than refining the knowledge.

**Programming-driven approaches** Self-awareness has been rarely incorporated in an explicit way to propose self-aware programming paradigms.

The work of [84] proposed the inclusion of self-awareness in stream programming model in which stream data arrive continuously and change dynamically in rate or content due to the changes in computing resources or communication infrastructure. The proposed model, called StreamAware, enables dynamic and automatic task rescheduling, as well as data parallelism in response to the changes of the stream data.

The work of [85] has proposed the inclusion of the notion of 'self' in object-oriented formal specification languages, in order to express the awareness by an object of its own identity. This results in 'self-aware' objects which support the reasoning about object interaction in object-oriented programming paradigm.

**Knowledge representation approaches** Knowledge representation is a key activity towards achieving self-aware systems. It enables modelling the acquired knowledge (whether it is related to the internal system state or the system environment) that is needed to reason about the adaptation decision making.

Few approaches have been proposed for the knowledge representation in self-aware software systems. [86] proposed a multi-dimensional access structure, called *Heterogeneity-Aware Distributed Access Structure (HADAS)*, that can be used in self-aware systems to make the system's nodes self-aware by storing reflective information about its own state, such as processing power, storage, etc. In [80], the authors introduced an abstract approach for knowledge representation to show that knowledge can be represented by rule-based models, frames, semantic networks, concept maps, ontologies and logic. Following this work, an approach for implementing self-awareness based on the KnowLang framework [96] was later proposed in [59] and [62]. The framework provides a knowledge base that abstracts some context and a reasoner that allows for knowledge access in that context. In [67], the authors have introduced the SCEL (Software Component Ensemble Language), that is an approach for providing linguistic abstractions for describing the behaviour and knowledge of self-aware systems taking into consideration the evolution of the system ensembles and interactions among them. These works address knowledge representation at a coarse-grain level. They provide less focus on the potential structure of the collected knowledge and do not explicitly deal with dynamic knowledge management in self-adaptive systems, which have the effect of limiting the effectiveness of the adaptation.

### **Development lifecycle-based approaches**

The work in [71] proposed a general software development life-cycle to engineer self-adaptive systems. The approach is based on the decomposition of a complex system into service components. The local awareness of a component informs local adaptive behaviour. Then, a collective awareness is achieved by grouping the inter-related elementary components into ensembles to enable communication and knowledge exchange.

### **2.3.3.6 Evaluation of Self-Aware Software Systems (Q5)**

This question investigates the approaches that have been used to evaluate the proposed self-aware approaches. The question also looks at the evaluation criteria, reported performance and overhead of the approaches.

### **Evaluation Approaches**

We observed that 28 papers out of the 38 (that proposed engineering approaches) have provided some kind of evaluation for their approaches. We categorise these approaches into the following categories: analysis-, illustrative example-, illustrative application-, and simulation-based evaluation. Table 2.16 lists the evaluation approaches categories and their related studies.

Table 2.16: Evaluation Approaches and Related Studies

<b>Evaluation Approach</b>	<b>Studies</b>
Analysis	[85]
Illustrative example	[90] [88] [55] [89] [58] [60] [70] [67] [65] [61] [74] [63]
Illustrative application	[51] [81] [38] [79] [66] [75] [77] [73] [72] [18] [83] [84] [19]
Simulation	[56] [29]

- **Analysis-based evaluation.** The approach presented in [85] has provided an analysis-based evaluation approach to show how the concept of self-awareness supports the reasoning about object interaction in object-oriented programming paradigm.
- **Illustrative example.** The studies listed under this category have presented case studies to explore their approaches and validate the applicability of the approaches. But, they do not provide any measurements related to the performance of the proposed approaches. For instance, [65] provides examples to show how their modelling languages can be used to model the self-adaptive systems. [67] provides examples to show how the proposed knowledge representation approach can be used to represent the captured knowledge to reason about the adaptation.
- **Illustrative application.** The approaches listed under this category have provided real implementation as an illustrative application of their approaches, in order to demonstrate the applicability of the approach in real life. However, the experiments are performed on small-scale cases due to the complexity of performing large-scale experiments in a real setting. For example, [18] evaluates the proposed self-aware approach using a cloud-based application that has the ability to select the adaptation strategy according to the demand of the cloud-based services at runtime. The illustrative application demonstrates the adaptation capabilities of the approach. However, the experiments have been performed using two physical machines with one or two virtual machines hosted on each of them; a case in which the scale is too small compared to the large scale of cloud systems.

- **Simulation-based evaluation.** Studies adopting simulation-based evaluation have provided an experimental evaluation based on simulations featuring large-scale experiments. Such simulations provide the possibility to perform scalable and repeated experiments in a relatively fast and inexpensive controlled environment. However, these approaches still need to demonstrate the applicability of the approaches in real environments.

Figure 2.10 illustrates the distribution of studies by evaluation approach categories. The majority of studies have evaluated their work using either illustrative example or illustrative application. Simulation-based evaluation, featuring scalability, is significantly less used.

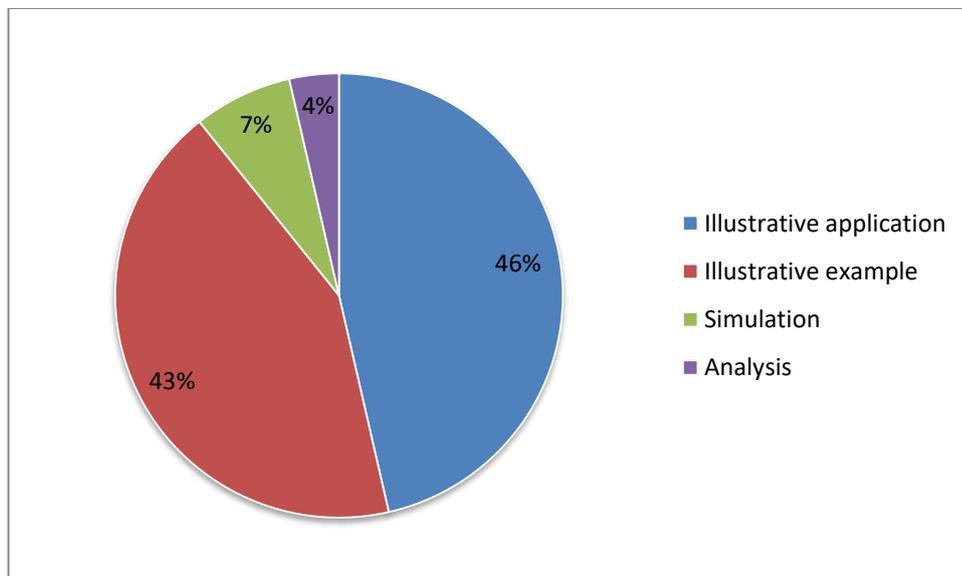


Figure 2.10: Distribution of Studies by Evaluation Approaches

**Evaluation Criteria** Below, we present the evaluation criteria that have been used in the mentioned studies, and then we present how each of the approaches addressed them. Table 2.17 lists the evaluation criteria and the corresponding studies.

Table 2.17: Evaluation Criteria and Related Studies

Study	Evaluation Criteria	Trade-offs
[51]	Accuracy	-
[38] [79] [75]	Accuracy, Efficiency	Accuracy, Efficiency
[81]	Processing Time	-
[56]	Number of bids, asks, allocations, average price, Market revenue	-
[66]	Reduction in communication	-
[29]	Number of violations	-
[73]	Power efficiency, Execution time	Power efficiency, Execution time
[77]	Power Consumption	
[72]	Lookahead, Latency, Number of achieved goals	Lookahead, Latency, Number of achieved goals
[18]	Accuracy, Adaptation Quality, Overhead, Reliability	Accuracy, Overhead
[83]	Local resources consumption, Time performance	-
[84]	Performance per Watt	-
[19]	Number of violations	-

**Performance** The studies listed under the *illustrative application* and *simulation* categories have reported on the performance of the proposed approaches. Though the motivation behind adopting self-awareness in the proposed approaches is to enrich the adaptation capabilities and to manage the trade-offs that exist among the different evaluation criteria, we observed that most of the evaluation approaches do not demonstrate how the improvements in one or more of the considered evaluation criteria affected the performance in terms of one or more of the ‘conflicting’ criteria. Also, we observed that some approaches claim the benefits of their self-aware systems without comparing their performance with non-self-aware or other self-aware approaches.

The approaches presented in [38], [79] and [75] have been evaluated using the *accuracy* and *efficiency* criteria. *Accuracy* measures the extent to which the actual performance of the system meets the performance goals. On the other hand, *efficiency* reflects the ability to minimise the power consumption while meeting the performance goal. The results show that the proposed approach has achieved higher accuracy, but with higher power consumption compared with a static approach. Similarly, [51] has evaluated the work using the *accuracy* criteria. They view *accuracy* as the probability of detecting email anomaly based on some adaptive measures, such as the mean and standard deviation of the captured data, which are application-specific evaluation criteria.

The evaluation of [73] has considered both *power efficiency* and *execution time*. The results highlight that the self-aware solution can achieve low execution time with minimal power consumption. The work of [77] has also considered power consumption in the smartphone case study. The results show that applying the self-aware strategies to activate system components on-demand has reduced the power consumption compared to a naive non-adaptive method.

The work of [72] has used three evaluation criteria, that are: *lookahead* that specifies the planning window in the future, *latency* that is the time required to finish planning, and the number of achieved goals. The results show that the larger the *lookahead* the higher the *latency* and *the number of achieved goals*. However, the above works do not demonstrate how their self-aware approaches are compared to non-self-aware (or other self-aware) approaches.

In [81], the authors evaluate the performance of their approach in terms of processing time. The results show that the proposed approach exhibit better performance compared to a 'conventional' approach. The work of [18] has used the weighted sum of *accuracy*, adaptation quality, overhead, and *reliability*, (assuming that the corresponding thresholds are specified in the Service Level Agreement) to evaluate the proposed approach. The results show that the weighted sum (called global benefit) in the self-aware case is higher than the weighted sum in the non-self-aware case. [84] has evaluated the self-aware approach using the normalised performance (in terms of computation time) per Watt in the presence of fluctuating input data streams and compares the self-aware approach with a set of static (non-self-aware) approaches. The reported results demonstrate that the approach's ability to adapt to the data stream fluctuations while keeping the performance per Watt close to the best static approach. [83] evaluates the approach using the *local resource consumption* and the *time performance* criteria. The paper claims acceptable time performance and resources consumption with increasing workload. However, the evaluation of these approaches does not demonstrate how the self-aware approach compare to non-self-aware (or other self-aware) approaches and does not address other quality attributes that may be adversely affected.

Simple metrics are found in the works of [29] and [19] that have evaluated their approaches using the number of violations. The results show that the number of violation is reduced leading to a more stable state. The work presented in [66] has considered an abstract quality criterion, namely, *communication*. The self-aware scenario results in a reduction of communication between the system objects compared to the non-self-aware scenario.

The evaluation scenario of [56] has considered a number of market-based metrics (number of bids, asks, allocations, average price, and market revenue) to show that the proposed cloud-market monitoring model is able to detect sudden changes in the demand for resources.

In brief, all the approaches show that performance improvements are obtained by leveraging self-awareness. The criteria used to demonstrate the improvements emerge from the application scenario domain, i.e. there are no agreed criteria. However, caution needs to be applied as other quality attributes may be adversely affected.

**Overhead** In this section, we investigate the overhead resulting from adopting self-awareness in software systems. Only 7 of the studies have reported on the overhead of adopting self-awareness. All of them considered overhead in terms of computation time.

[55] reported that the proposed approach is low-overhead without presenting experimentation results to demonstrate this claim. In [38] [79] and [75], the authors reported that the overhead of the proposed approach is very low and that the system can take adaptation decisions in 20.09 nanoseconds. However, other overheads related to adopting self-awareness, e.g. the overhead of monitoring, registering events and taking an action, have not been taken into account. [73] reported on the overhead related to the monitoring component of the approach. The reported runtime overhead is within 1%-2%, which the authors consider it to be negligible compared to the normal system's execution time. [83] reported the overhead of propagating the monitoring information across a network and stated that the overhead is 'acceptable' and limited. These approaches consider only the overhead of the monitoring activity.

The study of [19] has provided a more profound analysis of the overhead. The authors reported on the overhead of analysing the captured information and forecasting, as well as the overhead of the adaptation process. They reported that both overheads depend on the data, configuration settings, the techniques used for performance forecasting and the application specifications.

In general, leveraging self-awareness in computation will be accompanied by overhead. The overhead is due to the activities that collectively achieve the self-awareness, .e.g. monitoring, runtime analysis, knowledge management. Consequently, reasonable approaches to tackle this issue are generally required.

#### **2.3.4 Discussion**

In this section, we summarise the main findings, discuss the implications of the review on the research community, as well as report on the limitations and threats to the validity of the review.

##### **2.3.4.1 Main Findings**

We conducted this review with the vision of answering the five SLR questions. We reiterate that answering these five questions provides the background and knowledge on the literature of self-aware systems. This enables a better understanding of the thesis research question and informs the development and evaluation of our approach. The main findings of this systematic review are as follows:

- **Q1.** There is a growing attention to adopting self-awareness in modern software systems. The review results show that different research groups are active in the area of self-awareness. However, there is no common agreement

on the definition of self-awareness. Many researchers use the terms 'self-aware' and 'self-adaptive' interchangeably. Recent attempts to define self-aware systems include that a self-aware system should have multi-levels of knowledge representation and/or should support proactive adaptation. We attempted to provide a definition which mainly scopes to the case of dynamic software systems e.g. VC.

- **Q2.** Motivations for employing self-awareness were found clearly identified in the studies. Motivations varied between the general purpose of realising better autonomy for software systems and domain-specific purposes. The sources of inspiration were mainly nature and psychology, but the mapping between the self-awareness in software engineering and the source of inspiration is not well detailed in the majority of studies.
- **Q3.** Self-awareness was considered for self-adaptive software systems as a general software paradigm, with few studies focusing on a particular software paradigm or application type. Architecture design was found the most contributing software engineering practice in addressing self-awareness, as well system design and engineering adaptations. In general, although the paradigms are dynamic and require self-awareness to support self-adaptivity, they adopt means to limit the effect of dynamics (e.g. SLAs), which limits the demonstration of the value of self-awareness. In the following chapters, the thesis attempts to provide a case that better complies with the motivation of adopting self-awareness in dynamic software systems, e.g. the VC.
- **Q4.** The approaches for engineering self-aware software systems can be categorised as *model-driven*, *architecture-centric*, *programming-driven*,

*knowledge-representation* and *development lifecycle-based* approaches. In general, we notice that the engineering approaches are abstract and require concrete developments to realise knowledge management, which is central to self-awareness. Exceptions from that are the programming-driven approaches, however, they are domain specific and cannot be generalised. Chapter 5 and 6 present our attempt to provide an architecture-centric approach for realising self-awareness, which is quantitative in nature.

- **Q5.** Some of the studies have provided experimental evaluation of the proposed approaches. In general, the evaluations do not demonstrate the value added by self-awareness to the adaptation capabilities compared to the self-adaptive systems. The evaluations also need to report on the overhead accompanied with adopting self-awareness. The evaluations provided in this thesis show the added-value along with the overhead of adopting of self-awareness.

#### **2.3.4.2 Limitations and Threats to Validity**

The main limitations and validity threats of this review are related to the studies selection bias, inaccuracy in data extraction and analysis of collected studies.

- **Missing relevant studies.** The search was based on meta-data (abstract, title, and keywords) only and might have missed some studies that have considered self-awareness in software engineering as part of their proposed work, and are not mentioned this explicitly in the title, abstract and keywords. Though the meta-data are specified by the authors of the papers, we reasonably rely on how well the digital databases classify and index papers. Studies have been collected

from data sources that are basically academic indexing services. We have not considered other sources, e.g. companies' websites that might have addressed self-awareness in their industry-focused research and might have interesting findings.

- **Studies selection bias.** With respect to the selection of the initial studies, we adopted a set-up to guide the selection process, thus avoiding selection bias. For example, if the number of search results is more than 100 results, we selected the first 100 (an exception is the case of SpringerLink data source which treats the word 'self-aware\*' as two words, and the results that contain either the word 'self' or the word 'aware\*' were retrieved, which resulted in huge number of irrelevant results). Such set-up directed the selection based on the search results rather than only researchers' knowledge and background.
- **Inaccuracy in data extraction.** Inaccuracy can be introduced in the data extraction process due to different reasons, such as the background of the researcher, the researcher's subjectivity and the way the authors' studies used to present their approaches and findings. Aiming at minimising the inaccuracy in data extraction, we adopted a strategy for the data extraction process, such that data extracted from certain studies are double-checked. Also, we had thorough discussions to eliminate any confusion, which leads us to believe that the effect of this error is minimal.

#### **2.3.4.3 Implications for Research**

The aim of this systematic review on self-awareness in software engineering is to investigate how current research has adopted computational self-awareness to enrich

the self-adaptation capabilities of autonomous software systems. This chapter provides the first comprehensive review that summarises the relevant literature and reports on possible gaps. Overall, the review provides a quite representative state of the relevant literature. The findings inform the research for developing approaches for filling the gaps (e.g. the attempt presented in the rest of this thesis). The findings can also support researchers interested in future research for advancing self-aware systems.

## 2.4 Gaps in Brief

While self-awareness is getting popularity as an enabler for self-adaptation in software systems, there are issues and challenges that need further considerations. In this section, we present the main challenges related to the adoption of self-awareness in software engineering.

- **Dynamic knowledge management.** Dynamic knowledge management in the studied self-aware software systems architectures has been given little consideration; although it is a vital requirement for self-awareness. Most of the self-aware frameworks address knowledge management at a coarse-grained level. They do not provide concrete approaches for various representations of the captured data to extract profound knowledge. We argue that ‘finer’ knowledge representation can better address the users’ and system’s requirements in the environments that exhibit uncertainty and dynamism. It can also improve the quality and accuracy of adaptation. In this context, the knowledge management approaches should adhere to the separation of concerns principle by providing the capabilities to dealing with different types of knowledge, e.g. knowledge related to basic stimuli, historical performance, and interactions among the

different entities. Furthermore, the knowledge should be treated as moving targets that can change and evolve over time. Self-aware systems should be able to capture the evolution trends and use this information to better inform the adaptation decisions.

- **Evaluating the quality and overhead of self-awareness.** As found in the primary studies, the evaluation of the quality and overhead of self-awareness is not considered as it should be. The majority of the studies demonstrated the improvements in quality attributes after employing self-awareness. Meanwhile, the extent to which this improvement can be accepted has not been tackled. In other words, the evaluation needs to show the overheads that may accompany the improvements. Solutions for managing the overhead-quality trade-off are still required.

## 2.5 Related Reviews

The SLR presented in this chapter could be treated as the first attempt to provide a comprehensive overview on self-awareness in software systems. The review addressed several research questions to investigate the definitions, motivations, and engineering and evaluation approaches of self-awareness in software engineering. However, in this section, we briefly present other attempts which can relate to reviewing self-awareness to some extent.

An early review has been conducted about self-awareness and its application in computing systems [64]. This work surveyed definitions of self-awareness in biology and cognitive science. The work also discussed previous efforts that incorporated self-

awareness in different computing systems; such as pervasive computing. We have considered this study among the primary studies of this review. Focusing on context-awareness, the survey conducted by Baldauf et al. [97] presented common principles and elements of context-aware software architectures, as well as analysed aspects of context-aware computing, which may be considered as part of self-awareness.

Reviews from the field of self-adaptive software systems include [10] [98] [33]. Other surveys focused on one of the self-\* properties, such as self-protecting [99].

## **2.6 Summary**

The limitations of the self-adaptive systems motivated the researchers to incorporate self-awareness to enrich the self-adaptation capabilities. This chapter surveyed the landscape of the self-aware software systems literature. We conducted a systemic literature review to compile the studies related to the adoption of self-awareness in software engineering and explore how self-awareness is engineered and incorporated in software systems. From 532 studies, 46 studies have been selected as primary studies. We have analysed the studies from multiple perspectives, such as motivation and inspiration for employing self-awareness, software paradigms and engineering approaches addressing self-awareness, as well as evaluation approaches.

Results have shown that self-awareness has been used to enable self-adaptation in systems that exhibit uncertain and dynamic behaviour during their operation. Although there are recent attempts to define and engineer self-awareness in software engineering, there is no general agreement on the definition of self-awareness and there is a lack of distinction between self-aware and self-adaptive systems. Evaluating self-

awareness engineering approaches and exclusive mapping with their sources of inspiration still need to be addressed.

The main findings show that there is a growing attention to incorporate self-awareness for better reasoning about the adaptation decision making in autonomic systems. However, many pending issues and open problems still need to be addressed. The area of self-awareness in software systems demands the development of approaches for dynamic knowledge management to provide profound knowledge for enhancing self-adaptation. The approaches have also to provide capabilities for managing the trade-off between the expected quality and the incurring overhead.



# CHAPTER 3

## MOTIVATING SELF-AWARENESS FOR VOLUNTEER COMPUTING

### **3.1 Overview**

In chapter 2, we identified that the motivation behind adopting self-awareness as a capability in software systems is the complexity of modern software systems (e.g. emergent behaviours and unpredictable changes of the operating environment) with the purpose of achieving better autonomy for software systems. We also identified that one of the limitations of the current self-aware approaches in software systems is that they consider environments with limited or controlled dynamism for applying the proposed self-aware approaches. That is, the dynamism and the related uncertainty of the considered environment tend to be limited due to some controls (e.g. SLAs). The application of self-awareness in such environments lessens the illustration of the claimed benefits of self-awareness. For this reason, we consider the Volunteer Computing (VC) environment, as a highly-dynamic environment, to illustrate the engineering of our self-aware approach for dynamic knowledge management, which is presented in chapters 5 and 6.

For the sake of building our motivating scenario, which will steer the presentation of the rest of this thesis, this chapter introduces the VC environment along with the representative VC systems. The chapter briefly provides a qualitative analysis of the adaptation capabilities of the representative VC systems. We also touch on their approaches for selecting the volunteer hosts. After that, we deduce gaps in those VC systems.

### **3.2 Volunteer Computing: Challenges and Characteristics**

Volunteer Computing (VC) is an emerging distributed computing paradigm in which users make portions of their own resources available to others enabling them to do distributed computations and/or storage [100]. The paradigm is believed to be an enabler for cost-effective large-scale computation and sharing for storage, leveraging on spare resources that can be available and idle on the users' computing devices (e.g. PCs, laptops, smartphones, etc.). The paradigm has been seen as an alternative for purchasing resources in large scale projects, where utilising volunteered resources can bring the benefits of large-scale inexpensive and shared computing and storage [12].

The Grid Computing paradigm has given rise to the development of the early grid-based volunteer computing platforms, e.g. BOINC [101] and Xtremweb [102]. Such platforms enabled the volunteers to donate their resources for scientific projects, e.g. Seti@Home [103], Storage@Home [104], Folding@home [105], and others. Such projects are characterised by a requirement of large-scale computation and/or storage. After the emergence of the Cloud Computing paradigms, researchers tended to propose cloud-based VC platforms, e.g. Cloud@Home [106], Nebula [107], and SocialCloud [108].

Such platforms enable the volunteered resources to be offered as services, which requires less expertise and effort to create the VC projects [108] [109].

In VC the computation/storage units are edge devices controlled by individual users i.e. volunteers. The availability of those machines and the continuation of providing the 'promised' resources depend highly on the ability and/or the interest of the volunteers [100]. This introduces a challenge of distributing the tasks (i.e. selecting the volunteer machines) according to not only the capacity of the volunteer machines but also to the performance of those machines. In addition, the volunteers have heterogeneous machines with different and varying capabilities, which contribute to the above challenge. For example, the storage, computational capabilities, and the performance of a smartphone are very different from a powerful desktop machine. Therefore, the VC environment exhibits high dynamism, openness, and heterogeneity. Consequently, the service provision in this environment is accompanied by uncertainty and dilution of control. To put it more clearly, the provision of resources in VC faces the following challenges:

- *Resources-awareness*: The efficient utilisation of the volunteer resources is one of the greatest challenges of VC. The contributed resources need to be selected (and in some cases composed) and allocated to users achieving both maximum utilisation and minimum waste with minimum computation time [12].
- *Availability-awareness*: the volunteers contribute their resources during the time intervals in which they do not need these resources, i.e. the volunteered resources are not available permanently and there is uncertainty related to how long a volunteer resource will remain available [5].

- *Dilution of control*: As volunteer resources are offered on a voluntary basis by individuals and organisations 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 the volunteered resources can be offered and withdrawn at any time [29]. The right without the symmetric obligation to participate in VC makes Service Level Agreements (SLAs) less stringent as when compared to commercial services.
- *Dependability-awareness*: based on dilution of control challenge, dependability information of the volunteer hosts, in terms of the level of providing the promised resources, should be collected and used in VC allocation approaches. The use of this information in volunteer resources selection and allocation enables the selection of more 'dependable' resources, leading to more reliable service provision.

Under these circumstances, the dynamism in the VC environment calls for novel self-adaptive approaches for dynamically managing the processes of selecting and allocating volunteered resources. Furthermore, the selection approach should take into account the volatility of the volunteered resources so that the selection and adaptation decisions are improved. In cases when one volunteer host cannot satisfy a request, multiple volunteered resources need to be aggregated. In such cases, volunteer hosts should be selected based on the extent to which these volunteered resources are able to satisfy the user's request, avoiding over-provisioning the resources.

### 3.2.1 Performance Patterns of the Volunteer Hosts

In addition to the dynamism, uncertainty, and heterogeneity characteristics of the VC environment, another characteristic exists, which is the existence of periodical performance patterns. In [110] Douceur studied the distribution of the performance of internet hosts in terms of availability. The reported results show that the internet hosts availability follows either normal distribution or cyclical behaviour. These results motivated Lazaro et al. to perform a long-term study on the performance of the volunteer hosts [5]. The study analysed a large set of traces from 226,208 volunteering hosts, taken from the SETI@Home real system [103]. The reported results reveal the presence of periodic patterns in the performance of the volunteer hosts. The patterns are usually repeated over a certain time period that varies from one volunteer to another. Such period can be some hours, days, or weeks. Figure 3.1 shows the behaviour of a subsample of the hosts. The figure shows that periodic performance patterns are present for many hosts.

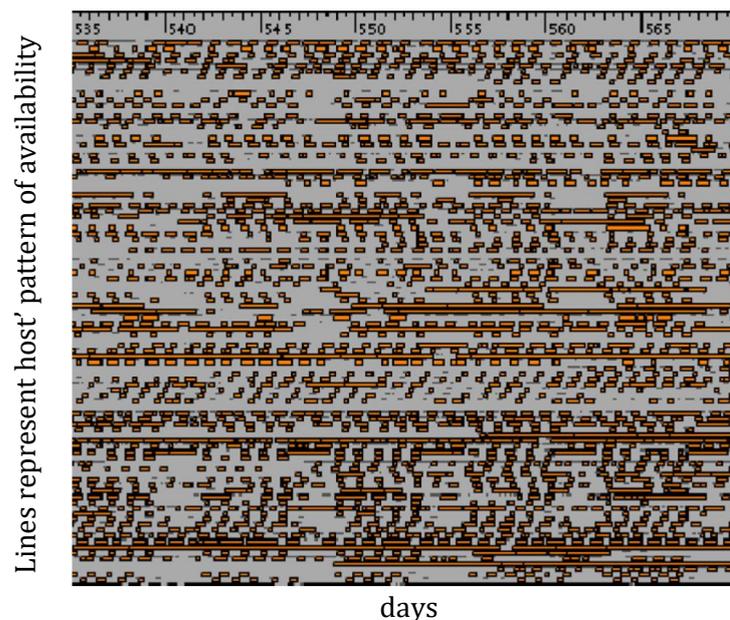


Figure 3.1: Trace of A Random Subsample of the Hosts Availability. Source [5].

In addition to that, other studies show that the independence assumption on hosts' performance is not always valid, i.e. hosts performance can be correlated [111] [112] [113]. In [111] Kondo et al. analysed the performance log files of 112,268 volunteering hosts, collected from the SETI@Home [103]. The study reported that the performance of the hosts can be positively or negatively correlated. Furthermore, the study detected that correlated periodic patterns exist among the volunteering hosts. The authors of [111] grouped the hosts into 5 clusters as shown in Figure 3.2<sup>2</sup>. (The legend shows the names of the clusters, which reflect a range from highly-available hosts to the low-available hosts) The figure demonstrates the existence of the positively and negatively correlated periodic patterns.

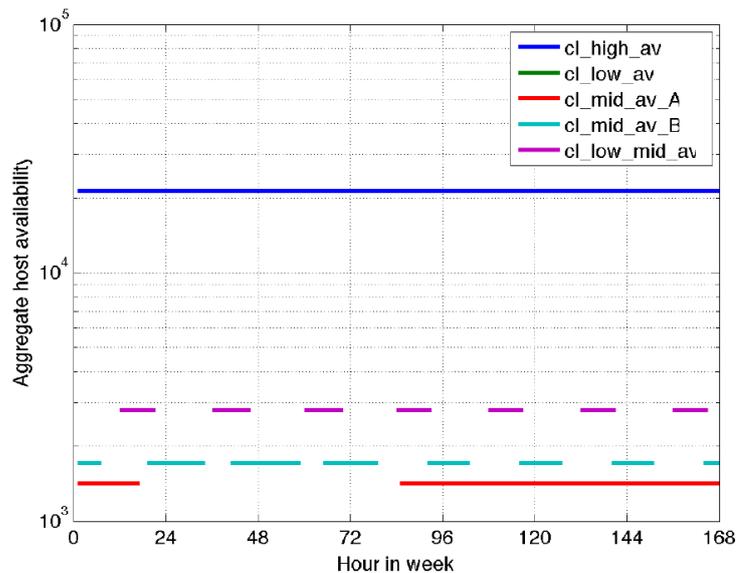


Figure 3.2: Hosts Clustered by Availability. Source [111].

Based on the above, we argue that the awareness of such periodic patterns can inform the selection, allocation, and adaptation of the volunteered resources. It enables

<sup>2</sup> As mentioned in [111] “Cluster *cl\_low\_av* does not appear in the figure because it is a vector with only zeros. However, the hosts in this cluster are those with low availability, not necessarily zero availability”

the prediction of the volunteer hosts' performance, which helps to reason about the selection and adaptation decisions. We also argue that the awareness of the existence of correlation enables reasoning on selecting the hosts that exhibit satisfying performance in the case when the hosts need to interact with each other, e.g. when composed to satisfy a certain request.

Accordingly, the complexity of the VC environment along with the periodic and correlated performance patterns of the volunteers stimulate the need for dynamic knowledge management approaches. Such approaches should provide the capabilities for representing knowledge at a fine-grained level by structuring the knowledge into different levels. Each level should address different knowledge concern, e.g. the periodical patterns, the correlation, the stimuli, etc. Correspondingly, the approaches should employ the fine-grained knowledge to enrich the self-adaptation capabilities in the case of VC.

Meanwhile, no doubt that the traces used in the aforementioned studies are useful to inform on the existence of periodical patterns and correlation in the volunteer hosts' behaviour. However, it should be taken into account that different volunteers contribute to different projects. That means the traces collected from one VC project (e.g. SETI@Home) cannot be used to predict the actual performance patterns of the volunteers contributing to a different project. Henceforth, knowledge needs to be captured, represented, and managed at runtime in a way that enables capturing the performance patterns and the correlated patterns. Moreover, it should be taken into consideration that the knowledge is evolving and can change over time. That means, the

knowledge data points arrive continuously and the knowledge models need to be built and updated incrementally over the operation time of the VC system.

### **3.3 Representative Volunteer Computing Systems**

In this section, we provide an overview of the current representative VC systems and assess their properties with respect to the aforementioned challenges. It is worth mentioning here that the focus of this thesis is not to improve any of the VC systems. The purpose of presenting those representative VC systems is two-fold, (1) to provide objective evidence that the VC paradigm can be a representative case to demonstrate the development related to self-awareness and (2) to investigate how these systems address the challenges of the VC environment. This overview will inform our scenario that will steer the development and presentation of the proposed self-awareness framework in the following chapters.

We assess the characteristics of the VC systems qualitatively using the following criteria. The reached conclusions are derived qualitatively by close assessment of seminal papers reporting on the fundamentals of the VC systems themselves and follow-up related application papers if any:

- 1. Knowledge representation and management:** This criterion determines how the VC system manages the collected knowledge about the volunteer hosts and how this knowledge is used to inform the adaptation decisions. The possible values of this criterion are implicit, explicit, or none.
- 2. Separation of knowledge concerns:** This criterion determines whether the VC system takes into account the potential structure of the knowledge or not. In other words, this criterion is to determine whether the VC classifies the collected

knowledge about the volunteer hosts into levels or classes where each level is concerned by a certain type of knowledge, e.g. knowledge about hosts' interactions, knowledge about hosts' historical performance, etc. The value of this criterion is the number of levels.

3. **The degree of autonomy:** This criterion determines the extent to which the VC system adapts autonomously without human intervention. The possible values of this criterion are fully-autonomous, semi-autonomous, or none.
4. **Time of adaptation:** This criterion determines whether the VC system supports reactive and/or proactive adaptation.
5. **Availability-awareness:** As mentioned above, the volunteers contribute their resources during the time intervals in which they do not need these resources. That means the resources can be available during certain time intervals, e.g. hours of a day or days in a week. This criterion determines whether the VC system considers the *availability time intervals* or the *instantaneous availability* of the volunteer hosts. Possible values are instantaneous or interval-aware.
6. **Volunteers' selection:** The selection of the volunteer hosts should consider both efficient utilisation of the volunteer resources and the performance of the volunteer hosts. This criterion figures out the criteria used in the representative VC systems to select the volunteer hosts' for a certain request.

### 3.3.1 BOINC

BOINC is the earliest VC middleware [101]. It enables for creating public-resource computing projects. Through this middleware, users can share their resources and specify their contributions to the projects. SETI@Home [103] 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 [105] project benefits from the huge computational power of volunteered processing resources to simulate a biological process, the protein folding. Storage@Home [104] is a project that has been developed to enable backing up, storing, and sharing huge amounts of scientific results using volunteered storage. Einstein@Home [114] is designed to search for and analyse gravitational waves, which requires massive computational power. These projects follow a master-worker computing model in which a master assigns the computing tasks to the workers and then verifies the returned results [12]. Users who are willing to provide their resources need to download and install a client on their machines. The client should be configured to connect the volunteer's machine to the project the volunteer wishes to contribute to. Then the client connects to the project's servers to request workloads and submit results.

BOINC makes use of application-level adaptation support. Applications can deal with the dynamism of environment providing their own adaptation capabilities [115]. SETI@Home, as a project running on BOINC, supports 'task-resubmission' as an adaptation technique. If a volunteer host fails while processing a task, the task is sent to another host [103]. Storage@Home adopts a replication strategy to mitigate data loss in the cases of hosts' failures.

#### Pros

- Adaptation capabilities are application-centric which gives flexibility to the application developers to implement their own adaptation techniques. However, SETI@Home is the only application that supports some kind of adaptation.

## Cons

- In SETI@Home, knowledge representation is implicit and restricted to the detection of the unavailability of the volunteer host. The system does not model and store knowledge on the historical volunteers' performance. Consequently, there is no separation of knowledge concerns. Also, there is no support for proactive adaptation.
- In SETI@Home, the efficient utilisation of resources is not considered. The volunteer hosts' selection is based on the promise of accomplishing the computation/storage task, without taking into consideration the over-provisioning of the volunteered resources and the historical hosts' performance.

### 3.3.2 Cloud@Home

Cloud@Home is a volunteer computing project funded by the Italian Ministry of Education and Research. The project goal is to develop a middleware to manage the contribution and usage of the volunteered computational and storage resources allowing creating private volunteer clouds that interoperate with the commercial clouds in a heterogeneous environment [116]. The Cloud@Home model involves three actors, *Cloud@Home-Provider*, *Cloud@Home-User*, and *Cloud@Home-Admin* [117]. The *Provider* contributes storage and/or computing resources to the system. The *Admin* sets up and manages the system. The *User* submits requests in order to obtain the resources.

Figure 3.3 shows the Cloud@Home system architecture which consists of three main modules, namely, the *Resource Management Module* [118] which is responsible for the provision of the resources to the Cloud@Home user, the *Resource Abstraction Module*

which is responsible for abstracting the heterogeneous resources offered by the Cloud@Home providers, and the *SLA Management Module* which is responsible for negotiating the QoS required by the Cloud@Home users [119].

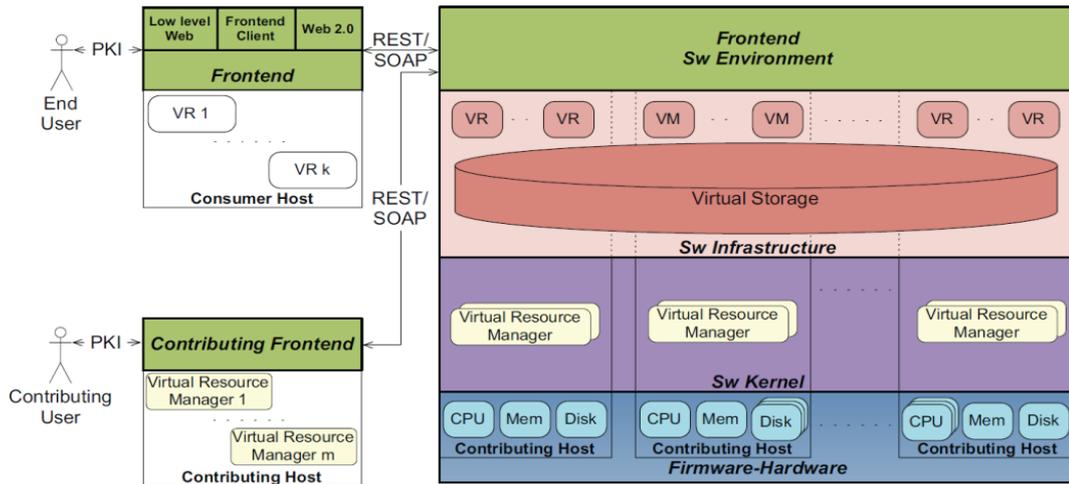


Figure 3.3: Cloud@Home Architecture. Source [118]

The Cloud@Home system introduces a process for the management of the QoS required by the users. The process considers only the *availability* parameter for the QoS and adaptation management. The process involves the traditional *Negotiation, Monitoring, Recovery, and Termination* activities [119]. The SLA manager component in the SLA management module carries out the negotiation activity when a user submits a request. During the negotiation activity, the user QoS requirements, in terms of *availability*, will be negotiated with the contributing host to determine whether they can be satisfied or not. The monitoring activity is implemented using MAGDA [120] approach in the MAGDA component. A MAGDA agent is supposed to run on each volunteered node. This agent sends periodic status reports to the MAGDA component in

the Cloud@Home system. The recovery activity provides the adaptation capabilities supported by the Cloud@Home. If a MAGDA agent does not send a status report within a specified period of time, the MAGDA component assumes that the node is no longer available and sends an alert to the *Resource and QoS Manager (RQM)* component in the *Resource Management Module*. Then the RQM triggers an adaptation action to replace that faulty node. Then a new MAGDA agent is sent to the newly selected node in order to monitor its status.

#### Pros

- The system is fully-autonomous and the adaptive part is separated from the functional part of the system, making it clear to understand the adaptation cycle.
- The system is availability-aware. The time interval in which the volunteer host is available is implicitly considered in the negotiation phase.

#### Cons

- Knowledge representation is implicit and restricted to the detection of the unavailability of the volunteer host. The system does not model and store knowledge on the historical volunteers' performance. Consequently, there is no separation of knowledge concerns. Also, there is no support for proactive adaptation.
- The efficient utilisation of resources is not considered. The volunteer hosts' selection is based on the promise of accomplishing the computation/storage task, without taking into consideration the over-provisioning of the volunteered resources.

### 3.3.3 Nebula

Nebula has been proposed to utilise voluntary resource to support data-intensive applications. The main goal is to take advantage of the geographic distribution of the voluntary resources by assigning tasks to the close voluntary resources in order to reduce the data mobility cost [107].

Nebula's architecture consists of four main components, namely, *Nebula Central*, *DataStore*, *ComputePool*, and *Nebula Monitor* [121]. The *Nebula Central* is the interface through which volunteers provide their resources and users submits their computing or storage requests. The *DataStore* consists of *Data Nodes* which provide the volunteered storage and the *DataStore Master* which makes the data placement decisions based on the geographic location of the data nodes. The *ComputePool* consists of the *Compute Nodes* which provide the volunteered computation and the *ComputePool Master* which schedules the computation tasks based on the request's requirements and data storage location. The *Nebula Monitor* is responsible for monitoring the volunteer hosts' location and availability. This information is used in the *Data Nodes* selection and the tasks scheduling carried out by the *DataStore Master* and *ComputePool Master*. Figure 3.4 shows the Nebula system's architecture.

Nebula adopts a *Ping-Found* mechanism in order to keep track of the availability status of the volunteer nodes. Each *DataNode* periodically *pings* the *DataStore Master* and each *ComputeNode* periodically pings the *ComputePool Master*. Nebula provides two fault tolerance mechanisms in order to handle the failure of the *Data Nodes* and the *Compute Nodes*. Once a failure is detected a corresponding fault tolerance mechanism is applied to handle that failure. Data replication is used for the *Data Nodes* failure cases.

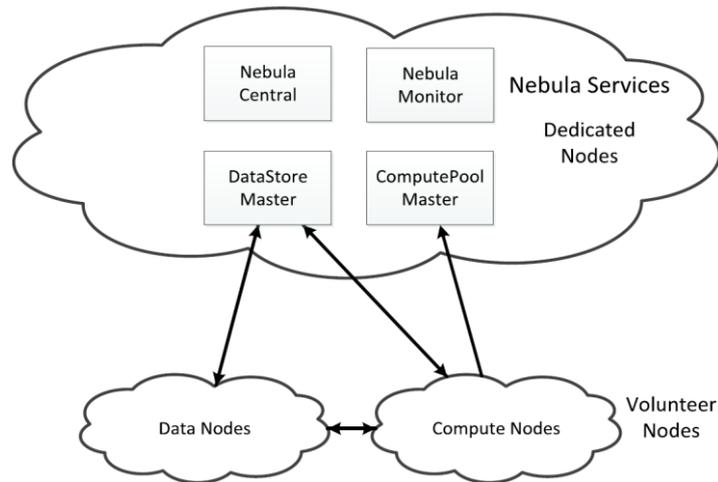


Figure 3.4: Nebula System's Architecture. Source [121]

The *DataStore Master* is responsible for keeping a number of replicas for each data file. The number is provided by the user. In the case of a *Data Node* failure, the replicas are used to restore the lost data. On the other hand, in the case of a *Compute Node* failure, the *ComputePool* master reassigns the execution of the task on another *Compute Node* [121].

#### Pros

- Location-aware resource selection reduces the overhead of data mobility.
- The system is fully-autonomous.

#### Cons

- Knowledge representation is implicit and restricted to the detection of the unavailability of the volunteer host. The system does not model and store knowledge on the historical volunteers' performance. Also, there is no support for proactive adaptation.

- The system does not consider the time interval in which the volunteer host is available, it considers only the ‘instantaneous’ availability of the hosts.
- The efficient utilisation of resources is not considered. The volunteer hosts’ selection is based on their geographical location and ‘instantaneous’ availability without taking into consideration the over-provisioning of the volunteered resources.

### 3.3.4 Cloud4Home

The work of [122] proposed the Cloud4Home system which aims at aggregating the home devices computing resources and the public cloud resources for better storage service delivery. Each Cloud4Home node maintains a *mandatory bin* for its storage needs and a *voluntary bin* which is the storage made available to the other nodes. The voluntary bins of all the nodes in the system form the *Home cloud*. If an application running on a node needs extra storage to store data, i.e. the mandatory bin is full; the system stores the data on any of the available volunteer hosts of the home cloud or on the public cloud according to the policy associated with the application.

The architecture of the Cloud4Home is realised in the VStore++ system which consists of two domains, the *Guest* domain and the *Control* domain, as shown in Figure 3.5. The *Guest* domain provides the interface for the user’s application that may request external storage. On the other hand, the *Control* domain provides the *meta-data resource management layer*, which stores the nodes’ identifiers and their available resources, the actual *store* and *fetch* operations for data storage and retrieval, and the interfaces for the *Home cloud* and the public cloud.

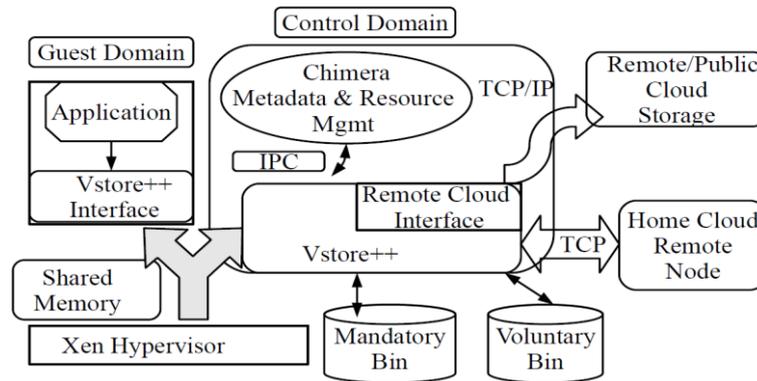


Figure 3.5: Cloud4Home Architecture. Source [122]

The Cloud4Home does not mention any mechanisms to handle the dynamicity of the environment. The authors left the adaptation issues for future work.

#### Pros

- Providing real application (VStore++) for leveraging the end devices voluntary resources for data storage.

#### Cons

- No handling of the dynamisms of the environment. There is no support for adaptation and knowledge management.
- Volunteer hosts' selection is based on their 'instantaneous' availability. The over-provisioning of the storage, the storage composition, and the historical hosts' performance are not considered to reason about the selection of hosts.

### 3.3.5 SocialCloud

The authors in [123] outlined their vision of a SocialCloud and defined SocialCloud as a "resource and service sharing framework utilising relationships established between members of a social network". The aim is to use the trust relationships that already exist

between members of a social network, e.g. the friendship relation in Facebook, for resource sharing. The resource contribution in SocialCloud can be either for gain or for no gain, in the latter case it is volunteering [108].

The authors realised the SocialCloud vision as a *Social Storage Cloud (SCC)* through a Facebook application. Friends can use the application for storing documents and photos leading to reduce the burden of infrastructure requirements of the provider. Figure 3.6 shows the architecture of the SCC. Resources contributors register their services (resources offers) through the *Registration and Discovery* component. When a user submits a request for storage the *Market Protocol* discovers the offered services by accessing the *Registration and Discovery* service. The *Market Protocol* component implements two gain-based resource allocation protocols, namely, *Posted price* and *Reverse auctions*. Based on the allocation protocol, a list of the available services will be displayed to the user, and then the user selects one of them. After that, an SLA will be generated by the *Agreement Management* component.

The SocialCloud does not mention any mechanisms to handle the dynamicity of the environment.

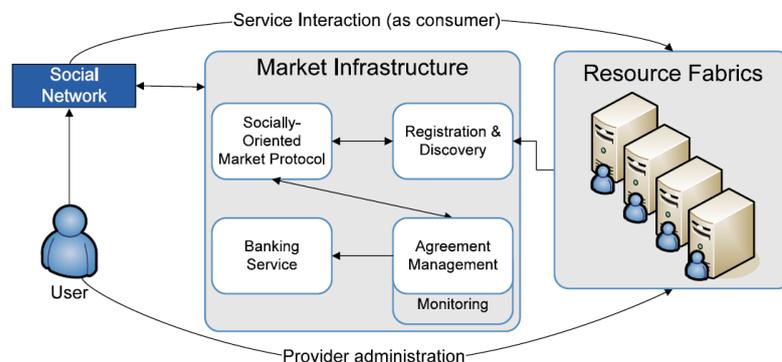


Figure 3.6: Social Storage Cloud Architecture. Source [108]

## Pros

- Utilising trust relationships in social networks encourages users to contribute resources.

## Cons

- No handling for the dynamisms of the environment. There is no support for adaptation and knowledge management.
- Volunteer hosts' selection is based on their 'instantaneous' availability and on the promised storage space. The over-provisioning of the storage, the storage composition.

Table 3.1 summarises the outcome of the comparative analysis.

Table 3.1: Summary of the Analysis of the Representative VC Systems

<b>VC system → Criteria ↓</b>	<b>BOINC</b>	<b>Cloud@- Home</b>	<b>Nebula</b>	<b>Cloud4- Home</b>	<b>SocialCloud</b>
<b>Knowledge representation and management</b>	implicit	implicit	implicit	implicit	implicit
<b>Separation of knowledge concerns</b>	1-level (unavailability of hosts)	1-level (unavailability of hosts)	1-level (unavailability of hosts)	0-levels	0-levels
<b>Degree of autonomy</b>	fully-autonomous	fully-autonomous	fully-autonomous	non-autonomous	non-autonomous
<b>Time of adaptation</b>	reactive	reactive	reactive	N/A	N/A
<b>Availability-awareness</b>	instantaneous	implicit interval-aware	instantaneous	Instantaneous	instantaneous
<b>Volunteers selection criteria</b>	instantaneous -availability	promised interval-availability	instantaneous-availability	instantaneous-availability	instantaneous-availability

### 3.4 Gaps Analysis

From the above, we deduce that

- The selection of the volunteer hosts to serve users' requests does not take into account the efficient utilisation of resources. The use of the selection approaches in the current VC systems can result in major over-provisioning of the resources. Although optimal resource utilisation in VC is not achievable due to scalability issues, tangible consideration should be given to the resources allocation in order to reduce resources waste. In this context, we motivate the need for a novel selection approach that takes such issue into consideration.
- Until recently, little attention has been given to the issues of autonomy and knowledge management in the aforementioned VC systems and projects, though the vitality of them to 'reliably' provide resources for large-scale computation and storage. For example, in SocialCloud and Cloud4Home, there are no means of dealing with the dynamism of the environment. These systems do not provide any adaptation capabilities. Other systems, e.g. BOINC, Nebula, and Cloud@Home provide minimal adaptation capabilities which are limited to replacing faulty hosts (i.e. volunteer machine) when their unavailability is detected. In these systems, knowledge management and representation is restricted to the detection of the faulty volunteer hosts. Those systems do not articulate how the evolving knowledge on the volunteers' performance is maintained and used for informing the adaptation decision making. We argue that 'finer' knowledge modelling is vital for improving the quality of adaptation. The captured knowledge of self-adaptive systems needs to be treated as 'moving targets' that

can change and evolve over time. Such fine-grained modelling can capture the performance patterns and better inform the adaptation. This can consequently improve the quality and precision of adaptation in VC as a dynamic, open, and uncertain environment. On the other hand, the overhead of such fine-grained modelling should be taken into account as high overhead cases will limit the performance of the system in terms of its ability to satisfy the users' requirements. Therefore we motivate the need for a framework for dynamic knowledge management that considers fine-grained knowledge management to inform the adaptation along with the accompanied overhead in VC.

It is worth mentioning here that these gaps apply to applications that require concurrent availability of multiple volunteer hosts, as when this is not required these gaps are not valid.

### **3.5 Summary**

In this chapter, we introduced the VC environment in which users make their resources available for other users or projects for free. We also introduced the characteristics of the VC environment that motivate the need for self-awareness with dynamic knowledge management. We illustrated that the evolving knowledge on the volunteer hosts performance, which is captured at runtime, needs to be managed and represented at a fine-grained level. After that, we overviewed the representative VC systems and deduced the gaps in their adaptive capabilities and volunteer hosts' selection. As a conclusion of this chapter, the characteristics of the VC paradigm, which relate to complexity and knowledge concerns, in addition to the mentioned gaps in the current VC systems self-

adaptation capabilities, make VC render itself as a sensible environment for understanding and improving dynamic knowledge management.

In the following chapters:

- We present our volunteer storage system as a motivating scenario that will steer the presentation of our self-aware approaches. We propose a utility-based approach for the selection of the volunteer resources to satisfy the users' request.
- We propose a self-aware framework for dynamic knowledge management. The framework supports knowledge representation at multiple levels of awareness and enables the switching between the different self-awareness levels based on the state of the system.
- We provide an evaluation of and a comparison between the different awareness levels.



# CHAPTER 4

## SERVICES SELECTION FOR VOLUNTEER COMPOSITE SERVICES: A UTILITY MODEL

### **4.1 Overview**

As discussed in the previous chapter, the VC paradigm is a promising paradigm for utilising volunteer resources for offering cost-efficient cloud-like services over the Internet. The properties of this paradigm render it as a representative paradigm for demonstrating the need for understanding and improving knowledge management in dynamic software systems. Therefore, this chapter develops a representative scenario of a volunteer storage system, which possesses the mentioned characteristics of the VC environment and exhibits its dynamic behaviour. The purpose is to utilise this scenario to demonstrate our attempt in addressing the mentioned gaps of the VC systems (and open environments in general) which relate to informing the selection and adaptation decisions through dynamic knowledge management.

Meanwhile, the fact that the volunteered resources are very heterogeneous makes resources utilisation a challenging task. That's because allocating the volunteered resources efficiently results in reducing the waste of those resources and hence saving more resources to serve more requests. Furthermore, composing the volunteer resources to serve a certain request helps to increase the percentage of satisfied requests, instead of declining the request if one volunteer host cannot satisfy the request. However, the selection of volunteer hosts should be carried out efficiently in terms of the computation time in order to reduce the waiting time of the users' requests. In this context, volunteer resources composition and allocation is an important issue to efficiently utilise the volunteered resources and increase the number of satisfied requests.

Accordingly, in this chapter, we propose a novel utility model for assessing the amount of contribution of each volunteer host to satisfy a certain request. Based on the utility model, we propose a novel greedy-search approach to select the volunteer resources as composite services to satisfy the users' requests. We also compare our approach with two basic approaches. The results show that our utility model based approach provides a systematic way for selecting the volunteer resources more effectively and efficiently while minimising resources waste when compared to the other approaches.

## **4.2 Volunteer Storage As a Service: A Steering Example**

Recent works in VC paradigm tend to view volunteer resources as services. For example, in [109] volunteer computation and storage resources are viewed as cloud services while in [108] volunteer storage resources are presented as web services. We adopt this

trend and specifically look at volunteer storage as a steering example and liken them to web services (WS). These volunteer storage services provide file manipulation operations that enable the users to access the physical storage. In this context, volunteers willing to provide their storage need to publish their service description, encoded in XML. The service description includes the service metadata, e.g. the amount of space, the availability period, etc. On the other hand, users need to use the volunteered storage, submit their requests to the volunteering system which allocates the volunteer storage, as volunteer service (VS), to the users. In cases when none of the VSs can satisfy a user's request, e.g. because the required space is higher than the available VSs spaces, VSs can be aggregated to form one composite service (CS) that satisfies the user's requirements - a practice we term as volunteer service composition (VSC), which is similar to the Web Service Composition (WSC) in service-oriented computing [124]. However, some significant differences exist which discriminate the VS from the classical WS. First, some of the attributes of the VS and WS are used in different ways. Specifically, the availability of a WS is defined as the probability that the service is accessible, computed as the total amount of time in which the service is available during a specified interval [4]. That means the WS is ideally expected to be available 24/7 and the availability value reflects the quality of the WS in terms of its availability. On the other hand, the availability of a VS includes not only the 'instantaneous' availability but also the time interval in which the service is available. Accordingly, as we mentioned in the previous chapter, the VC system needs to consider the time-interval availability of the volunteer resources before submitting the jobs. In other words, the VS is not expected to be available 24/7 and the selection of VS for composite service must consider the availability not only as a 'quality' but also as an interval in which the VS is

available to satisfy the users' requests. Second, each WS provides some specific functionality and these services are interconnected according to some business process and predefined workflow. For example, an integrated travel planning web service can be composed of a weather forecast service, a transportation service, and a hotel reservation service. But in VSs, space amounts should be combined together in specific time intervals in order to secure a total of required storage during a specific time interval. Third, WSs are usually provided by commercial providers that offer these services according to a service level agreement which provides some guarantee with regard to the quality of service (QoS), especially the availability of the service. On the other hand, a volunteer can, for example, withdraw her VS whenever she wants which makes the VC environment more dynamic and the provision uncertain. These differences call for special selection approaches for VSC.

It is worth mentioning here that the development of the selection approach of VSs for composition is required to build the motivating scenario that we use to steer the presentation of the self-aware approaches in the following chapters.

#### **4.2.1 Volunteer Storage Scenario**

In this section, we introduce the scenario of volunteered storage which motivates the VS selection approach. The case assumes a heterogeneous and dynamic environment which consists of varied computing nodes like PCs, laptops, smartphones, etc. These nodes are connected via a network. Individual people owning these nodes, known as *publishers*, offer their idle storage resources as services through a *Volunteer Storage System (VSS)* that adopts the publish-subscribe model. Users who need to use the storage are called *subscribers*. Assume a subscriber needs to do some computation and store data

temporarily but she has insufficient storage. To overcome this issue, she can explore the network and search for volunteer storage services to use. If she finds a service offering the required storage, while satisfying her requirements (e.g. availability, security etc.), she will request it for her use. Otherwise, volunteer storage services can be composed together to form a total storage that meets the subscriber's needs. Figure 4.1 shows an example in which the subscriber  $S_1$  submits a request to search for storage. To make this volume available to  $S_1$ , the composer service, named FindSpace4Me, searches the pool of published services and returns three possible composition strategies:

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

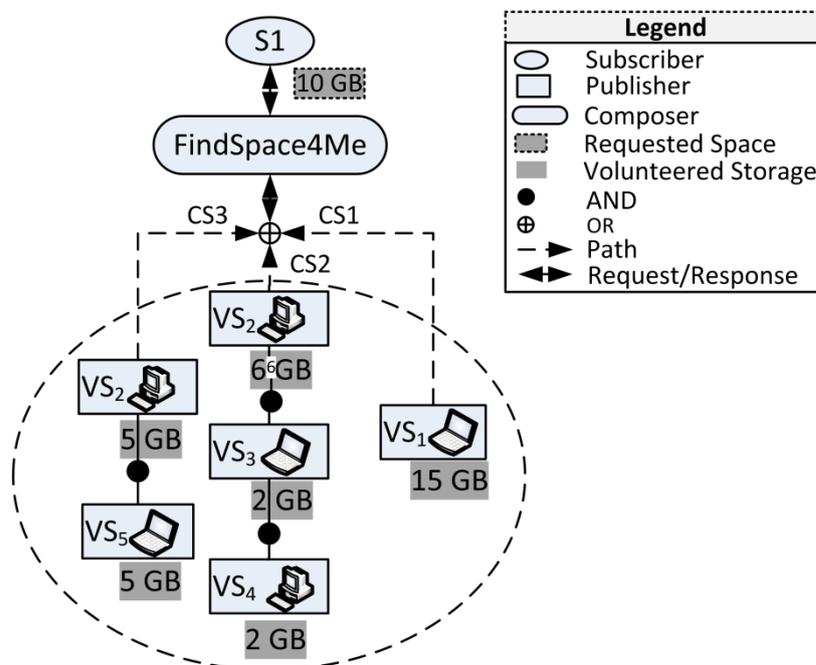


Figure 4.1: Motivating Scenario - Composition Request of  $S_1$

Now, which strategy should be selected to satisfy the request? These different composition strategies could be the result of using different algorithms for generating the composite services. The design goals for each selection algorithm, e.g. minimising computation time or maximising resource utilisation, specify which of the strategies will be selected to serve the request. For example, the availability of sufficient time for generating all the possible composite services enables the use of an algorithm that finds the ‘best’ composite service in terms of resource utilisation, e.g. by applying brute force search. Alternatively, an algorithm that is able to find a sub-optimal composite service can be applied.

In the following, we define three generic criteria for representing the volunteer storage services, namely, Storage, Availability Time, and Security. We also provide formal definitions for volunteer service, subscriber’s request, and composite service. After that, we propose a new utility-based approach for VS selection and compare it with two basic approaches, exhaustive search and a novel naïve search.

#### **4.2.2 Formulation of Volunteer Service Selection**

In this section, we introduce a set of definitions in order to formulate the problem of selecting VSs for volunteer composite services.

**Definition 4.1 (Service attributes).** In the presence of the identical functionality of the VSs, the services’ attributes are the criteria used to discriminate between services when a request is submitted. We use three generic attributes for this purpose, namely, *Storage*, *Availability Time*, and *Security*. However, other criteria can be defined without fundamental changes to the selection approaches, as shown in the next chapters.

- *Storage.* Given a volunteer service  $VS_i$ , the storage  $Stg_i$  is the volunteered storage space in Megabytes where  $Stg_i > 0$ .
- *Availability Time.* Given a volunteer service  $VS_i$ ,  $T_i$  is the time interval  $[a_i, b_i]$  in which  $VS_i$  is promised to be available, where  $a_i$  is the start time and  $b_i$  is the end time.
- *Security.* Given a volunteer service  $VS_i$ ,  $Sec_i$  is the level of security promised at the volunteer host, where  $0 \leq Sec_i \leq Sec_{max}$  and  $Sec_i, Sec_{max} \in \mathbb{N}$ .

With regard to the promised security level of a voluntarily contributed service, this is interpreted as follows. The security level indicated by the volunteer represents, ideally, the extent to which the volunteer's machine complies with the standard security guidelines and best practices [125]. Such compliance is essential as securing the VC system against the different security attacks is premier to efficiently utilise the voluntary resources. In the scope of this thesis we select simple guidelines in order to show how the security level can be specified by the volunteers, which are as follows:

- Level 0: no security promised.
- Level 1: the machine operating system is up-to-date and the system's latest security configurations are installed.
- Level 2: Anti-virus software is installed and up-to-date on the machine.
- Level 3: A comprehensive internet security software installed and up-to-date on the machine.

**Definition 4.2 (Volunteer service).** A volunteer service  $VS_i$ , is a 3-tuple  $(Stg_i, T_i, Sec_i)$  where  $Stg_i$  is the volunteered storage space,  $T_i$  is the time interval  $[a_i, b_i]$  in which the  $VS_i$  is promised to be available, and  $Sec_i$  is the service's promised security level. A service repository (SR) is a set of disjoint volunteer services. We denote a SR with  $n$  services as  $SR = \{VS_1, VS_2, \dots, VS_n\}$ . In the rest of this thesis, we denote  $Stg_i$ ,  $T_i$ , and  $Sec_i$  as the attributes of the service or the quality of the service. Figure 4.2 shows a graphical representation of a VS.



Figure 4.2: Volunteer Service Representation

**Definition 4.3 (Subscriber's request).** A subscriber's request  $R$  is a 3-tuple  $(Stg^R, T^R, Sec^R)$ , where  $Stg^R$  denotes the required storage,  $T^R = [a^R, b^R]$  is the required time interval in which  $Stg^R$  is required, and  $Sec^R$  is the required security level where  $0 \leq Sec^R \leq Sec_{max}$  and  $Sec^R, Sec_{max} \in \mathbb{N}$ . Figure 4.3 shows a graphical representation of a request.

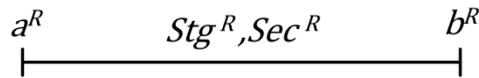


Figure 4.3: Storage Request Representation

**Definition 4.4 (Requests queue).** When a subscriber submits a request and the system is busy, the request is queued in a priority queue. Once the system becomes available

and the queue is not empty, a request is selected according to the Smallest-size Job First (SJF) policy. The request size is defined as:

$$size = stg^R \times |T^R|$$

where  $stg^R$  is the required storage in megabytes and  $|T^R|$  is the length of the required time interval in hours. The queueing model can be viewed as an M/M/1 model (according to Kendall's notation [126]) where:

- The requests arrival process follows a Poisson process with an arrival rate  $\lambda$ .
- The serving time (the time required to find a composite service which satisfies the request) has an exponential distribution with a serving rate  $\mu$ , 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.

**Definition 4.5 (Composite service).** Given a subscriber's request  $R$  and a service repository  $SR$ , a Composite Service  $CS$  is a set of VSs,  $\{VS_1, VS_2, \dots, VS_k\}$ , such that the following constraints are satisfied (denoted as  $CS \vdash R$ ):

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

Figure 4.4 shows an example of a composite service that satisfies the request  $R$  which requires 10 GB of storage with security level 2 during the time  $[10, 13]$  (we assume the time interval in hours in this example for simplicity).

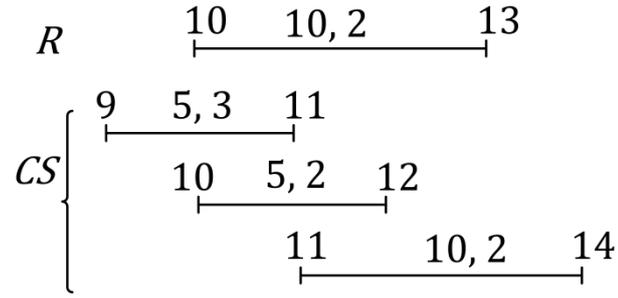
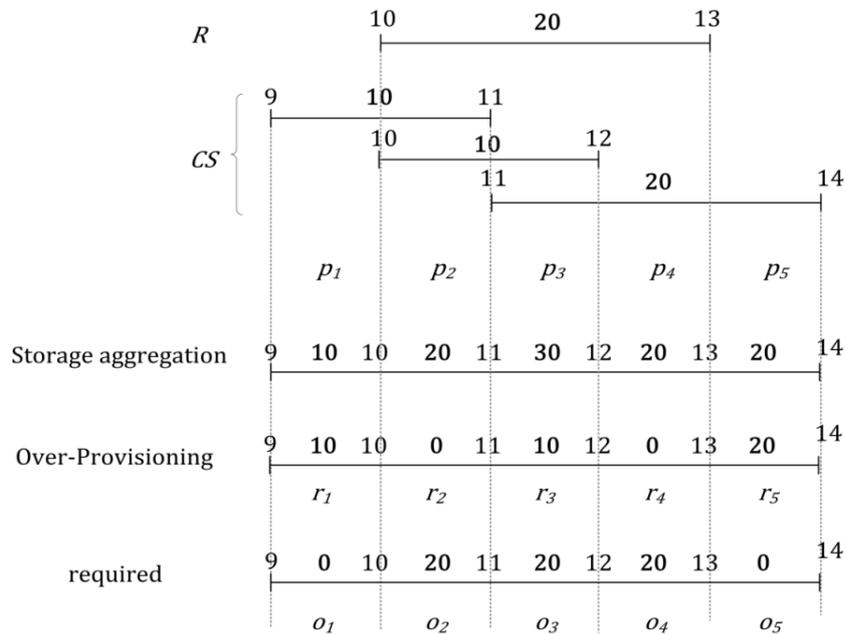


Figure 4.4: Example of a Composite Service

**Definition 4.6 (Resources waste).** Given a subscriber's request  $R$  and a corresponding composite service  $CS$ , the resources waste ( $RW$ ) is defined as the amount of over-provisioned storage of the services of  $CS$  proportional to the total amount of storage, which is allocated to satisfy  $R$ . Obviously,  $RW$  should be minimised in order to maximise the resources utilisation and save resources to serve other requests. In order to clarify how to calculate the waste, assume that a  $CS$  consists of  $k$  services. The boundaries of the time intervals, in which these services  $k$  are available, form  $P$  periods. Now, for each period  $p_i$ , compute  $o_i$  by multiplying the amount of *over-provisioned* storage in that period by the *length* of  $p_i$ . Then, the summation  $\sum_{i=1}^p o_i$  is the total *over-provisioned* storage. Similarly, compute the required storage  $r_i$  in each period  $p_i$  by multiplying the amount of *required* storage by the *length* of  $p_i$ . Then, the summation  $\sum_{i=1}^p r_i$  is the total *required* storage. After that, calculate the  $RW$  by dividing the summation of the *over-provisioned* storage  $\sum_{i=1}^p o_i$  over the summation of the required storage  $\sum_{i=1}^p r_i$ . Figure 4.5 shows an example of computing the waste in which three services have been composed to satisfy the request  $R$  (we assume the time interval in hours in this example for simplicity). In this example,  $k = 3$ , 20 GB are required in the interval  $[10, 13]$ , and

the *CS* interval is [9, 14]. The boundaries of the services of *CS* form the periods [9, 10], [10, 11], [11, 12], [12, 13], and [13, 14]. The over-provisioned storage in the period  $p_1$  is computed as the following:  $p_1$  boundaries are 9 and 10, and the bold number (10) above  $p_1$  represents the over-provisioned storage. So, the over-provisioned storage  $o_1$  during  $p_1$  is calculated as  $(10 - 9) \times 10$ . Similarly, the over-provisioned storage during  $p_2, p_3, p_4,$  and  $p_5$  is computed as 0, 10, 0, 20 respectively. Now, the required storage in the period  $p_1$  is computed as  $(10 - 9) \times 0$ , because actually no storage is required in  $p_1$ . Similarly, the required storage during  $p_2, p_3, p_4,$  and  $p_5$  is computed as 20, 20, 20, 0 respectively. Then the summation of the over-provisioned storage (40) divided by the summation of required storage in each of the periods  $p_1, p_2, p_3, p_4,$  and  $p_5$  (60) represents the total waste *RW* (0.66667).



$$\text{over-provisioning} = (10-9)(10) + (11-10)(10) + (12-11)(10) + (13-12)(0) + (14-13)(20) = 40$$

$$\text{required} = (10-9)(0) + (11-10)(20) + (12-11)(20) + (13-12)(20) + (14-13)(0) = 60$$

$$RW = \text{over-provisioning} / \text{required} = 40/60 = 0.6667$$

Figure 4.5: Waste Computation Example

**Definition 4.7 (Waiting time).** Given a subscriber's request  $R$  and a service repository  $SR$ , the waiting time  $WT$  is defined as the time needed by the system to find a  $CS$  that satisfies  $R$  in seconds. If the system failed to find a  $CS$  due to a lack in the resources, the system inserts them into the tail of the request queue. After three failed trials, the request will be removed from the queue.

**Definition 4.8 (Percentage of satisfied requests).** Given  $m$  requests, the percentage of satisfied requests  $PSR$  is defined as the number of requests that the system satisfied successfully divided by the total number of requests  $m$ .

Based on the above definitions, given  $n$  volunteer services and  $m$  subscribers' requests, the system's goal, ideally, is to produce composite services for the requests such that the following function,  $g$ , is minimised, where the values of the weights  $W_i$  express the system administrator's preferences regarding the  $RW$ ,  $WT$ , and  $PSR$ . However, finding the optimal solution for this objective function faces scalability problems when the  $n$  increases, as shown in the next section.

$$\begin{aligned} & \text{minimise} && g = (f_1, f_2, f_3) \\ & \text{subject to:} && PSR > 0 \end{aligned}$$

$$\text{where: } f_1 = W_1 \frac{\sum_m RW}{m}, f_2 = W_2 \frac{\sum_m WT}{m}, f_3 = W_3 \frac{1}{PSR}, \sum_{i=1}^3 W_i = 1 \quad (4.1)$$

### 4.3 Services Selection for Volunteer Composite Services

In this section, we propose a novel utility model for quantifying the contributions of the VSs to satisfy a certain request. Then, based on the utility model, we propose a systematic greedy approach for the selection of VSs for volunteer composite services. However, before reaching the utility model, we scope two basic selection approaches,

namely, *Exhaustive search* and *Naïve search* for the benefit of the utility model. The experimental evaluation provides evidence that utility-based approach outperforms the other two approaches in terms of the *WT*, *PSR*, and *RW*.

#### 4.3.1 Exhaustive Search

One possible solution to find a CS is to perform an exhaustive search to search through the possible composite services. In this approach, for each request  $R$ , the system finds all possible sets of CSs and extracts the CSs that satisfy the request. Then, the system selects the optimal CSs; the one that satisfies the request with minimum resource waste. If no permutation satisfies the global constraints, the system notifies the subscriber that the request  $R$  cannot be satisfied. For example, assume we have a request  $R$  and a set  $S$  of published VSs:  $S = \{VS_1, VS_2, VS_3\}$ . Then exhaustive search will generate the following permutations:  $\{VS_1\}$ ,  $\{VS_2\}$ ,  $\{VS_3\}$ ,  $\{VS_1, VS_2\}$ ,  $\{VS_2, VS_3\}$ ,  $\{VS_1, VS_3\}$ ,  $\{VS_1, VS_2, VS_3\}$ . After that, for each permutation, the CS constraints are checked (see Definition 4.5). Assume here that  $\{VS_2, VS_3\} \vdash R$ ,  $\{VS_1, VS_3\} \vdash R$ , and  $\{VS_1, VS_2, VS_3\} \vdash R$ . Then the waste is computed for these permutations, as shown in the previous section, and the permutation that shows minimum waste will be returned as  $CS \vdash R$ .

It is obvious that exhaustive search will give the optimal CS in terms of minimising the resources waste. However, it is expensive in terms of computation time. It can be applicable only if the number of published VSs is very low. Otherwise, we need other practical approaches even if they are sub-optimal in terms of over-provisioning.

### 4.3.2 Naïve Search

Random assignment has been used in VC systems [127] [128] to randomly send computational tasks to the worker hosts. In the context of volunteer storage services the, e.g. the volunteering case of the SocialCloud [123], market-based selection approaches are adopted. Since the services will be advertised for no gain, the user can select any of those services; a case which we liken it to the random assignment. In the context of VSC, we use the random assignment approach to develop a *naïve* service selection for volunteer composite services, which can be applied as follows. The system iterates over the published services and selects any service that satisfies any part of the required availability period as the first service of the CS. Then, the order of the published services is changed randomly, and another service is selected. After the selection of each service, the CS constraints are evaluated (see Definition 4.5). When the set of selected services satisfies these constraints, the set is returned as the CS. If no composite service can be found, the system notifies the subscriber that the request cannot be satisfied. For example, assume we have a request  $R$  and an ordered list  $L$  of published VSs:  $L = \{VS_1, VS_2, VS_3, VS_4, VS_5\}$ . Then the random assignment search will randomly pick one VS, e.g.  $VS_3$ , and add it to the set  $CS$ . Then the global constraints will be checked. If they are not satisfied, another VS will be picked randomly, e.g.  $VS_2$  and added to  $CS$ , so  $CS = \{VS_3, VS_2\}$ . Then the global constraints will be checked. The process continues until  $CS \vdash R$ . If the all services in  $L$  are selected and the global constraints are still not satisfied, the subscriber will be notified that the request  $R$  cannot be satisfied. The detailed selection algorithm is shown in Algorithm 4.1.

The naïve search can be in exchange for the exhaustive search approach, especially in the large-scale cases. However it is an ad-hoc approach, i.e. the performance of the

system can fluctuate from time to time. Therefore, more systematic approaches are required.

---

Algorithm 4.1: Naïve Search volunteer service selection

---

**Input:** A list of Volunteer Services  $L$ , A request for storage  $R$ .  
**Output:** A composite service  $CS = \{VS_1, VS_2 \dots VS_k\} \vdash R$  OR *null*.  
**Begin**

- 1      $CS: \{\}$
- 2      $tmpRequest = R$ ;
- 3     **While** ( $L$  is not Empty)
- 4         If the availability interval of  $VS_i$  intersects with the requested interval  
        and  $sec_i \geq sec^R$
- 5         Add  $VS_i$  to  $CS$
- 6         Remove  $VS_i$  from  $L$
- 7         Find the unsatisfied intervals of  $tmpRequest$
- 8         **If** all  $tmpRequest$  intervals are satisfied
- 9             Return  $CS$
- 10         **Else**
- 11             Recalculate  $tmpRequest$
- 12             Randomise the List  $L$  order
- 13         **End If**
- 14     **End While**
- 15     Return *null*

**End**

---

### 4.3.3 A Utility Model for Volunteer Composite Services

We propose a utility-based approach which provides a systematic method for services selection. The idea is to quantify the amount of contribution that each service exhibits to satisfy the request. Then the greedy selection starts from the services that contribute 'best' to the request. In this section, we present our proposed utility model and the corresponding selection algorithm.

#### 4.3.3.1 Utility Model

The utility model expresses the amount of contribution that each VS exhibits to satisfy a request as a utility value taking into account each of the VS attributes. The utility model is designed to assign maximum utility to services which promise to provide the amounts of resources that exactly match the subscriber requirements. That is, services that provide higher or lower than the required resources are assigned lower utilities thus reducing their probabilities to be selected to serve the corresponding request. The purpose is to maximise the utilisation of resources by serving each request by roughly as much as it needs.

**Storage utility.** Given a volunteer service  $VS_i$  and a request  $R$ , the storage utility  $U_{stg}(VS_i)$  of  $VS_i$ , defined in (4.2), measures the amount of storage contributed by  $VS_i$  to  $R$ . Figure 4.6 plots  $U_{stg}$  over services volunteer storage space. This utility function gives maximum value of '1' if  $Stg_i = Stg^R$  and a value less than '1' otherwise. The parameters  $\beta$  and  $\alpha$  are set by the system administrator to specify whether to give utility values to services with  $Stg_i > Stg^R$  higher than the utility values to services with  $Stg_i < Stg^R$  or not, 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.

$$U_{stg}(VS_i) = \begin{cases} e^{-\beta(Stg_i - Stg^R)}, & Stg_i \geq Stg^R \\ e^{\alpha(Stg_i - Stg^R)}, & Stg_i < Stg^R \end{cases} \quad (4.2)$$

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

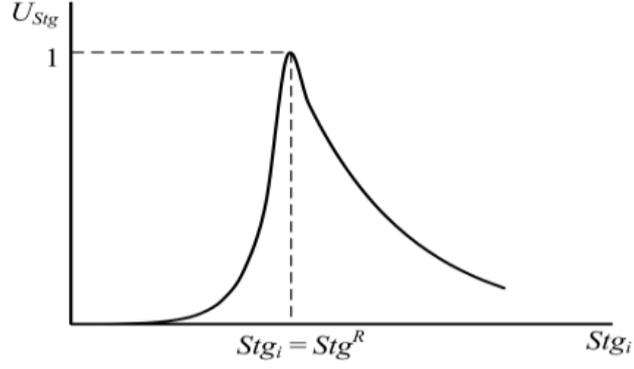


Figure 4.6: Services Storage Utility

**Availability Time Utility.** Given a volunteer service  $VS_i$  and a request  $R$ , the time utility  $U_{time}(VS_i)$  of  $VS_i$ , defined in (4.3), measures the amount of time contributed by  $VS_i$  to  $R$ . The 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. Figure 4.7 shows an example of the time utility for different services applying (4.3). For example, since  $a_1 = a^R$  and  $b_1 = b^R$ ,  $VS_1$  is assigned a utility equals '1'. Another example is  $VS_3$ , since  $a_3 < a^R$  and  $b_3 > b^R$ , it is assigned value less than '1'.

$$U_{time}(VS_i) = \begin{cases} 0, & b_i \leq a^R \text{ or } b^R \leq a_i \\ e^{\gamma(b_i - b^R)}, & a_i < a^R, a^R < b_i < b^R \\ e^{\gamma(a^R - a_i)}, & b_i > b^R, a^R < a_i < b^R \\ \frac{e^{\alpha(b_i - a_i)}}{e^{\alpha(b^R - a^R)}}, & a_i \geq a^R, b_i \leq b^R \\ \frac{e^{-\beta(b_i - a_i)}}{e^{-\beta(b^R - a^R)}}, & \text{otherwise} \end{cases} \quad (4.3)$$

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

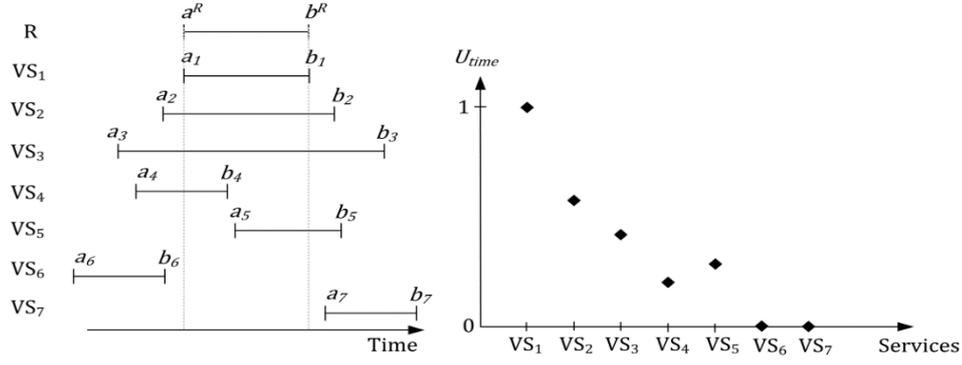


Figure 4.7: Services Time Utility Example

**Security Utility.** For the scope of this thesis, we consider the security as the extent to which the volunteer host adhere to the security best practises (e.g. having the up-to-date system security configurations and up-to-date Antivirus and Internet security software). A script similar to many existing tools that perform an analysis of the computer machines security baseline, e.g. [137], can be used to assess the security of the volunteer service. In this context, we define security utility as follows. Given a volunteered service  $VS_i$  and a request  $R$ , the security utility  $U_{sec}(VS_i)$  defined in (4.4) compares between the security level provided by  $VS_i$  and the requested level. Figure 4.8 plots  $U_{sec}$  over services security levels. The function in (4.4) 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  $Sec_i = Sec^R$  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.

$$U_{sec}(VS_i) = \begin{cases} 1 - \Delta u(Sec_i - Sec^R), & \text{if } Sec_i \geq Sec^R \\ 0, & \text{otherwise} \end{cases} \quad (4.4)$$

$$\text{where } \Delta u = \frac{1 - \epsilon}{Sec_{max}}, \quad 0 < \epsilon < 1$$

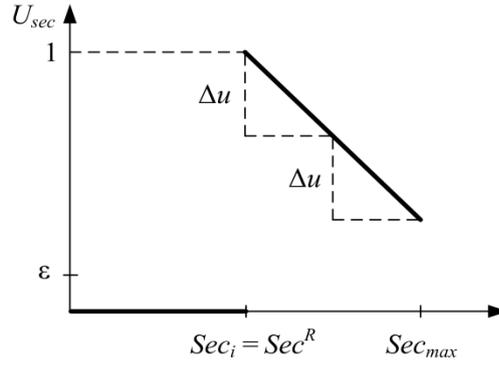


Figure 4.8: Services Security Utility

#### 4.3.3.2 Utility-based Greedy Selection Approach

When a subscriber submits a request, the system retrieves the available services from the service repository and creates an empty CS. Then the system computes the utility for each service in order to use these utilities as selection criteria. After that, the system finds the non-dominant set of services, i.e. the Pareto-optimal choices [129], and selects the highest utilities service using (4.5) and adds it to CS.

$$\begin{aligned}
 & \text{maximize} \{U_{Stg}(VS_i), U_{time}(VS_i), U_{sec}(VS_i)\} \\
 & \text{s. t. } U_{sec}(VS_i) > 0
 \end{aligned} \tag{4.5}$$

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 selection algorithm is shown in Algorithm 4.2 which includes the following steps:

- Step 1 (lines 1-9). Create an empty composite service CS, clone the request object, and create a list for storing the services' utilities. Then iterate over the services and compute the storage, time, and security utilities.

- Step 2 (lines 10-22 ). Find the non-dominant set of services, select one of them randomly and add it to  $CS$ . This results in partial satisfaction of  $R$ , i.e. the selected service can serve the subscriber with some storage space during some time; according to its promised storage space and availability time. Accordingly, the sub-intervals of the request which still unsatisfied need to be determined (line 15), if any. In case of the selected service provides the required storage space over the request interval,  $CS$  will be returned (line 17). Otherwise, the request needs to be recalculated to exclude the satisfied sub-intervals (line 19) and consequently the utilities will be recomputed (line 20).

---

Algorithm 4.2: Utility-based Volunteer Service Selection

---

**Input:** A list of Volunteered Services  $L$ , A request for storage  $R$ .  
**Output:** A composite service  $CS = \{VS_1, VS_2 \dots VS_k\} \vdash R$  OR *null*.  
**Begin**

```

1   $CS: \{\}$ 
2   $tmpRequest = R$ ;
3  Create an empty List  $UL$ 
4  For all  $VS_i$  in  $L$ , do
5      Compute the storage, time, and security utilities using (4.2), (4.3), and
6      (4.4) respectively
7      If  $U_{time}(VS_i)$  and  $U_{time}(VS_i) > 0$ 
8          Add  $VS_i$  and its utilities to  $UL$ 
9      End If
10 End for
11 While ( $UL$  is not Empty)
12     Find the Pareto-optimality set
13     Select one service,  $VS_i$ , from the Pareto-optimality set using (4.5)
14     Add  $VS_i$  to  $CS$ 
15     Remove  $VS_i$  from  $UL$ 
16     Find the unsatisfied intervals of  $tmpRequest$ 
17     If all  $tmpRequest$  intervals are satisfied
18         Return  $CS$ 
19     Else
20         Recalculate  $tmpRequest$ 
21         Re-compute the storage, time, and security utilities.
22     End If
23 End While
Return null

```

**End**

---

- Step 3 (line 23). Reaching this line means that no composite service has been found to satisfy the request  $R$ . In such case the system will notify the subscriber to relax the requirements or try later.

#### **4.4 Experimental Evaluation**

In this section, we conduct experiments in order to evaluate the performance of the proposed utility-based selection approach and compare it with the executive search and the naïve search approaches.

Generally, research in VC is experimental in nature [130]. Taking into consideration that hosts in VC are individually owned edge devices, it is almost obvious that having a reasonable number of volunteers host to conduct repeatable and scalable experiments is practically not achievable. In this context, open-source computing platforms, e.g. BOINC [131] OpenNebula [132] and Eucalyptus [133], do not help us. Therefore, as most VC researchers do, we resort to simulation-based evaluations [130] so that repeatable and scalable experimentation is manageable. However, the simulation results can be used to guide the application of the selection approaches in real-world scenarios.

In the same vein, using simulators such as SimGrid [134] and GridSim [135] would require major modifications to support the greedy approach for storage services compositions as these simulators are mainly designed to simulate distrusted computational algorithms in grid environments. The main goal of these simulations is to evaluate the scheduling algorithms used in the grid environment.

Based on the above context, we wrote a simulator for VS in Java V1.7.0. The experiments were conducted on a desktop PC with an Intel core i5-3570 3.5 GHZ processor, 4G RAM, Windows 7.

The experiments implement the scenario described in section 4.2.1 as a publish/subscribe model in which  $n$  services are published and  $m$  requests are submitted. A service is represented as a tuple of the attributes: Storage, Availability Time, and Security Level. The experiments are conducted using synthetic data generated based on data distributions reported in related studies, e.g. [104] [5]. The volunteered *storage* is assumed to follow a uniform distribution with *expected value*  $\mu = 10\text{GB}$  and standard deviation  $\sigma=2$  [104]. The time intervals' bounds and the security level values are generated randomly. Table 4.1 shows the ranges of the services' and requests' attributes values. A subscriber's request is represented in the same way as services. 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. For each test case, the experiment was conducted 100 times and the average was computed. In these simulations, we assume that the volunteer services will provide the promised resources once they are allocated to satisfy a certain request.

Table 4.1: Utility Model Attributes' and Parameters' Values

	Service		Subscriber	
Attribute	<i>min</i>	<i>max</i>	<i>min</i>	<i>max</i>
<b>Storage</b>	1	20	1	40
<b>Availability Time</b>	1 Jan. 00:00	31 Dec. 23:59	1 Jan. 00:00	31 Dec. 23:59
<b>Security</b>	0	3	0	3
$\alpha$	0.1			
$\beta$	0.1			
$\gamma$	0.1			
$\epsilon$	0.2			

In accordance with this thesis objective, we investigate the effectiveness of using the utility model for VS selection. The effectiveness is measured in terms of resources waste, waiting time, and percentage of satisfied requests, which are defined above in section 4.2.2. We also empirically investigate the effect of scale on the performance of the utility-based selection approach by increasing the number of services and requests. It is worth noting here that our focus is on the quantification of the promised contributions of the services as a basis for selection. Consequently, the main objective of the evaluation is to evaluate the performance of the three selection approaches in terms of the above criteria, and thus the actual data storage and the related aspects (e.g. data transfer and data integrity) are not considered.

#### **4.4.1 Experimental Results**

**Comparing the Resources Waste (RW).** The first set of experiments evaluates the efficiency of allocating resources to the subscribers. The experiments were conducted in two parts. The first part evaluates the RW for serving one request whereas the number of services is varied. The second part evaluates the RW for serving a varying number of requests in the presence of varying number of services. This part involves only the utility-based method and the naïve search method. The exhaustive search has not been involved because the computation time is ‘infinite’ in high-scale cases. Figure 4.9 plots the RW for the three approaches. In this experiment, we vary the number of published services  $n$  and set the number of requests to 1. The results show that the RW in the exhaustive search case is always the minimum. This is obvious because the exhaustive search checks all the composition possibilities and returns the optimal composition. However, because the exhaustive search is not scalable, the results are not obtainable

for the case when  $n > 15$ , due to ‘infinite’ computation time. Also, the figure clearly shows that the RW in the utility-based case is always less than the naïve search case.

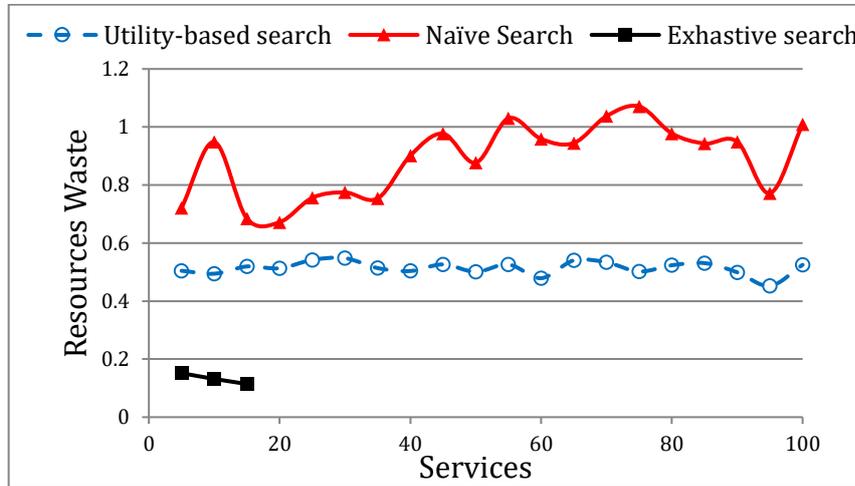


Figure 4.9: Average Waste in Small-Scale Experiment

Figure 4.10 compares the RW between the utility-based search and the naïve search approaches where we vary the number of published services  $n$  and the number of requests. The waste of the naïve search case is higher than the utility-based case especially when the number of requests is low, relative to the number of published services. The reason is that, the utility-based search is able to find the services that ‘best’ contribute to the subscriber’s requirements and avoids selecting services that provide resources higher than the needed to serve the request.

**Comparing the Percentage of Satisfied Requests (PSR).** The second set of experiments evaluates the number of requests that each approach can satisfy proportional to the total number of submitted requests. Figure 4.11 compares the PSR of the three approaches. In this experiment, we vary the number of published services  $n$  and set the number of requests to 1. The results show that the exhaustive search is able

to satisfy more requests, however, it does not scale. The utility-based search exhibits higher PSR than the naïve search approaches when the number of published services is not high. But, when the number of services is high, relative to the number of requests, the two approaches perform equally. The reason is that the high number of services increases the possibility of finding services that can contribute to satisfy the request, i.e. they are available during the request time interval and their security level is greater than or equal to the requested security level.

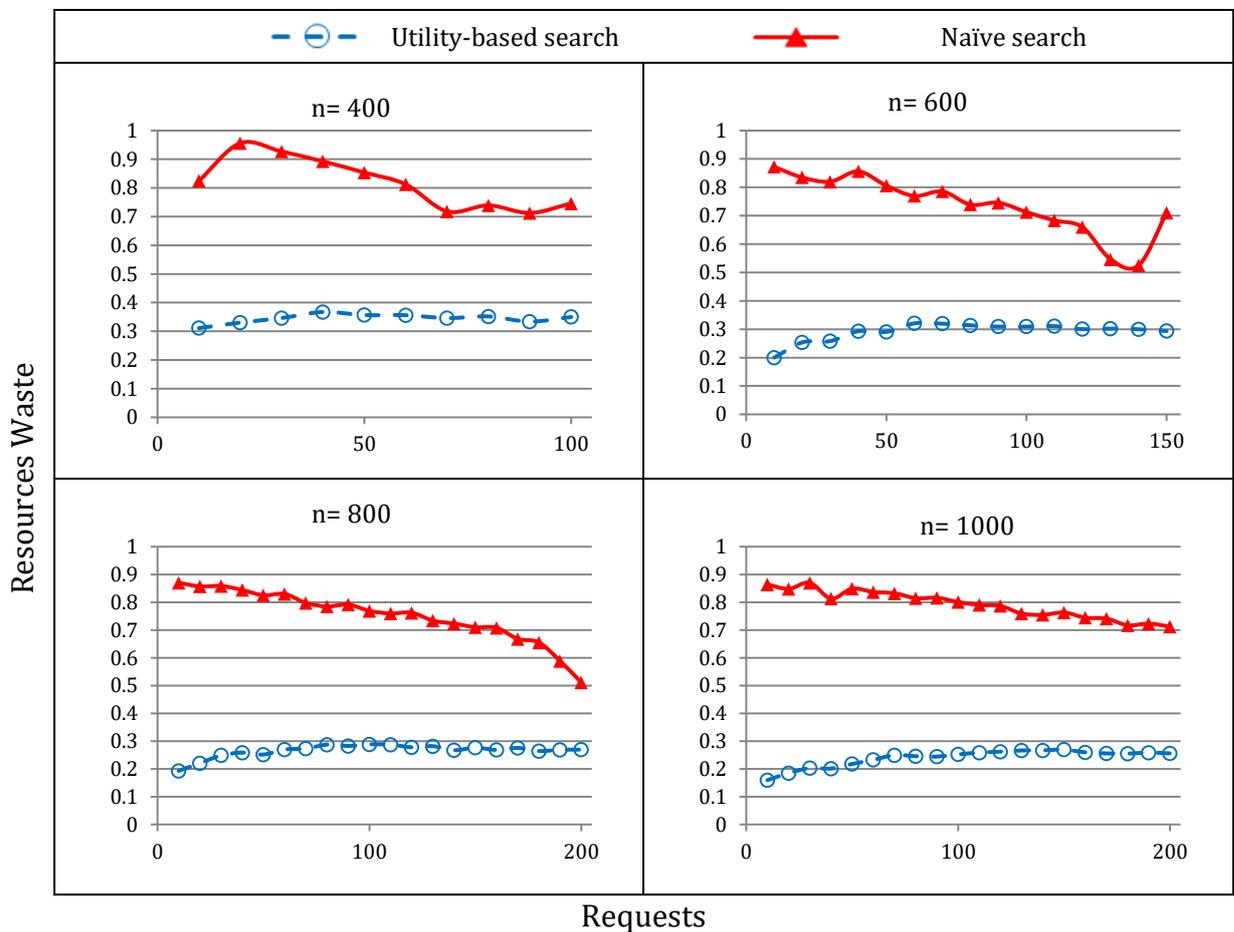


Figure 4.10: Average Waste in High-Scale Experiment

Figure 4.12 shows more results of comparing the naïve search and the utility-based search in a high-scale case. In this experiment, we vary the number of published services

$n$  and the number of requests. The figure shows that in the cases of low number of requests, the two approaches can equally satisfy the requests. But, as the number of requests increases, the ability of the naïve search approach to satisfy the requests drops significantly. On the other hand, the utility-based search satisfies high number of requests. The reason is that the utility-based search selects services that best contribute to the request with respect to the resources needed resulting in saving more services to serve other requests.

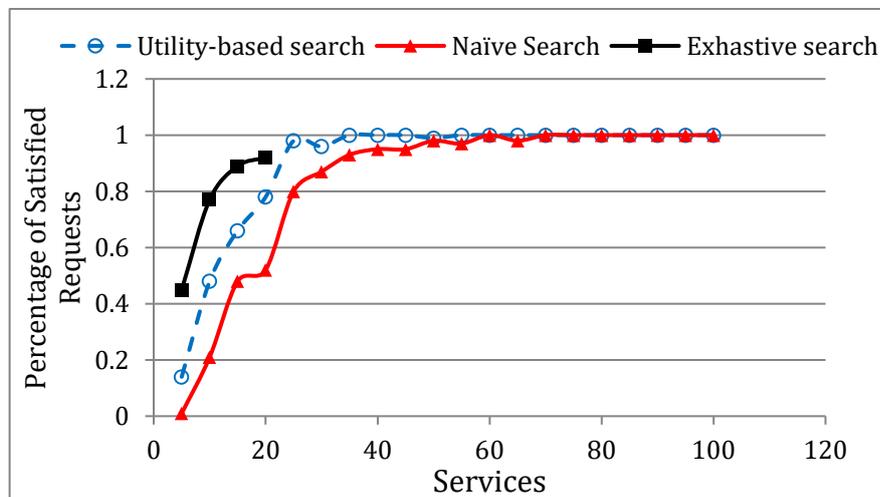


Figure 4.11: Average PSR in Small-Scale Experiment

**Comparing the Waiting Time (WT).** The third set of experiments evaluates the WT of the approaches. Figure 4.13 plots WT in seconds. In this experiment, we vary the number of published services  $n$  and set the number of requests to 1. The figure shows that in the exhaustive search approach the WT increases exponentially in very small scale. On the other hand, the WT is low in the cases of the utility-based search and the naïve search. Therefore, the exhaustive search is not a practical approach, especially when the number of published services is high.

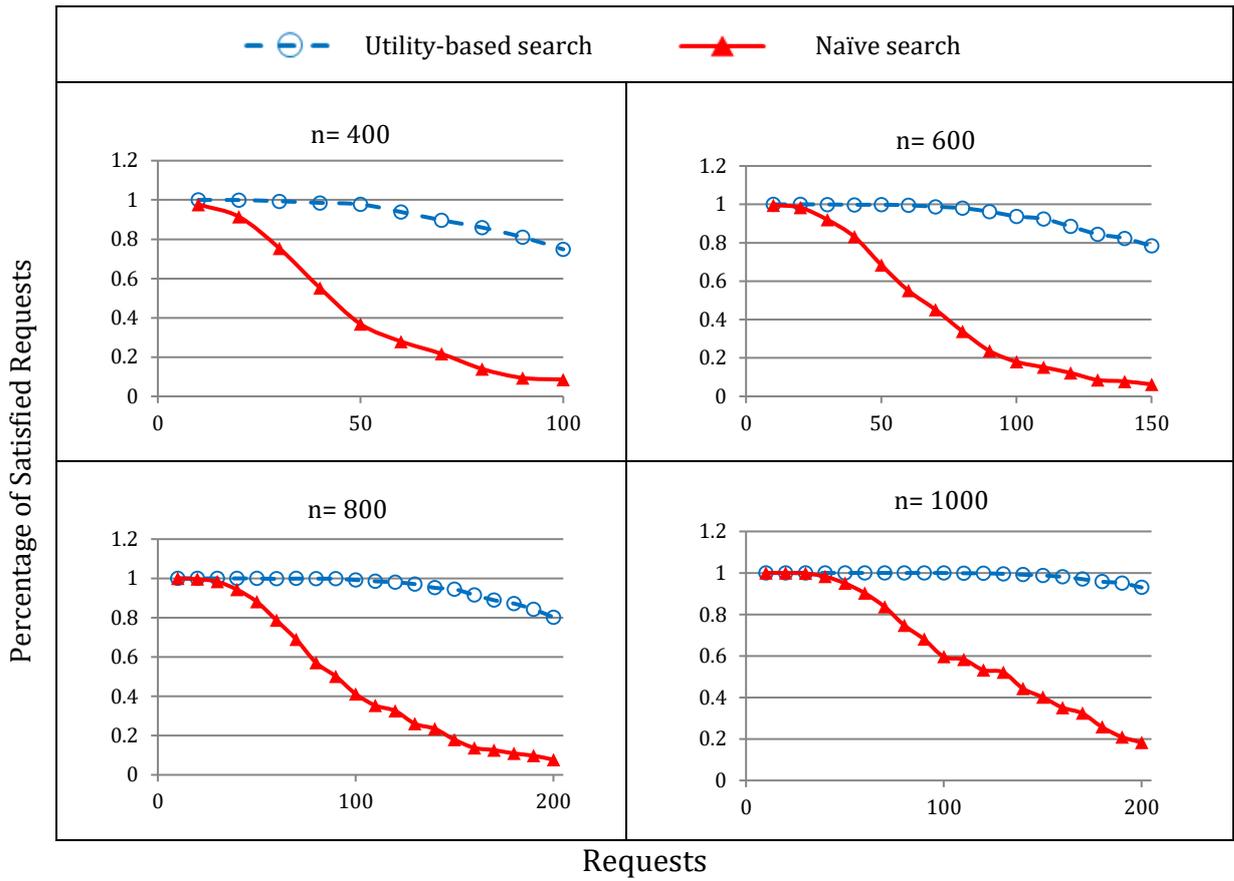


Figure 4.12: Average PSR in High-Scale Experiment

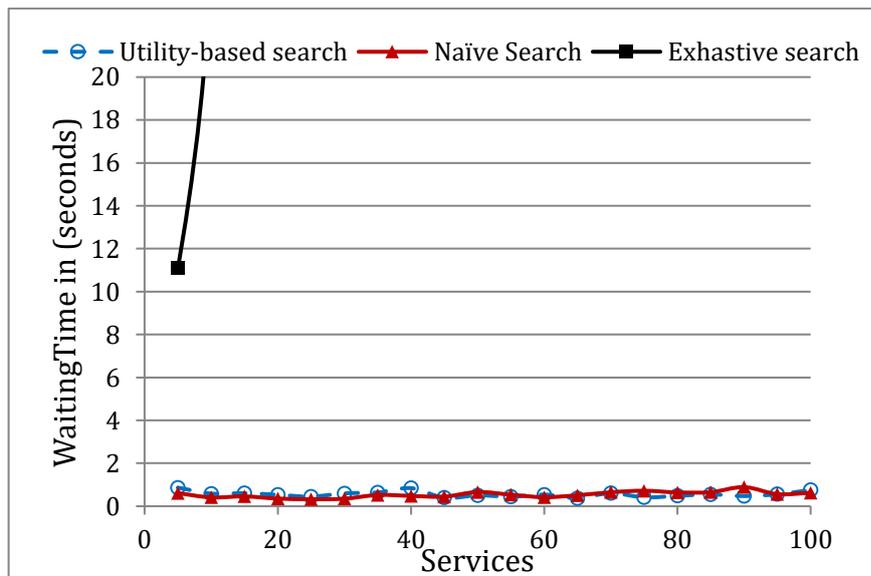


Figure 4.13: Average Waiting Time in Small-Scale Experiment

Figure 4.14 plots WT in seconds in case of varying number of requests. In this experiment, we vary the number of published services  $n$  and the number of requests. The comparison includes the utility-based search and the naïve search approaches. The figure clearly shows that the WT of the utility-based search is always lower. The reason is that services that contribute more to the request are chosen first (refer to Algorithm 4.2), which reduces the iterations that are required to find the CS.

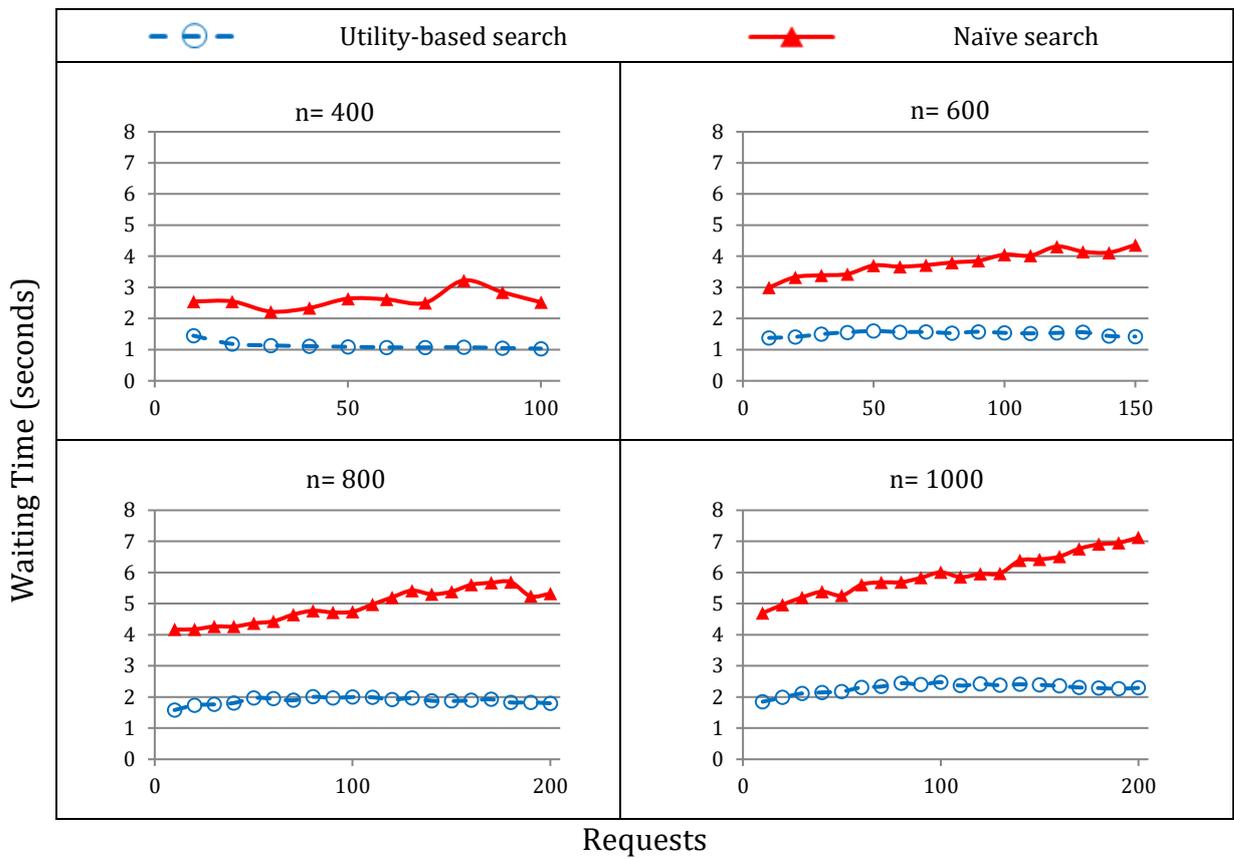


Figure 4.14: Average Waiting Time in High-scale Experiment

## 4.5 Conclusion

An efficient VS selection and allocation approach is required to utilise the volunteered resources and increase the number of satisfied requests. Current approaches in VC either do not address the composition problem or select the services in a ‘random’ way;

resulting in a low percentage of satisfied requests. Therefore, in this chapter, we have proposed a utility-model to enable quantifying the contribution that each VS makes to a certain request. Based on the utility-model, we developed a greedy approach for selecting the VSs to satisfy subscribers' requests. One of the core advantages to this approach is its applicability in large-scale cases while minimising the over-provisioning of resources. We have experimentally evaluated the approach and compared it to related approaches, namely, the exhaustive search and the naïve search approaches. The results show that the exhaustive search is the most effective at minimising the waste in resource provisioning and maximising the number of satisfied requests. However, the exhaustive search is not applicable in environments where the number of publishers and/or subscribers is not very low. The results show also that the utility-based approach performs well in small-scale environments and best in large-scale ones, relative to the naïve search and the exhaustive search approaches.

So far, we assumed that the VSs will provide what they promise. In other words, the proposed approach does not deal with the dynamism of the environment. Taking into consideration that volunteers can offer and withdraw their services at any time adds a significant challenge to the problem. In this context, the proposed utility-based selection approach will form the basis for extending this work. The extension will aim at providing self-adaptation capabilities (through self-awareness with dynamic knowledge management) to deal with the dynamism of the environment. Such self-adaptation capabilities should enable the system to satisfy the subscriber's requests in the presence of the dynamic changes in VSs performance.



# CHAPTER 5

## SELF-AWARE FRAMEWORK FOR VC WITH DYNAMIC KNOWLEDGE MANAGEMENT

### 5.1 Overview

As mentioned in chapter 3, volunteer services tend to be published and withdrawn without restrictions, thus, uncertainties, dynamisms, and dilution of control, related to the decisions of selection and composition, are complex problems. Taking these challenges into consideration, a handful of VC systems have been contributed along with self-adaptation capability e.g. [101] [107] [116]. However, these systems make simple assumptions about knowledge representation and management for the process of decision making. As a result, these VC systems tend to be limited in their adaptation capabilities, which are restricted to re-allocating the resources when a host fails. Such weakness is due to a lack of their architectures' capabilities in capturing and representing the evolving knowledge at runtime; a gap where self-awareness can be rendered to overcome such weakness.

Meanwhile, self-awareness has been given more attention in recent research efforts as an enabler for self-adaptation; resulting in a handful of conceptual approaches for engineering self-awareness in computing systems e.g. [16] [18] [14] [136]. Some of these recent research works, e.g. [18] [14], have suggested that finer knowledge representation can better address the users' and systems' requirements in environments that exhibit uncertainty and dynamism and can improve the quality and accuracy of adaptation. Although we agree with this suggestion, we argue that, as mentioned in chapter 2, dynamic knowledge management has been given little consideration; though it is a vital requirement for self-awareness. The knowledge should be treated as moving targets that can change and evolve over time. Therefore, self-aware systems need to be able to capture the evolution trends and use this information to better inform the adaptation decision.

Given the above background, the contributions of this chapter are as follows:

- (1) A general architecture of VC systems is presented with mapping a self-aware framework for VC to enrich the self-adaptation capabilities of VC systems.
- (2) Dynamic knowledge management approaches are proposed to improve the self-aware framework and enact self-awareness. This improvement is driven by the VC scenario; however, it is fundamental to the self-awareness as it equips the self-aware framework with dynamic knowledge management capabilities that they do not have. At the same time, this improvement benefits the VC systems to fertilise their self-adaptivity. The dynamic knowledge management approaches use dynamic data structures, namely the *dynamic histograms*, to represent the knowledge. The dynamic histograms are able to capture the evolving

performance patterns of the VSs. Algorithms for the dealing with the evolution of the dynamic histograms are developed.

## 5.2 Motivation for Self-awareness

The utility-based approach, proposed in the previous chapter, provides a systematic approach for the selection of VS based on the utilities that a VS promises to provide. However, as discussed in chapter 3, volunteers deviate from their promises and therefore the VSs performance tends to exhibit uncertainty and violation of the promised quality. Furthermore, it was shown in chapter 3 that the VSs tend to exhibit periodic performance patterns, which are often repeated over a certain time period. Consequently, the awareness of such periodic patterns enables the prediction of the performance of the services leading to better adaptation. Furthermore, as these services do not work in isolation (as mentioned in chapter 3), the awareness of the correlation between the services' performance is necessary for satisfiable service provision.

Accordingly, such cases motivate the need for more 'intelligent' selection and adaptation approaches to deal with such dynamism to mitigate the corresponding consequences. To clarity, we refer to the volunteer storage scenario of Figure 4.1 and re-ask the following question, *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, a question arises here about the feasibility of that adaptation action, i.e. will the undertaken adaptation result in better performance? Another possibility is that, 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. Moreover, the deeper the knowledge the system has on the performance of the services (e.g. capturing the correlation between the services' performance), the more intelligent the decision will be. Here where self-awareness can be adopted to reason about the self-adaptation actions; enabling intelligent selection and adaptation decisions. To clarify the above with an example, refer to Figure 4.1 and 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 assume that the performance of  $VS_1$  is anticipated to be well at  $t_2$ , then the system will select the third strategy. But, if we assume that  $S_1$  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 may select the third strategy. However, if  $VS_5$  exhibited poor performance when composed with  $VS_2$ , then the system will select the second strategy. This example shows that decomposing knowledge about the service performance to fine grain increases the system's awareness and 'better' informs the selection and adaptation decisions. In conclusion, the VS selection scenario motivates the need for self-awareness and its engineering principles. It also requires dynamic knowledge management for capturing and modelling the performance trends to realise self-awareness.

### **5.3 General Architecture for Self-aware Volunteer Computing**

Drawing on conclusions from chapter 3, we identified common features which relate to a general architecture of VC systems. Figure 5.1 abstracts some of the essential components which tend to be present in a typical volunteer storage system (VSS).

Basically, the architecture involves three parts, the volunteering part, the usage part, and the management part.

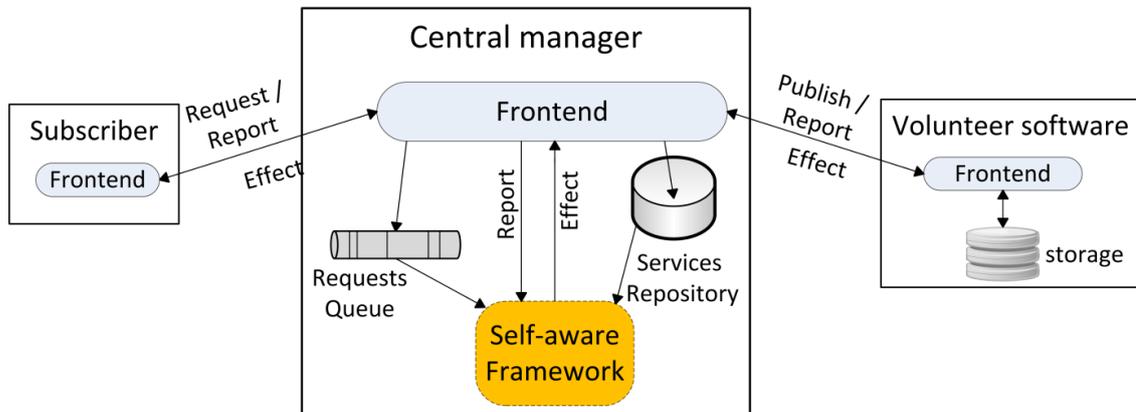


Figure 5.1: Architecture for Self-aware Volunteer Storage System

Based on this general architecture, we have realised the VSS as a web-service based system. On the one hand, the system aggregates the published volunteered storage from the volunteers end devices. On the other hand, it offers the aggregated storage in a service-oriented perspective as composite services (CSs); based on the subscribers' requirements. Volunteers willing to provide their storage need to sign up to register their identities. After that, they need to download and install volunteering software in order to interact with the VSS and the subscribers. After installing the volunteer software, the volunteer can sign in to the VSS and advertise her VS's information which will be encoded using XML-based metadata and submitted to the VSS. Subscribers also need to sign up to the VSS to register their identities and then submit their storage requests. Having received a request, the system uses the utility-based greedy approach, proposed in the previous chapter, to find a VS or a CS for that request. To improve the effectiveness of the architecture to deal with volunteering at scale, it is imperative that

intelligence is needed to seamlessly manage these resources. For this purpose, we ‘plug-in’ a self-aware framework to provide such intelligence, building on the utility-based VS selection approach. The general architecture of the VSS (shown in Figure 5.1) consists of the following parts, volunteer software, subscribers’ frontend, and the central manager:

**Volunteer Software**        The volunteer software should be installed on the volunteer physical machine to enable (1) the interaction between the volunteer machine and the VSS central manager, and (2) the access to the actual storage by the users. The volunteer interaction with the central manager involves the following:

- (1) Submitting the VS’s attributes, specifically, the volunteered storage space  $stg$ , the availability period  $T = [a, b]$ , the security level  $sec$ , and the binding information required to access the physical storage location. This software will be periodically assessing the security level of the volunteer’s machine and reporting any violation of the promised security level. This is realised by a script that checks the state of volunteer machine in terms of following the standard security guidelines and best practices, e.g. having the up-to-date system security configurations and up-to-date Antivirus and Internet security software. Such script is similar to many existing tools that perform an analysis of the computer machines security baseline, e.g. [137].
- (2) Periodically sending an “I am alive” heartbeat during the availability period to confirm the availability to the central manager.

The interaction between the volunteer’s machine and the subscribers is realised through the VS. The volunteer software installs a web server that hosts the VS. 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.

**Subscriber's frontend** A subscriber interfaces with the VSS through the frontend to submit requests and report the subscriber's feedback to the VSS. Having logged into the system, the subscriber specifies her storage requirements and submits her requests. The submitted requests will be sent to the system queue. The VSS processes each request in the queue to find a VS or a CS that satisfies that request. Having found a VS or a CS, the system encodes it as XML-based metadata, which provides the information required to access the VSs. Then, the system sends the metadata to the subscriber's frontend to enable access to the actual storage through a web interface. Also, the VSS provides an interface for the subscribers to rate the VS reputation after using the VSs. The reputation reflects the subscriber's experience of using the VSs. This subscriber's feedback will be used by the self-awareness approaches for informing the selection of the VSs. In summary, the frontend enables sending feedback on each VS to the VSS that involves:

- (1) Reporting any violation in the promised storage space, this is done 'automatically' by the frontend.
- (2) Reporting the VS reputation level  $Rep(VS_i)$ , which is done manually by the subscriber. The reputation level has the value of 1 if data storage/retrieval was perfect, 0.5 if any minor data loss/corruption occurs, or 0 if a major data loss/corruption occurs. Based on the subscribers' feedback, the representative reputation of a certain service  $VS_i$  is computed as the average of the subscribers' feedback on  $VS_i$ .

**Central manager** The central manager is responsible for storing the VSs and requests information, allocating services/composite services to requests, listening to events, and adapting the services allocations either reactively (after a violation occurs) or proactively (if a violation is expected to occur). The central manager is composed of the following parts: the frontend, the service repository, and the self-aware framework.

- (1) **The frontend.** The frontend provides interfaces for the volunteers' and subscribers' interactions through a web portal. It collects and manages the VSs' descriptions and the subscribers' requests. All the VSs' descriptions are stored in the service repository and all the requests are stored in the system queue.
- (2) **The Service Repository.** The service repository is simply a database that stores the VS's descriptions and the VS's status; either allocated or not.
- (3) **The self-aware framework.** In the presence of the challenges of dynamism and uncertainty, knowledge is essential to steer the adaptation decisions. The self-aware framework provides the knowledge and adaptation management for the VSS. It enables multi-level knowledge acquisition and representation, through different levels of awareness, at runtime and implements corresponding adaptation approaches.

The following sections zoom in on the structure of the self-aware framework then present the dynamic knowledge management approaches.

## 5.4 Architecture of the Self-aware Framework

As shown in chapter 2, there is a growing trend of adopting self-awareness as an enabler for self-adaptation in software systems. This resulted in different frameworks and approaches for engineering self-awareness, e.g. the representative projects under the EU

Proactive Initiative Self-Awareness in Autonomic Systems [138] [20] [139] [140] and road-mapping agenda of the Dagstuhl seminar [17]. Obviously, for such a hot research topic, there is no definitive solution for engineering self-awareness. Therefore, in this thesis, we are taking a ‘flavour’ of self-aware framework, the EPiCS framework [20], and exploit it to build our contribution of dynamic knowledge management. In this section, we overview the conceptual self-aware framework contributed by the EPiCS project then adapt and map the framework to the case of VC.

#### **5.4.1 Overview of the EPiCS Framework**

The EPiCS project has produced a conceptual framework related to how self-awareness can be used to engineer self-adaptive computing systems. We believe that this framework is suitable to build our contribution on for the following reasons:

- The framework adheres to the separation of concerns principle by defining multi-levels of self-awareness. In our context, this enables the separation of knowledge concerns and the development of multi-level adaptation capabilities using the different types of knowledge.
- Conceptually, the framework provides the primitives for adjusting and reasoning about the way in which self-awareness is realised. This allows for enriching the adaptation capabilities by switching between different approaches of knowledge management.

Figure 5.2 illustrates the architectural diagram of the framework which consists of four main components, namely, Sensors, Self-awareness, Self-expression, and Meta-self-awareness:

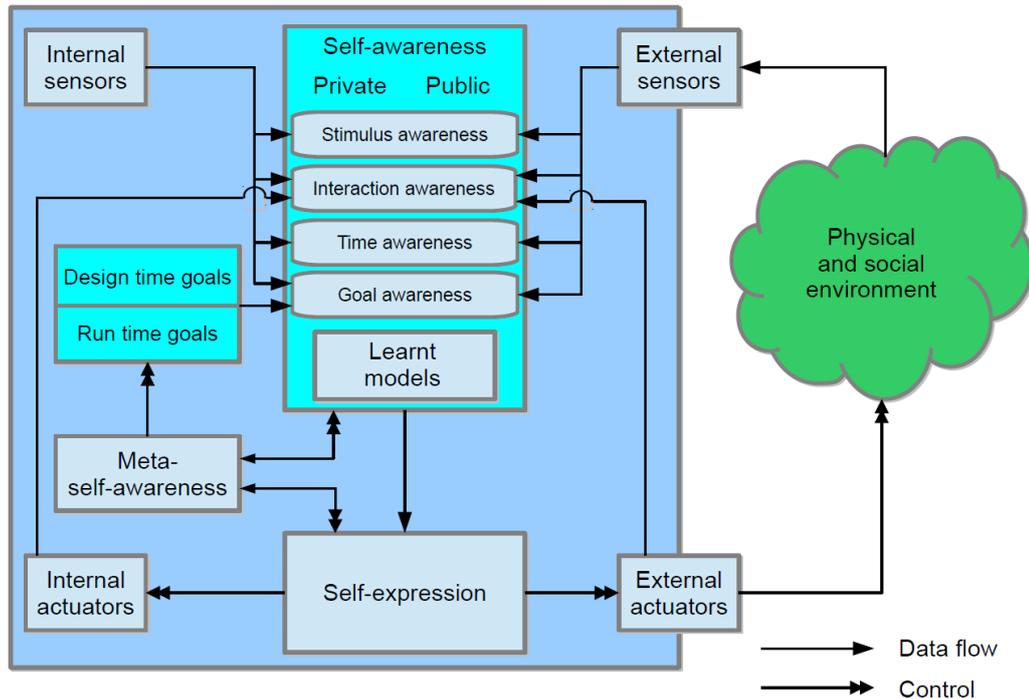


Figure 5.2: The Conceptual Self-Awareness Framework (Source [14])

- (1) Internal/external sensors:** The sensors are responsible for collecting data on the private experiences internal to the system and public experiences related to the system's physical environment.
- (2) Self-awareness:** The self-awareness component is responsible for modelling the received data into knowledge and passing the models to the self-expression and meta-self-awareness components as inputs. The component introduces four levels of awareness:
- Stimulus-awareness. This level provides the basic knowledge on the changes that occur internally and externally. It enables the system to react to internal and external events, thus, it is necessary for providing the basic level of self-adaptation.

- Time-awareness. This level is intended to provide knowledge about internal and/or external historical performance; assuming the presence of the stimulus-awareness.
- Interaction-awareness. This level also assumes the presence of the stimulus-awareness and is responsible on modelling the knowledge captured from the interactions that occur in the system.
- Goal-awareness. The aim of this level of awareness is to maintain the system's goals and objectives.

### **(3) Meta-self-awareness**

The system is meta-self-aware if it has knowledge about its current level of awareness along with the benefits and costs of that level. It should enable switching between the levels of awareness at runtime.

### **(4) Self-expression**

This component makes use of the learnt models passed by the self-awareness component and performs the actual adaptation decisions. Based on the activated level of awareness by the meta-self-awareness, the corresponding models are used to inform and execute the adaptation actions.

Specific patterns of this framework were applied in some demonstrators for the sake of motivating the need for self-awareness in computing systems and demonstrating the applicability of the framework. For example, in [20] a pattern that involves the stimulus- and interaction-awareness has been used in a smart-camera self-adaptive application. The framework enabled a smart camera to 'sell' an object it is tracking to another

camera when that object is about to leave the first camera field of view, which is an example of interaction-awareness. Another example, presented in [61], uses a pattern of time-awareness where ratings about cloud services are stored and used for the cloud services selection. However, these demonstrators make simple assumptions regarding the knowledge representation and the scale of the environment, as they intended only to show the applicability of the framework.

#### **5.4.2 The Self-aware Framework for the Volunteer Computing**

In Figure 5.3 we adapt the self-awareness framework and map it to the case of VC as the following:

##### **(1) Internal/external sensors.**

The internal sensors are responsible for monitoring the queue status. Events related to the arrival of requests to the queue along with the size of the queue are reported to the self-expression and the meta-self-awareness components. The external sensors are responsible for collecting data on the services engaged in a composition and the services offered in the service repository. The data include any changes in the promised quality of service. Then the collected data are passed to the stimulus-, time-, and interaction-aware levels in the self-awareness component.

##### **(2) Self-awareness.**

The self-awareness component is responsible for representing the acquired knowledge on the services performance and the users' goals. The component introduces four levels of awareness, which correspond to four levels of knowledge management. Each level produces the corresponding learnt models. Then the learnt models are passed to the

self-expression and meta-self-awareness components as inputs. The four levels of awareness are:

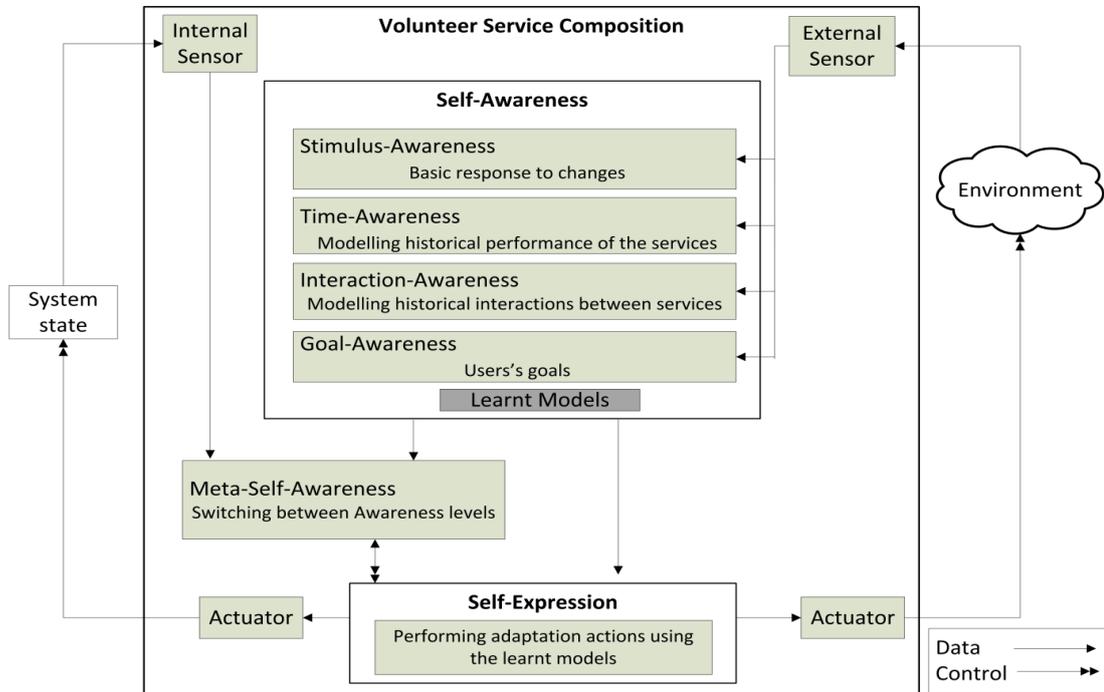


Figure 5.3: The Self-Awareness Framework for The Volunteer Computing Case

- Stimulus-awareness. This level provides the basic knowledge on the changes that occur in the performance of the services (e.g. a service violated the requirements). This knowledge supports the ability to adapt the CS. For example, if the change results in violating the constraints, then the corresponding service will be replaced.
- Time-awareness. This 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. The time-awareness is able to represent the evolving performance patterns of the VSs using the dynamic data structures, *the dynamic histograms*. This level enables the

system to take more intelligent adaptation decisions by selecting services which exhibit dependable historical performance.

- Interaction-awareness. This level also assumes the presence of the stimulus-awareness and adds more awareness by considering the historical interactions between the services in pairs in terms of dependability. In other words, this level is able to capture knowledge from the interactions between services so that services that had good performance when composed together in the past will have higher chance to be selected again together.
- Goal-awareness. The aim of this level of awareness is to maintain the users' goals. This level has the knowledge of the constraints of the CSs (see section 4.2.2). Given the services performance attributes and user requirements, goal-awareness deduces whether the VSs and the CSs satisfy the users' requests or not.

### **(3) Meta-self-awareness**

This component represents an extra level of awareness that acts as a cognitive system that controls the activation/deactivation of the above levels of awareness. The learnt models and information on the system state (in terms of the level of achieving the users' goals) are passed to this component to allow for deciding whether the activation of a certain awareness level is beneficial or not. The internal structure and the mechanism of the meta-self-awareness level are introduced in details in the next chapter.

### **(4) Self-expression**

This component performs the actual adaptation action based on the learnt models received from the self-awareness component. Based on the activated level of awareness

by the meta-self-awareness, the corresponding algorithm is used to find and adapt the composite services.

In the following sections, we propose the dynamic knowledge management approaches, which extend the framework and realise the above levels of awareness.

## 5.5 Dynamic Histograms for Dynamic Knowledge Management

A histogram is an estimate of the data distribution of a certain variable. Given a certain dataset, a histogram divides its data into subsets called buckets based on a partitioning rule [141]. The adoption of different partitioning rules results in different types of histograms. The most popular types of histograms are the Equi-width and the Equi-depth histograms. In the Equi-width histogram, the data range is divided into equal-width buckets, then the data points that have values between the minimum and the maximum of a certain bucket are grouped in that bucket. In the Equi-depth histogram, the bucket boundaries are specified so that the buckets have the same number of data points. Other types of histograms include: (i) the Compressed histogram in which data points that have highest frequencies are grouped in a single bucket, then the rest of the data points are grouped according to the Equi-width rule, (ii) the V-optimal histogram in which the quantity  $\sum n_i V_i$  is minimised, where  $n_i$  is the number of data points and  $V_i$  is the variance of the data points values in bucket  $i$ , and (iii) the V-Optimal-End-Biased histogram which is an enhancement to the V-optimal histogram and in which highest frequency data points and lowest frequency data points are grouped in individual buckets while the rest of the data points are grouped in a single bucket. The Equi-width histogram has been widely used in commercial systems as it is easy to implement when the minimum and maximum of each bucket are apriory known [142]. However, in this

type some of the buckets may include no or few points, i.e. they do not provide enough information. In such cases, the Equi-depth histograms may be a better choice for providing information on the data distribution, however, specifying the buckets boundaries can be expensive. The other types of histograms have been rarely used as they are very expensive to construct [142].

Meanwhile, in cases when data points arrive continuously and the dataset is built incrementally over time, the histogram needs to be recreated from scratch which results in degradation of system performance [23]. To deal with such cases, *dynamic histograms* have been proposed to capture and estimate the data distribution in evolving datasets. Dynamic histograms are constructs that dynamically approximate data distributions at runtime [23]. They have been used in database management systems' applications in order to maintain and represent the data which continuously arrive and vary with time [142]. Dynamic histograms are continuously updated to tackle the changes in the evolving datasets. The main idea in the dynamic histograms is to reconstruct the buckets, which involves splitting and/or merging buckets, at run time based on the partitioning rule of the histogram in order to keep the properties of the histogram.

The dynamic knowledge management approaches build on the findings of the long-term studies [5] [111] [112] [113], which reported the presence of periodic performance and correlated patterns of the volunteer host. As mentioned in chapter 3, since different volunteers contribute to different projects, the data collected from one VC project cannot be used to predict the actual performance patterns of the volunteers contributing to a different project. Henceforth, knowledge needs to be captured, represented, and managed at runtime in a way that enables capturing the performance

patterns and the correlated patterns. For this purpose, we use dynamic histograms. Our self-aware approaches (presented below), divide the services' usage time into time intervals dynamically at runtime. The time intervals correspond to the dynamic histograms buckets. Then the captured knowledge on the performance of the services is stored in the buckets as data points. The insertion and deletion of the data points may result in splitting and/or merging buckets based on the sufficiency of the number of data points in those buckets. The sufficient number of data points is specified using a method based on Chebyshev's inequality. In the next sections, we briefly introduce Chebyshev's inequality and present the dynamic histogram evolution operations we have developed.

### 5.5.1 Chebyshev's Inequality

Suppose that we have a set of  $N$  data points for a random variable (e.g. observations of a service's performance) but the distribution of the random variable is unknown. The expected value can be estimated using the data points. Then the Chebyshev's inequality can be used in order to know how close the estimated expected value is to the actual one. In other words, Chebyshev's inequality bounds the probability that a random variable deviates from its expected value by a sufficiently small positive number  $\varepsilon$ , called confidence threshold [143]. Mathematically, Chebyshev's inequality is expressed as:

$$P(|E(X) - \hat{E}(X)| \geq \varepsilon) \leq \frac{\sigma^2}{N \cdot \varepsilon^2} \quad (5.1)$$

where  $E(X)$  is the actual expected value,  $\hat{E}(X)$  is the estimated expected value,  $\sigma$  is the standard deviation,  $N$  is the number of data points, and  $\varepsilon$  is the confidence threshold.

In our approach, we use Chebyshev's inequality in a different way. Our purpose is to know when the number of data points,  $N$ , in a bucket of the dynamic histogram will be sufficient to give a close estimate of the expected value, which helps to decide when to

split the bucket and evolve the histogram. So, given the confidence threshold  $\varepsilon$  and the probability of confidence  $P(|E(X) - \hat{E}(X)| \geq \varepsilon)$  and solving (5.1) then the number of sufficient data points is calculated as in (5.2). The corresponding method is presented in the next section.

$$N = \frac{\sigma^2}{P(|E(X) - \hat{E}(X)| \geq \varepsilon) \cdot \varepsilon^2} \quad (5.2)$$

### 5.5.2 Evolution Operations

As mentioned, the system starts from 'zero-history' and then the knowledge is captured and managed incrementally at runtime using the dynamic histograms. We adopt a dynamic histogram for each service in order to continuously insert the observed data points taking into account the time interval in which the data point has been observed. Then the continuous update of the dynamic histogram, by splitting and/or merging the buckets, results in refining the histogram structure and capturing the periodic performance pattern of the services. Accordingly, a data point is defined as follows:

**Definition 5.1 (Data point)** A data point is a tuple of  $(T = [a, b], value)$  where  $T$  is the time interval in which the observation has been recorded,  $a$  is the start date (the date and time at which the services is involved in a CS) of  $T$  and  $b$  is the end date (the date and time at which the service violated the promised utility or completed serving the request), and  $value$  is the value of the performance metric (i.e. dependability).

The update process of the dynamic histogram involves inserting a new data point into the appropriate bucket(s), splitting a bucket when the number of data points is sufficient to estimate the performance, and merging each empty bucket with a

neighbour one. In the following, we describe each of the mentioned operations and show the corresponding algorithm.

**(1) Insert new data point.**

Based on Definition 5.1, a data point might fall into one or more buckets depending on the intersection between the data point time interval and the bucket(s) boundaries. Algorithm 5.1 is used to find the appropriate bucket(s) in which the data point will be inserted.

---

Algorithm 5.1: Find Appropriate Buckets

---

```

1   Input: Dynamic Histogram dhist, Data Point dp
2   Output: Array appropriateBuckets
3   Begin
4   |   for all bucket in dhist do
5   |       // check if the time intervals of dp and bucket intersect
6   |       if dp.start_date < bucket.end_date && dp.end_date > bucket.start_date then
7   |           add bucket to appropriateBuckets
8   |       end if
9   |   end for
10  |   return appropriateBuckets
11  End

```

---

**(2) Split a bucket.**

A bucket in the dynamic histogram corresponds to a time interval in which the performance of a service has been observed. The periodic performance of the services means that a service exhibits different behaviour in different time intervals. Therefore, the buckets boundaries need to be dynamically updated as the data points are inserted in order to track the changes in the performance and coincide with the time intervals in which the service has been used and observed. When the number of data points in a bucket is sufficient to estimate the performance of the service in the corresponding time interval, the bucket will be split into smaller buckets. The splitting provides finer

representation of the time intervals in order to capture the pattern periods. The sufficient number of data points in a bucket is determined by solving (5.2). We can bound the variance  $\sigma^2$ . Assuming the worst case; the variance is maximum when one half of the values is at lowest possible and the other half is at the highest possible value. In this work we express the performance in terms of dependability, (defined in the next section), which has a value in  $[0.0, 1.0]$ . Based on that, the lowest value of the performance is 0.0 and the highest is 1.0. As a result, the maximum variance is 0.25 and the splitting threshold  $split\_th$  is given by:

$$split\_th = \frac{0.25}{P(|E(X) - \hat{E}(X)| \geq \varepsilon) \cdot \varepsilon^2} \quad (5.3)$$

Consequently, when the number of data points in a bucket exceeds  $split\_th$ , the bucket will be split into two equal-length buckets using Algorithm 5.2. We set a minimum length of the bucket (time interval). If the bucket length is less than the minimum length, the bucket cannot be split. In this case a *forget* strategy is applied to remove the oldest point(s) and allow the new data point(s) to be inserted.

---

Algorithm 5.2: Split Bucket

---

```

1  Input: Bucket bucket
2  Output: Bucket bucket1, Bucket bucket2
3  Begin
4  |   Calculate splitting_date = (bucket.start_date + bucket.end_date) / 2
5  |   Create Bucket bucket1 such that bucket1.start_date = bucket.start_date and
6  |   bucket1.end_date = splitting_date
7  |   Create Bucket bucket2 such that bucket2.start_date = splitting_date and
8  |   bucket2.end_date = bucket.end_date
9  |   for all data point dp in temp_array do
10 |       if dp.time_interval intersects with bucket1.time_interval
11 |           Insert dp into bucket1
12 |       end if
13 |       if dp.time_interval intersects with bucket2.time_interval
14 |           Insert dp into bucket2
15 |       end if
16 |   end for
17 |   Delete bucket
18 |   return bucket1 and bucket2
19 End

```

---

### (3) Merge empty buckets

If the splitting operation resulted in an empty bucket, then that bucket will be merged with its preceding neighbour. If the empty bucket does not have a preceding neighbour, it will be merged into the following one.

Pseudo-code for the update method of the dynamic histogram is presented in Algorithm 5.3.

The self-aware approaches we propose in the next section use the dynamic histograms to model the captured knowledge on the performance of the services.

---

#### Algorithm 5.3: Dynamic Histograms Update

---

```
1  Input: Dynamic Histogram  $dhist$ , Data Point  $dp$ 
2  Output: Updated version of  $dhist$ 
3  Begin
4  |    $appropriateBuckets = FindAppropriateBuckets(dp, dhist)$ 
5  |   for all Bucket  $bucket_i \in appropriateBuckets$  do
6  |   |   insert  $dp$  in  $bucket_i$ 
7  |   |   if  $bucket_i.size \geq split\_th$  then
8  |   |   |    $Bucket[] temp\_array \leftarrow SplitBucket(bucket_i)$ 
9  |   |   |    $bucket_1 \leftarrow temp\_array[0]; bucket_2 \leftarrow temp\_array[1]$ 
10 |   |   |   Replace  $bucket_i$  by  $bucket_1$  and  $bucket_2$ 
11 |   |   |   Set the successor and predecessor buckets for  $bucket_1$  and  $bucket_2$ 
12 |   |   end if
13 |   end for
14 |   for all Bucket  $bucket_i$  in  $dhist$  do
15 |   |   if  $bucket_i$  is empty then
16 |   |   |   Merge  $bucket_i$  with its successor or predecessor
17 |   |   end if
18 |   end for
19 |   return  $dhist$ 
20 End
```

---

## 5.6 Self-aware Selection and Adaptation Levels

### 5.6.1 Stimulus-aware Selection and Adaptation

The selection of the VSs in this approach is based on the promised utilities of the volunteers. When a subscriber submits a request, the system computes the utilities

using the utility model and applies the utility-based search (see in Algorithm 4.2) to find a composite service that satisfies the request.

With regards to self-adaptivity, the stimulus-aware adaptation is considered as the basic level of adaptation, as it is the adaptation approach supported in the current volunteer computing systems. The adaptation actions are limited to replacing the violating service by another one in order to maintain the corresponding composite service. To clarify, 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  $VS_i$  by re-executing the utility-based search. If the adaptation process is successful, then the violating service is replaced, otherwise, the subscriber is notified that the violation cannot be treated.

### **5.6.2 Time-aware Selection and Adaptation**

The aim of the 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 performance of the services in terms of *dependability*. We consider a service  $VS_i$  to be dependable if  $VS_i$  provides the promised storage and security level in the promised time availability. In this section, we introduce the definition of dependability then the time-aware VS selection 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 *dependability* in a broad sense to measure 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 attribute, there will be a

dependability measure for each service attribute. We introduce the definition of dependability as follows:

**Definition 5.2** Given that a volunteer service  $VS_i$  has been selected in a composite service  $CS$  to serve the request  $R$ . Assume that  $U_{stg}^P(VS_i)$  is the storage utility promised by the volunteer of  $VS_i$ . Assume also that the actual storage utility provided by  $VS_i$ , captured by the self-aware framework sensors during serving  $R$  is  $U_{stg}^A(VS_i)$ . Then the storage dependability of  $VS_i$ ,  $D_{stg}(VS_i)$ , is defined as in (5.4). The availability time dependability,  $D_{time}(VS_i)$ , is defined similarly as in (5.5). The security dependability is calculated as a weighted sum that involves the reported level of security (provided by the subscriber frontend) along with the reputation of the service (provided as a feedback by the subscriber) as shown in (5.6).

$$D_{stg}(VS_i) = \begin{cases} \frac{U_{stg}^P(VS_i) - U_{stg}^A(VS_i)}{U_{stg}^P(VS_i)}, & U_{stg}^A(VS_i) < U_{stg}^P(VS_i) \\ 1, & \text{Otherwise} \end{cases} \quad (5.4)$$

$$D_{time}(VS_i) = \begin{cases} \frac{U_{time}^P(VS_i) - U_{time}^A(VS_i)}{U_{time}^P(VS_i)}, & U_{time}^A(VS_i) < U_{time}^P(VS_i) \\ 1, & \text{Otherwise} \end{cases} \quad (5.5)$$

$$D_{sec}(VS_i) = \begin{cases} W_1 \frac{U_{sec}^P(VS_i) - U_{sec}^A(VS_i)}{U_{sec}^P(VS_i)} + W_2 Rep(VS_i), & U_{sec}^A(VS_i) < U_{sec}^P(VS_i) \\ W_1 + W_2 \cdot Rep(VS_i), & \text{Otherwise} \end{cases} \quad (5.6)$$

where  $W_1 + W_2 = 1$

## (2) Knowledge management using dynamic histograms

Our aim is to capture the periodic performance patterns of the VSs, in terms of dependabilities, so that the system can use such 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, a dynamic histogram is created for each service attribute. Initially, each dynamic histogram contains one bucket, then the dynamic histogram evolves by dividing/merging buckets as the dependabilities' data points arrive. For each service, a new data point will arrive in two cases, (i) a service violates the promised utilities or (ii) a request, in which the service is involved to satisfy, has been satisfied. In both cases, the dependabilities will be computed using (5.4), (5.5) and (5.6) and inserted into the appropriate bucket(s) using Algorithm 5.3. After a certain period of time, the dynamic histogram converges to a state in which the buckets represent the service's pattern periods. The length of the convergence period depends on how often the service is used.

### **(3) Time-aware service selection**

When a subscriber submits a request, the following key steps are executed in order to satisfy the request:

*Step 1:* For each  $VS_i \in SR$ , compute the  $U_{stg}$ ,  $U_{time}$ , and  $U_{sec}$  using the utility functions (4.2), (4.3), and (4.4) respectively.

*Step 2:* For each  $VS_i \in SR$  find the appropriate buckets from the corresponding dynamic histogram. Each bucket overlaps with request interval is considered an appropriate bucket (see Algorithm 5.1).

*Step 3:* For each bucket, estimate the representative  $D_{stg}$ ,  $D_{time}$ , and  $D_{sec}$  for each  $VS_i \in SR$  by counting the number of data points which have a value greater than or equal to the dependability threshold  $D_{th}$ , (which is set by the system administrator), and dividing that number by the total number of data points in the bucket.

*Step 4:* Find the average storage dependability,  $AVD_{stg}$ , for each  $VS_i$  by summing the representative storage dependability of each bucket and dividing over the number of buckets. Similarly find  $AVD_{time}$  and  $AVD_{sec}$ .

*Step 5:* Find the non-dominant set of services using (5.7), select one of them randomly, and add it to  $CS$ .

$$\begin{aligned} & \text{maximize} \{U_{stg}(VS_i), U_{time}(VS_i), U_{sec}(VS_i) \\ & \quad AVD_{stg}, AVD_{time}, AVD_{sec}\} \end{aligned} \quad (5.7)$$

After executing the above steps, the subscriber request will be partially satisfied, then the request requirements will be recalculated in order to update the remaining requirements, and the above steps will be repeated to select the next service. After selecting each service, the constraints of the  $CS$  (see section 4.2.2) 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  $CS$  will be returned and the subscriber will be notified that the request cannot be satisfied.

#### **(4) Time-aware adaptation**

The self-adaptivity in the time-aware approach is two-fold, in terms of the question:

*“When should we adapt?”*

- **Reactive adaptation:** When a change in the promised quality of service is reported to the time-awareness component, the actual utilities will be computed using (4.2), (4.3), and (4.4) and subsequently the dependabilities using (5.4), (5.5) and (5.6). Then the dependabilities will be stored in the corresponding dynamic histogram. After that, an adaptation action will be carried out by the

self-expression component. This adaptation action involves executing the above time-aware service selection steps in order to replace the service that violated the requirements.

- **Proactive adaptation:** The system performs proactive adaptation in order to adapt a composite service before a violation occurs. The proactive adaptation is triggered in two cases, (1) the dependability of a service involved in a CS is expected to drop, according to the performance pattern captured in the service dynamic histogram, or (2) a service has become available in the SR which is expected to perform better than an existing one, according to its performance patterns. In both cases, the system will execute the above time-aware service selection steps in order to adapt the CS.

### 5.6.3 Interaction-aware Selection and Adaptation

The aim of the interaction-aware approach is to consider the past interactions between the services in order to capture the correlation that exists between services. In other words, the interaction-aware approach aims to predict which services are most appropriate to be composed together to satisfy a request. To achieve that, we maintain a matrix of dynamic histograms for the services for each attribute and then we use the same machinery we described in the time-aware approach for updating the dynamic histograms, estimating the dependability, and adapting the composite services. The dynamic histograms matrix  $DHM_{stg}$  for the storage attribute has the form:

$$DHM_{stg} = \begin{matrix} & VS_1 & VS_2 & \dots & VS_n \\ \begin{matrix} VS_1 \\ VS_2 \\ \vdots \\ VS_n \end{matrix} & \begin{pmatrix} - & dh_{12} & \dots & dh_{1n} \\ dh_{21} & - & \dots & dh_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ dh_{n1} & dh_{n2} & \dots & - \end{pmatrix} \end{matrix}$$

where  $VS_1, VS_2, \dots, VS_n$  are the available services,  $dh_{ij}$  is the dynamic histogram that maintains the dependabilities of  $VS_i$  when composed with  $VS_j$ . Similarly, two matrices are maintained for the time availability ( $DHM_{time}$ ) and security ( $DHM_{sec}$ ) attributes. Then, when a request is submitted, the dynamic histograms will be used to estimate the interaction dependabilities between services. The key steps for the interaction-aware service selection are as follows:

*Step 1:* For each  $VS_i \in SR$ , compute the  $U_{stg}$ ,  $U_{time}$ , and  $U_{sec}$  using the utility functions (4.2), (4.3), and (4.4) respectively.

*Step 2:* Find the non-dominant set of services using (4.5), select one of them randomly,  $VS_i$ , and add it to  $CS$ .

*Step 3:* If  $VS_i$  satisfies the request, then the composite service  $CS$  will be returned, otherwise, execute the following steps.

*Step 4:* For each  $VS_j \in SR, j \neq i$ , retrieve the dynamic histograms from the matrices  $DHM_{stg}$ ,  $DHM_{time}$ , and  $DHM_{sec}$ .

*Step 5:* Find the appropriate buckets from the corresponding dynamic histograms. Each bucket overlaps with request interval is considered an appropriate bucket (see Algorithm 5.1).

*Step 6:* For each bucket, estimate the  $D_{stg\_interaction}$ ,  $D_{time\_interaction}$ , and  $D_{sec\_interaction}$  for each  $VS_i \in SR$  by counting the number of data points which have a value greater than or equal to the dependability threshold  $D_{th}$  and dividing that number by the total number of data points in the bucket.

*Step 7:* Find the average storage dependability,  $AVD_{stg\_interaction}$ , for each  $VS_j$  by summing the representative storage dependability of each time slot and dividing over

the number of time slots. Similarly find  $AVD_{time\_interaction}$  and  $AVD_{sec\_interaction}$ .

*Step 8:* Find the non-dominant set of services using (5.8), select one of them randomly, and add it to  $CS$ .

$$\begin{aligned} & \text{maximize} \{U_{stg}(VS_i), \quad U_{time}(VS_i), \quad U_{sec}(VS_i) \\ & \quad ID_{stg}, ID_{time}, ID_{sec}\} \end{aligned} \quad (5.8)$$

Similar to the time-aware case, after executing the above steps, the subscriber request will be partially satisfied, then the request requirements will be recalculated in order to update the remaining requirements, and the above steps will be repeated.

## 5.7 Experimental Evaluation

In this section, we conduct experiments in order to evaluate the performance of the stimulus-, time- and interaction-aware approaches using simulations. The experimentations setup and context is as described in the previous chapter. Table 5.1 lists the values of the required parameters. With respect to the performance of the services, it is assumed to have a periodical daily or weekly patterns, according to the findings of the long-term studies reported in [5] and [111] as presented in chapter 3.

Table 5.1: The Values of the Parameters

Parameter	Value
Confidence threshold ( $\epsilon$ )	0.18
Confidence probability $P( E(X) - \hat{E}(X)  \geq \epsilon)$	0.9
Minimum interval length ( $minLength$ )	1 day
Dependability threshold ( $D_{th}$ )	0.8
$W_1$	0.5
$W_2$	0.5

The objective of this evaluation is to investigate the effectiveness of the self-aware dynamic knowledge management, represented in the time- and interaction-awareness in

comparison with the basic self-adaptive approaches, represented by the stimulus-awareness. The approaches are compared in terms of the RW, WT, and PSR, which are defined in the previous chapter. With regard to the RW, we adapt the definition to include the resources of the services that are replaced due to violating the promised utilities.

**Comparison in Resources Waste (RW)** The first set of experiments compares the average resources waste over simulation time. Figure 5.4(a), (b), (c), and (d) shows the average RW for varying arrival rates  $\lambda$ . The figure shows that the RW in the stimulus-awareness case is high. A possible explanation for these results is that the ineffectiveness of the adaptation capabilities at the stimulus-awareness level. That is, replacing the VSs that violate the promised utilities with other VSs based on their promised utilities does not provide any guarantee that the new VS will perform as promised. That means the new VS can violate the promise again. The figure also shows that the RW in the time- and interaction-awareness cases is high in the initial interval of the simulation time, especially in the case of low requests arrival rate. These results can be due to the lack of the knowledge at the time- and interaction-awareness levels as the system starts from 'zero-knowledge' and accumulate the knowledge at runtime. After a while of accumulating the knowledge, the RW in the time-aware case decreases and after another while the RW in the interaction-aware decreases further. Moreover, the figure shows that the initial period of ineffectiveness in the time- and interaction-awareness cases decreases with the increase of the requests arrival rate. A possible explanation for this is that the more requests arrive, the more services are used and the hence the more knowledge is captured. In conclusion, the time- and the interaction-aware approaches have the advantage of reducing the RW, which can be due to the

selection of the dependable services when the required knowledge about the services dependability becomes adequate.

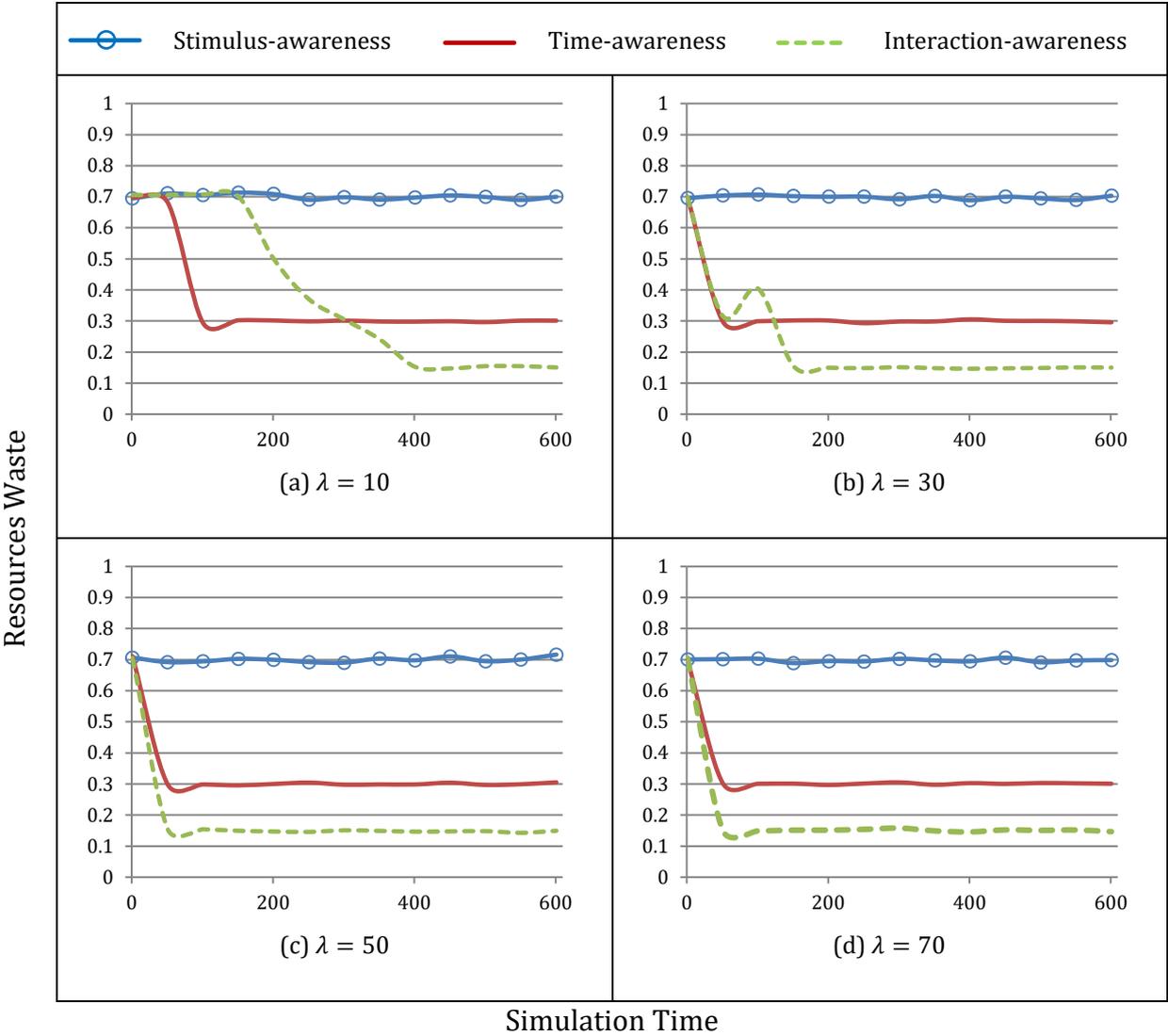


Figure 5.4: Comparison in Resources Waste

**Comparing the Percentage of satisfied requests (PSR)** The second set of experiments compares the average PSR over simulation time. Figure 5.5 (a), (b), (c), and (d) shows the average PSR for varying arrival rates  $\lambda$ . The figure shows that the PSR in the stimulus-awareness case is low. This can be due to the ineffectiveness of the

adaptation capabilities at the stimulus-awareness level by selecting the VSs based on their promised utilities. The figure also shows that the PSR in the time- and interaction-awareness cases is low in the initial interval of the simulation time, which can be due to the lack of the knowledge in this period. After a period of accumulating the knowledge, the PSR in the time-aware case increases and after another period the PSR in the interaction-aware increases further.

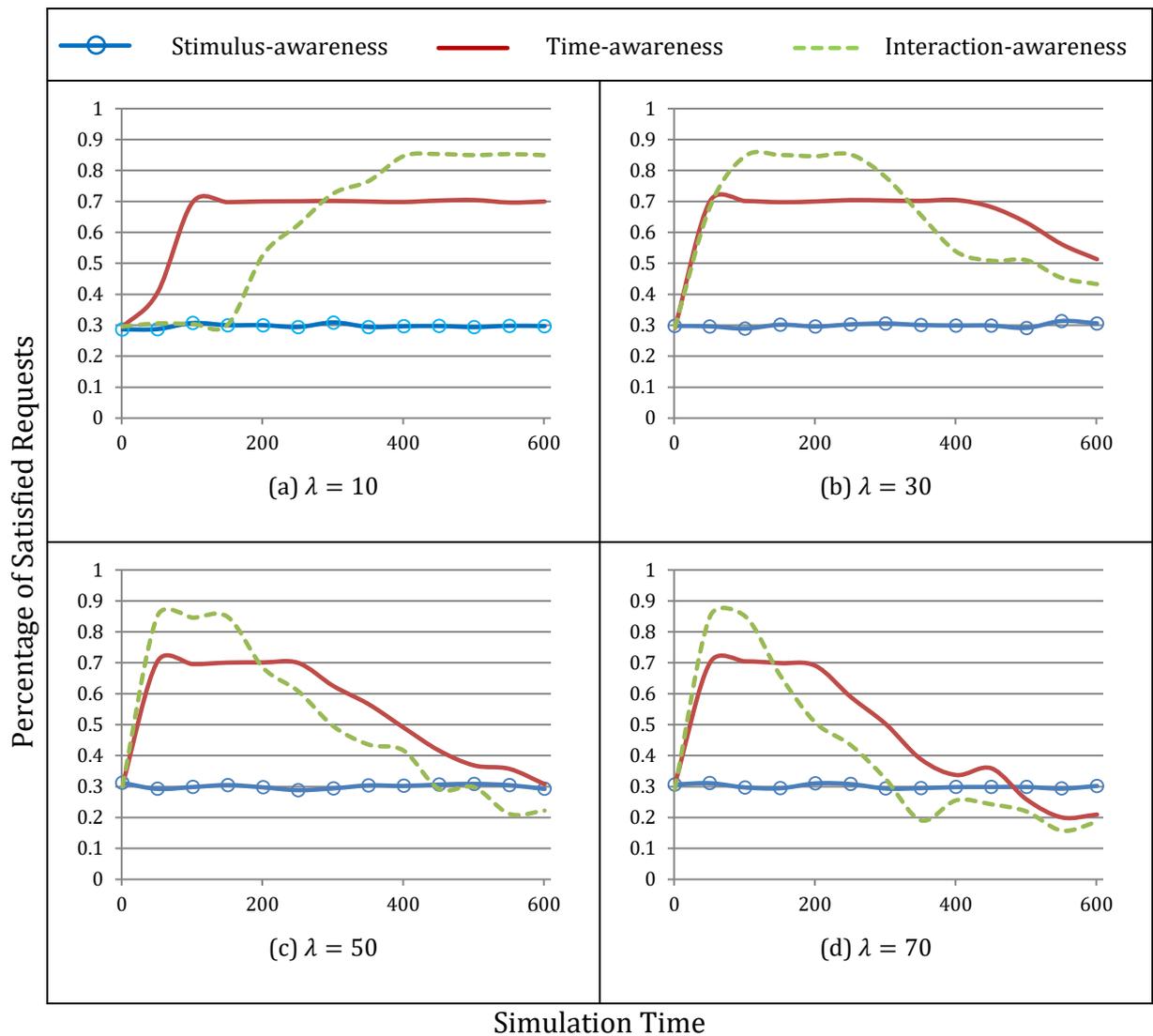


Figure 5.5: Comparison in Percentage of Satisfied Requests

Additionally, the figure shows that the initial period of ineffectiveness in the time- and interaction-awareness cases decreases with the increase of the requests arrival rate. Again, the reason can be that the more requests arrive, the more services are used and hence the more knowledge is captured. However, the PSR decreases in the cases of high arrival rates, especially after a long period of simulation time. It is possible that this result is due to the dropping of the requests from the system queue, especially with the increase of the knowledge size after the long period of simulation time. In general, having the advantage of selecting the dependable services using the time- and the interaction-aware approaches results in increasing the PSR.

**Comparing the Waiting time (WT)** The third set of experiments evaluates the WT of the three approaches. Turning to the simulation results, Figure 5.6 (a), (b), (c) and (d) shows the WT for varying arrival rates  $\lambda$ . The figure shows that the use of the stimulus-aware approach provides the least waiting time. The reason is that using the stimulus-awareness approach does not require the computation of accumulating and using the knowledge for the services selection. In other words, the adaptation using the stimulus-awareness involves executing the utility-based search to select the VSs. On the other hand, the figure shows that the WT in the time- and interaction-awareness cases increases over simulation time. This can be due to the computation required to calculate the dependabilities of the services, which are maintained in the dynamic histograms. The increase of the knowledge size over time also contributes to the increase of the WT. Also, the WT increases further with the increase of the requests arrival rates as more processing will be required.

It is notable also that the WT in the time- and the interaction-awareness approach increases linearly over time, whereas the WT in the stimulus-aware approach is not affected. However, in the case of interaction-awareness, the waiting time can be expected to increase fast when the dynamic histograms matrix grows massively. A possible way to mitigate this situation is to limit the number of VSs that can be involved in a CS if the interaction-awareness is to be used. Furthermore, the meta-self-awareness level (presented in the next chapter) provides an approach for switching between the awareness levels, which enables avoiding the use of a certain awareness level if the WT is not acceptable compared to the usefulness of this level.

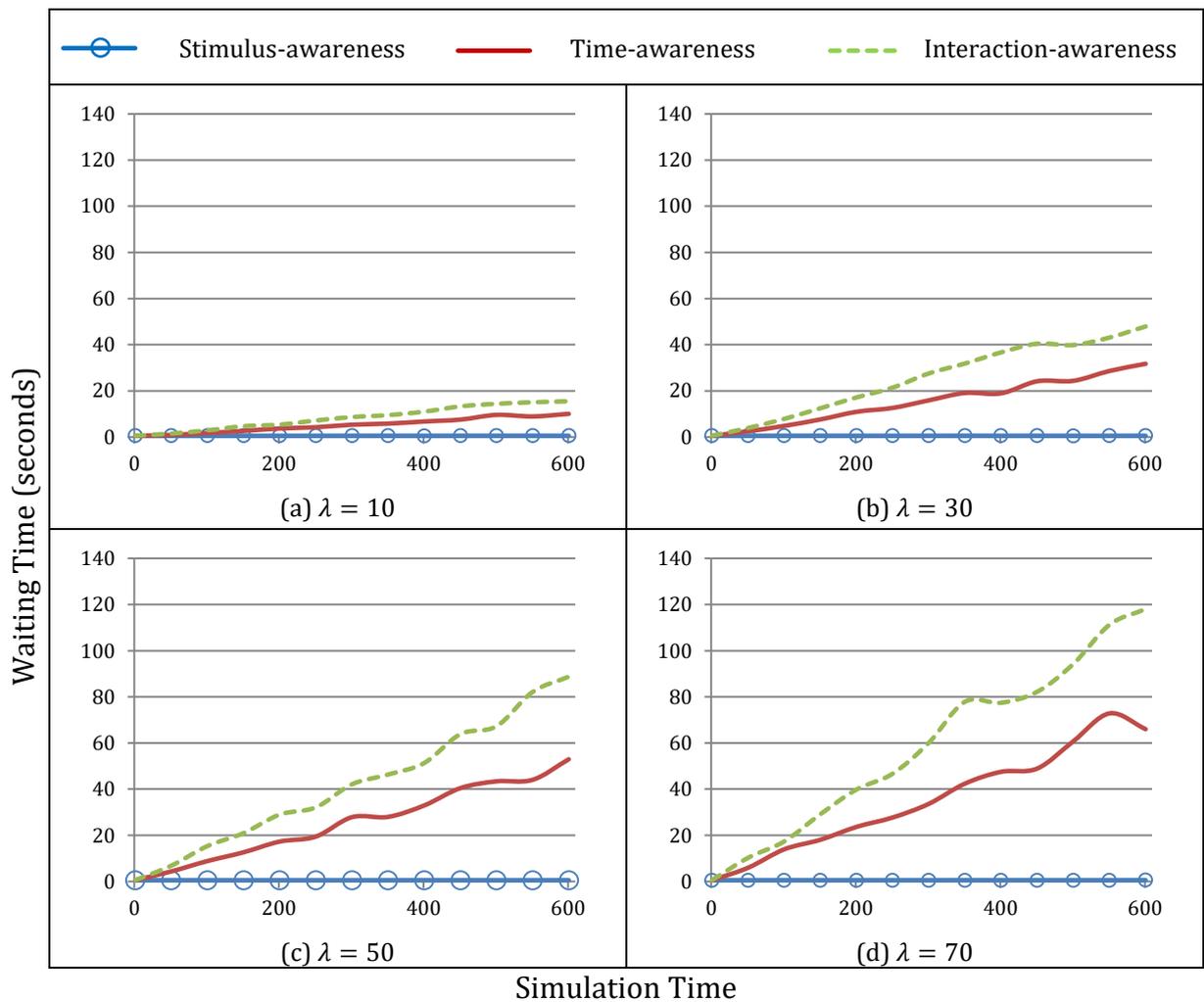


Figure 5.6: Comparison in Waiting Time

## 5.8 Conclusion

This chapter presented our attempt for enacting self-awareness in the dynamic and open environment of the VC. We provided an approach for dynamic knowledge management that is able to dynamically build and maintain the awareness about the performance patterns of the services. The approach is two-fold:

- A ‘flavour’ of self-awareness has been used to incorporate self-awareness in VC in order to enrich the self-adaptation capabilities in this environment. We used the EPiCS [20] self-aware framework as it adheres to the principle of separation of concerns by providing the fundamentals for multi-level knowledge representation, which make it appropriate for the case of VC. We adapted and mapped the framework to the case of VC.
- We proposed fundamental improvements to the framework by developing concrete algorithms for multi-level knowledge management, specifically, the stimulus-, the time-, and the interaction-awareness levels. The approaches use the dynamic histograms to capture the performance trends of the VSs. We developed algorithms for managing the evolution of the dynamic histograms and incorporating them in the dynamic knowledge management approaches that realise the self-awareness levels.

Meanwhile, even though the extension of the self-aware framework for the way of knowledge management has benefited the case of VC it is likely that the benefit will extend beyond the VC area. Applications which exhibit characteristics similar to the VC and would require more sophisticated handling for the knowledge could take advantage of the work of this thesis.

The chapter evaluates and compares the performance of the self-awareness levels (i.e. the dynamic knowledge management approaches). The results show that using the dynamic histograms for dynamic knowledge management helps to refine the performance models at runtime. As the above figures show, the advantages of selecting dependable services and satisfying more requests are noticed after the knowledge in the dynamic histograms is refined. However, the improvements are accompanied with overhead which is mainly the increase in the waiting time. This increase results from the increase of the computation required for updating the dynamic histogram. In the next chapter, we introduce the meta-self-awareness, to provide the self-adaptation capability at a meta-level by switching between the awareness levels based on the benefits and overhead of each of them.



# CHAPTER 6

## SYMBIOTIC-BASED META-SELF-AWARENESS FOR SELF-AWARE SYSTEMS

### 6.1 Overview

In the previous chapter, we have proposed approaches for dynamic knowledge management which contribute to engineering self-awareness by extending the conceptual architectural framework of EPiCS. The approaches realise various levels of self-awareness to enable different self-adaptive capabilities. Each level is enacted through a different type of knowledge and knowledge management mechanisms, providing different self-adaptive strengths. Besides that, the decomposition of knowledge into 'fine grain' and the utilisation of the different types of knowledge in different selection and adaptation approaches can result in different selection and adaptation strategies. For example, referring to Figure 4.1, the reason behind the existence of different strategies can be the use of different self-aware approaches to select the services and, obviously, the more intelligent the approach, the more appropriate services are selected. Nevertheless, the advantages are often accompanied

by overheads. This is exemplified in the relative performance of the e.g. interaction-awareness and stimulus-awareness where the adoption of interaction-awareness results in improving the percentage of satisfied requests and resources utilisation but higher waiting time, compared to the stimulus-awareness.

Motivated by the above, we argue that an effective management for the trade-offs of dependability requirements can be achieved through seamless switching between awareness levels as it is the case of cognition. Although such argument has been mentioned in the related self-awareness framework of EPiCS, an approach for the dynamic switching between the awareness levels, based on the benefits and overheads of each level, has not been tackled yet. In this chapter, we address this problem by proposing a novel approach, referred to as *meta-self-awareness* to address this problem. This level is intended to serve as a 'cognitive system' that can dynamically switch between the different levels of awareness based on the current state of the system and the quality of the knowledge acquired at each of the awareness levels.

In this context, we propose an internal structure of the meta-self-awareness level which consists of two main parts, namely, the *symbiotic simulator* and the *Decision maker*. The symbiotic simulator evaluates the alternative decisions that could have been taken and suggests an awareness level. The *Decision maker* evaluates the current workload of the system along with the suggestion of the awareness level and decides on adopting the suggested level or not. Consequently, the work mechanics of the meta-self-awareness level is two-fold:

- The meta-self-awareness approach makes novel use of a symbiotic simulation – in the heart of the self-adaptive process - to provide the basis for *what-if* analysis

in cases where the knowledge for adaptation is stringent. The symbiotic simulator performs what-if experiments to simulate the alternative VSs selection and adaptation decisions which could have been taken by the alternative levels of awareness. The simulation evaluates the performance of the different awareness levels in terms of percentage of satisfied requests, resources waste, and waiting time. In order to judge the awareness level using these conflicting criteria, a Multi-Criteria Decision Making (MCDM) technique is used to suggest a level of awareness.

- The *Decision maker* component of the meta-self-awareness takes into consideration the stability of the system's queue, in terms of avoiding the rapid growth of the queue size and the consequent high waiting times and requests dropping. For example, the meta-self-awareness will avoid switching to the interaction-awareness, which exhibits relatively large waiting times, if the system is receiving more requests.

Given the above background, the contributions of this chapter are as follows:

- Architecture for the meta-self-awareness level which makes novel use of a symbiotic simulation – in the heart of the self-adaptive process – to simulate and evaluate alternative selection and adaptation decision.
- A quantitative approach for self-adaptation at the meta-level which enables managing the trade-off between benefits and overheads of adopting a certain level of awareness taking into account the stability of the system's queue.

- An evaluation of the performance of the meta-self-awareness approach to investigate its effectiveness in switching between the other levels of self-awareness.

## 6.2 Symbiotic Simulations: A background

The notion of symbiotic simulation systems was introduced to the computing paradigm at the Dagstuhl Seminar on Grand Challenges for Modelling and Simulation in 2002 [144]. This notion was inspired by the biology where some kinds of interaction between some biological species are referred to as symbiosis. The Dagstuhl Seminar defined A Symbiotic Simulation System as *“one that interacts with the physical system in a mutually beneficial way”*. The definition implies that both the physical system and the symbiotic simulation system benefit each other. The simulation system receives real data collected by the physical system. This data is necessary to simulate 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 the alternatives. The physical part uses the results of the simulation to inform the decision-making process and optimise its performance. Figure 6.1 shows the relationship between the physical system and the symbiotic system according to the definition of the Dagstuhl Seminar [144].

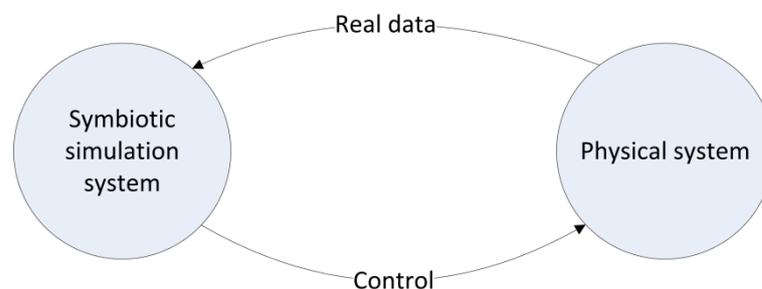


Figure 6.1: The Mutually Beneficial Symbiotic Simulation Paradigm. Adapted from [145]

In [146] this definition has been extended to involve other types of relationships between the two systems, which were also inspired by the symbiosis concept in biology. The extension introduced other types of symbiotic simulation systems which may or may not be mutually beneficial. In these types the symbiotic simulation receives real data from the physical systems may send feedback to the physical system which may effect this feedback or use it as guidance. The purpose of using the symbiotic simulation paradigm specifies the type of the symbiotic simulation will be used. In this context, two sorts of symbiotic simulations exist, namely, closed loop and open loop systems. In the closed loop system, a feedback will be created, analysed, and utilised to make a decision to control the physical system. Such feedback can be in the form of a decision that will be applied directly to the physical system or it can be only a suggestion that may or may not be applied. On the contrary, no feedback will be communicated to the physical system in the case of an open-loop system, where the output can be used by external operator or tools for specific purposes, e.g. visualisation. Figure 6.2 shows the relationship between the physical system and the symbiotic system in the closed loop and the open loop cases.

Based on that, five types of symbiotic simulations have been reported in [147] [146]:

- **Symbiotic simulation decision support system (SSDSS):** This type is a closed-loop system. The results of the symbiotic simulation will be communicated to an external *Decision maker* component as a suggestion rather than a decision. The suggestion will be used in the decision-making process by the *Decision maker* component, which will implement the decision in the physical system.

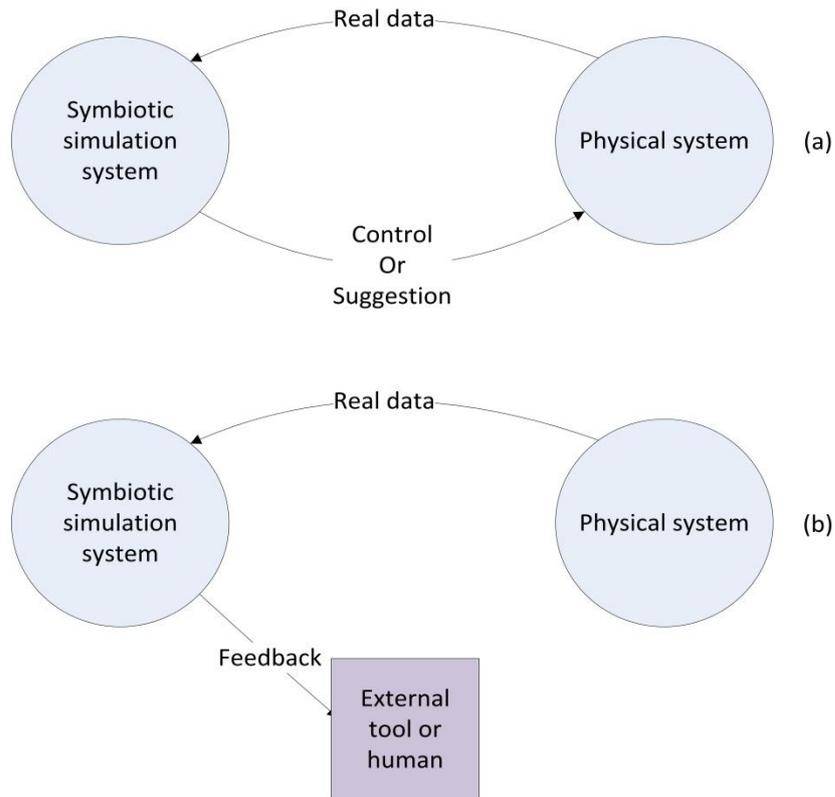


Figure 6.2: Symbiotic Simulation Paradigm (a) Closed-Loop (b) Open-Loop

- **Symbiotic simulation control system (SSCS):** This type is a closed-loop system. The symbiotic simulation results are considered as a decision which will be directly applied to the physical system by means of actuators. The Dagstuhl Seminar definition of symbiotic simulation matches the definition of this type.
- **Symbiotic simulation forecasting system (SSFS):** This type is an open-loop system. The simulation can predict the behaviour of the physical system in different scenarios without communicating any suggestions or decisions to a *Decision maker* component or to the physical systems. The simulation results can be used for further analysis by external tools.

- **Symbiotic simulation model validation system (SSMVS):** This type is an open-loop system. The purpose of the simulation of this type is to produce a model (or a set of reference models using different what-if scenarios) that represents the current state of the system, called reference model. Then the reference model can be used for validation purposes.
- **Symbiotic simulation anomaly detection system (SSADS):** This type is an open-loop system. It uses a reference model, e.g. produced by an SSMVS, of the physical system to compare its behaviour with the behaviour of the physical system. If the two behaviours are inconsistent then the system deduces that an anomaly may exist.

The different types of the symbiotic simulation paradigm have been used in the various applications. In [148] a framework for incorporating symbiotic simulations in the enterprise production systems. The framework combines the different types of symbiotic simulation where each type serves specific part of the production system. In [149] a symbiotic simulation-based technique is used in a framework for Traffic Management. Historical information about the traffic dynamics is fed to the simulator to perform what-if simulations to test alternative strategies for traffic management. The alternatives are compared and the best strategy will then be recommended for actual implementation. In [150] a symbiotic simulation-based framework is designed to support the resources provisioning in the cloud. A set of pre-formulated policies for virtual machines provision and real-time measurements of running cloud applications are fed to the simulation system. The system uses these inputs to optimise the cloud system by selecting the appropriate provisioning policies and applying them to the cloud

system. Furthermore, in the Dynamic Data-Driven Application Systems (DDDAS) paradigm, a closed-loop simulation system is used to steer the data collection about the real application [151]. The measurements gathered from the real application are used by the simulation for performance prediction. Then, using the simulation results, the simulator guides the data measurement process for the sake of achieving efficient data collection. In this context, the DDDAS paradigm can be viewed as a symbiotic simulation control system. Other symbiotic simulation related works can be found in the literature.

The meta-self-awareness approach proposed in this chapter focuses on both the state of the system queue and the benefits of each awareness level as factors that drive the switching between the awareness levels. In order to determine the benefits of each of the awareness levels, what-if experiments can be performed to simulate the performance of alternative awareness levels and obtain feedback. Since the state of the system queue is another factor in the switching decision-making process, the feedback should not be applied directly as a decision. Therefore, we adopt an SSDSS to provide a suggestion for adopting a certain level of awareness, as detailed in the following section.

### **6.3 Symbiotic-based Meta-self-awareness Approach**

The results of the previous chapter revealed that the different levels of awareness provide the system with different capabilities of self-adaptivity, based on the used type of knowledge. Each level has advantages and disadvantages in terms of achieving the goals of the system, i.e. improving the percentage of satisfied requests, the resources waste, and the waiting time. For example, making the selection and adaptation decisions using the interaction-awareness level confers high percentage of satisfied requests but with high waiting times relative to the making the decisions using the stimulus-awareness

level. The high waiting times mean that the requests will remain longer time in the system's queue. This may result in rapid growth in the queue size and consequently dropping the incoming requests down to decreasing the percentage of satisfied requests. Furthermore, the obtainment of the interaction- and time-awareness advantages is subject to the availability and the quality of the corresponding knowledge. As the system normally starts with 'zero-knowledge' about the performance of the services, it will be illogical to use the time- or the interaction-awareness levels in the first period of the system operation. The system will spend some time collecting and representing the knowledge that would be sufficient to capture the performance patterns of the services so that it can use the time- and interaction-awareness levels. The questions that arise here are (1) how to judge on the quality of the collected knowledge? and (2) which level of awareness should be used?

Our approach to answering these questions is to equip the system with self-adaptation capabilities at the meta-level. These capabilities are realised through a meta-self-awareness level which acts as the 'brain' of the self-awareness framework. It is useful to manage the trade-offs that exist between the system goals. The role of the meta-self-awareness is to enable the system to switch between the levels of awareness (stimulus-, time-, and interaction-awareness) based on the advantages and disadvantages of each level. This ability 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). For this purpose, the meta-self-awareness investigates the quality of the knowledge using a symbiotic simulator. The simulator performs what-if experiments to compare the benefits and

overheads of the three levels and suggest a level to a *Decision maker* component. The *Decision maker* also receives information about the queue status and analyses the queue's stability. Consequently, the *Decision maker* decides on an awareness level. In the following, we present the internal structure of the meta-self-awareness level, see Figure 6.3, and explain the adaptation decision-making process at this level.

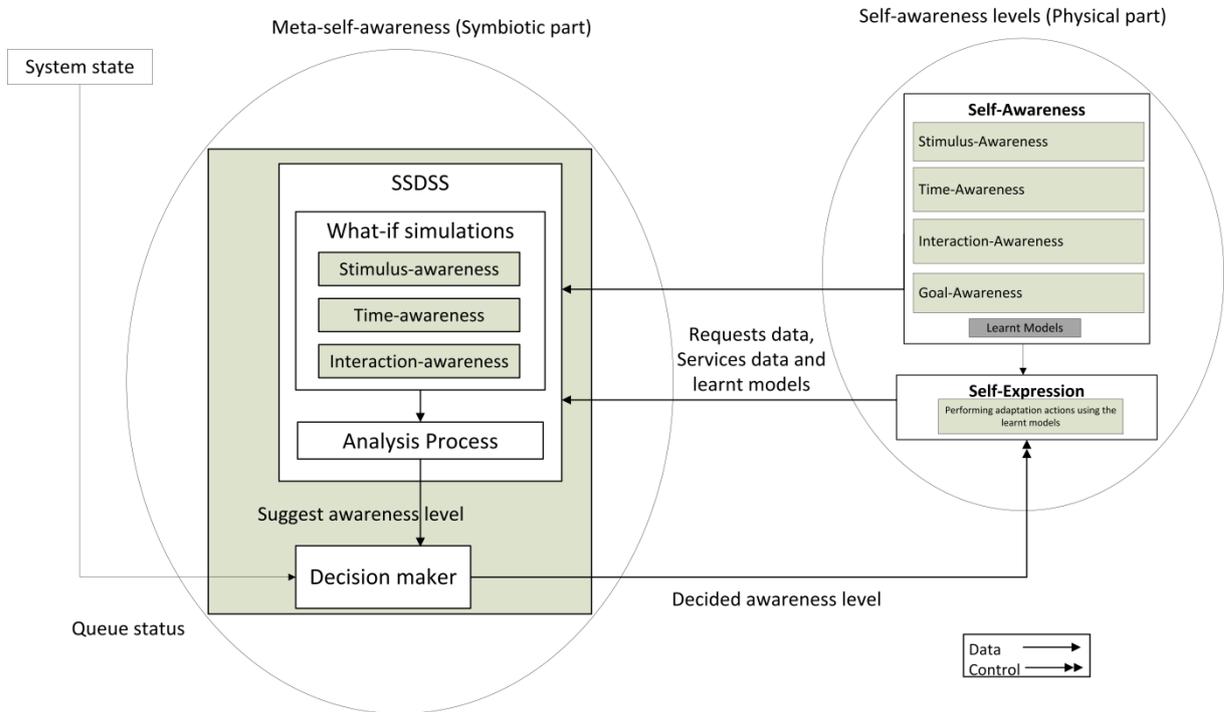


Figure 6.3: Architecture of The Meta-Self-Awareness Level

### (1) Adaptation Checkpoints

The decision of switching from one awareness level to another is based on the periodic evaluation of system's 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 runtime according to the changes in the system state.

In order to update  $T_p$ , we adopt a strategy similar to the one used in [152]. After each evaluation period, if the evaluation resulted in no transition to a new awareness level, then  $T_p$  is increased to  $\min(2T_p, T_{max})$ . Conversely, if the evaluation resulted in a transition to a new level, then  $T_p$  is decreased to  $\max(T_p/2, T_{min})$ .

## **(2) Symbiotic Simulation Decision Support**

As shown in Figure 6.3, the meta-self-awareness level leverages a symbiotic feedback loop which performs what-if experiments to investigate alternative awareness levels. The decision of adopting one of the awareness levels is made by the *Decision maker* component which considers the suggested awareness option (by the *Analysis process*) along with the queue status. The symbiotic feedback loop consists of two components, (1) the *What-if simulations* and (2) the *Analysis process*.

At every checkpoint, the system passes the set of requests which have been processed by the real system and the models learnt by the stimulus-, time-, and interaction-awareness levels. Then the what-if simulator simulates the processing of the requests using the real learnt models in order to investigate the alternative decisions which could have been made using the other awareness levels. The simulation results will be 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 percentage of satisfied requests (PSR), resources waste (RW), and waiting time (WT), which are defined in chapter 4.

Upon receiving the simulation results at every checkpoint, the *Analysis process* compares the resulting performance measures of the awareness levels to find out which level is the best; relative to the other levels. The *Analysis process* applies a simple additive

weighting [153] 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 metrics values form a matrix,  $D$ , which has the following form:

$$D = \begin{matrix} & PSR & RW & WT \\ Stimulus & r_{11} & r_{12} & r_{13} \\ Time & r_{21} & r_{22} & r_{23} \\ Interaction & r_{31} & r_{32} & r_{33} \end{matrix} \quad (6.1)$$

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

- Step 2: Since the waiting time and the resources waste are negative criteria (i.e. the higher the value the lower the benefit) whereas the satisfied requests percentage is a positive criterion (i.e. the higher the value the higher the benefit) and since each metric has its own range and units, the metrics should be scaled to make them comparable. The scaled matrix,  $N$ , has the form:

$$N = \begin{matrix} & PSR & RW & WT \\ Stimulus & n_{11} & n_{12} & n_{13} \\ Time & n_{21} & n_{22} & n_{23} \\ Interaction & n_{31} & n_{32} & n_{33} \end{matrix} \quad (6.2)$$

where  $n_{ij} = \frac{r_{ij} - r_j^{min}}{r_j^{max} - r_j^{min}}$  for the satisfied requests percentage,  $n_{ij} = \frac{r_j^{max} - r_{ij}}{r_j^{max} - r_j^{min}}$  for the waiting time and the resources waste,  $r_j^{max}$  is the maximal value of a metric in matrix  $D$ , and  $r_j^{min}$  is the minimal value of a metric in matrix  $D$ .

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

$$V = \begin{pmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \\ v_{31} & v_{32} & v_{33} \end{pmatrix} = \begin{pmatrix} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \\ n_{31} & n_{32} & n_{33} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} \quad (6.3)$$

where  $w_i \geq 0$  and  $\sum w_i = 1$ .

- Step 4: The overall benefit of each level is computed by adding the corresponding row elements.

The level with the maximal benefit value will be suggested to the *Decision maker* component. In the rest of this chapter, we denote *suggested = S*, *suggested = T*, or *suggested = I* to express the suggestion of the stimulus-, time-, or interaction-awareness levels, respectively.

It is worth mentioning here that the criteria used in the symbiotic simulation to compare the levels of awareness emerge from the application domain. That means, in different contexts different criteria can be used. Similarly, the levels of awareness which are simulated in the symbiotic simulator can vary according to the application domain.

### **(3) Adaptation at the Meta-Level**

The *Decision maker* component performs the actual adaptation at the meta-self-awareness level. The Decision maker may or may not consider the suggestion from the *Analysis process*. The adaptation decision depends also on the following criteria:

- Queue Stability. According to Loynes' theorem [154], if the requests arrival and serving processes are independent, then the queue is stable if the requests arrival rate  $\lambda$  is less than the serving rate  $\mu$ . This condition is necessary to avoid indefinite growth of the queue. Thus, the meta-self-awareness should tend to choose the awareness level such that the queue stability is achieved. Obviously, the different

levels of awareness exhibit different serving rates due to the different amount of computation performed at each of the levels. We use the queue utilisation factor  $\rho = \lambda/\mu$  and require  $\rho \leq 1$  for the queue to be stable.

- Percentage of Dropped Requests (PDR). When the requests arrival rate is greater than the serving rate, the queue will grow. Then when the queue becomes fully occupied the system starts to drop the incoming requests. We define *PDR* as the number of those dropped requests divided over the total number of requests and require  $PDR < PDR_{th}$ , where  $PDR_{th}$  is the threshold under which the percentage of the dropped requests is acceptable. As can be expected, the meta-self-awareness should tend to activate the awareness level such that  $PDR < PDR_{th}$ . It is worth mentioning that we consider this criterion to deal with the case of occupying the whole queue capacity.

In the following, we present the transitions from the current level to the next one, as shown in Figure 6.4:

*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. In this case, if ( $PDR \geq PDR_{th}$ ) then the system should notify the admin and advise her to increase the queue capacity.
- If ( $(\rho \leq 1) \text{ AND } (suggested == S)$ ), then although the queue is stable, no transition is performed because no benefit is expected from the other levels of awareness (e.g. because the captured knowledge is not adequate to be used).

- If  $((\rho \leq 1) \text{ AND } ((\text{suggested} == T) \text{ OR } (\text{suggested} == I)))$ , then the system performs a transition to the suggested awareness level.

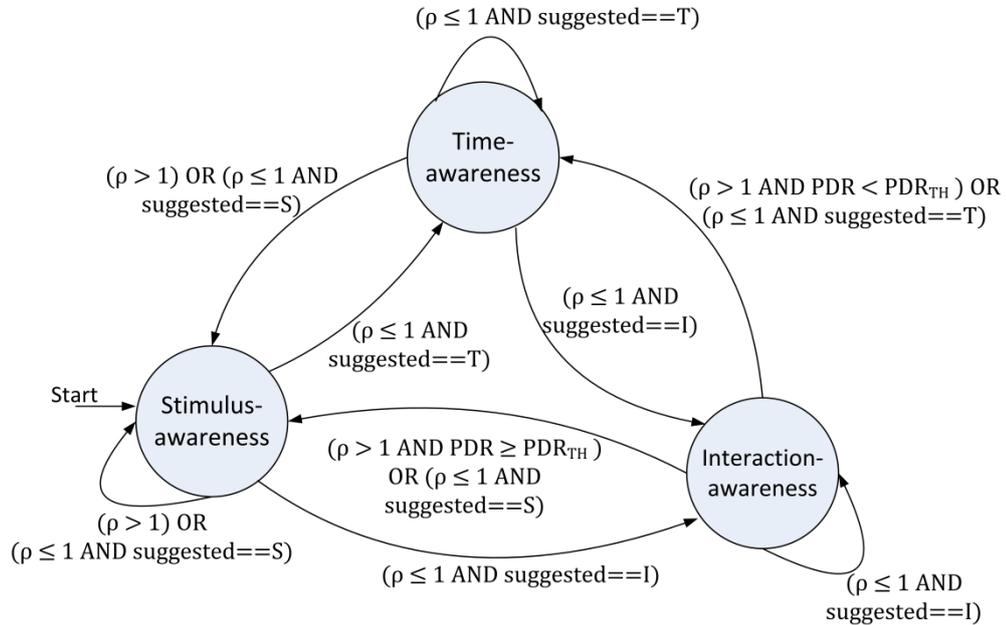


Figure 6.4: The State Diagram of The Awareness Levels Switching

*Current state is time-awareness:*

- If  $(\rho \leq 1 \text{ AND suggested} == T)$ , then the queue is stable and the time-awareness is suggested. Therefore, no transition is performed.
- If  $((\rho \leq 1) \text{ AND } ((\text{suggested} == I) \text{ OR } (\text{suggested} == S)))$ , then the queue is stable and the system performs a transition to the suggested awareness level.
- If  $(\rho > 1)$ , then the queue is growing up, then the system transitions back to the stimulus-awareness level in order serve the requests faster.

*Current state is interaction-awareness*

- If  $((\rho > 1) \text{ AND } (PDR \geq PDR_{th}))$ , then the system transitions to back to the stimulus-awareness level (regardless of the suggested level) in order serve the requests faster.
- If  $((\rho > 1) \text{ AND } (PDR < PDR_{th}))$ , then the system transitions to the time-awareness level in order serve the requests faster.
- If  $((\rho \leq 1) \text{ AND } ((suggested == T) \text{ OR } (suggested == S)))$ , then the queue is stable and the system performs a transition to the suggested awareness level.
- If  $(\rho \leq 1 \text{ AND } suggested == I)$ , then the queue is stable and no transition is performed.

## 6.4 Experimental Evaluation

### 6.4.1 Performance of Meta-self-awareness

We compare the meta-self-aware version of the self-aware architectural framework with the non-meta-self-aware version, i.e. the stimulus-, time-, and interaction-aware approaches. The experiments compare the four approaches in terms of the criteria mentioned above, namely, the PSR, the RW, and the WT. The value of the threshold PDR is set to 0.1.

**Comparison in PSR**      The first set of experiments compares the percentage of well-satisfied requests over simulation time. Figure 6.5 (a), (b), (c), and (d) shows the average PSR for varying arrival rates  $\lambda$ . The figures show that the average PSR of the interaction- and the time-awareness cases are higher than the stimulus-awareness cases in the initial period of simulation time. However, the PSR of the interaction- and the time-

awareness drops when the requests arrival rate gets high. More importantly, the figures show that the average PSR of the meta-self-aware case matches the best PSR of the other three cases. For example, in Figure 6.5 (a), the PSR of the meta-self-awareness case is the same as the time-awareness until the PSR of the interaction-awareness gets higher than the time-awareness (after simulation time = 300). At that point, the PSR of the meta-self-awareness will be the same as the interaction-awareness case. In Figure 6.5 (b) and (c) after 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

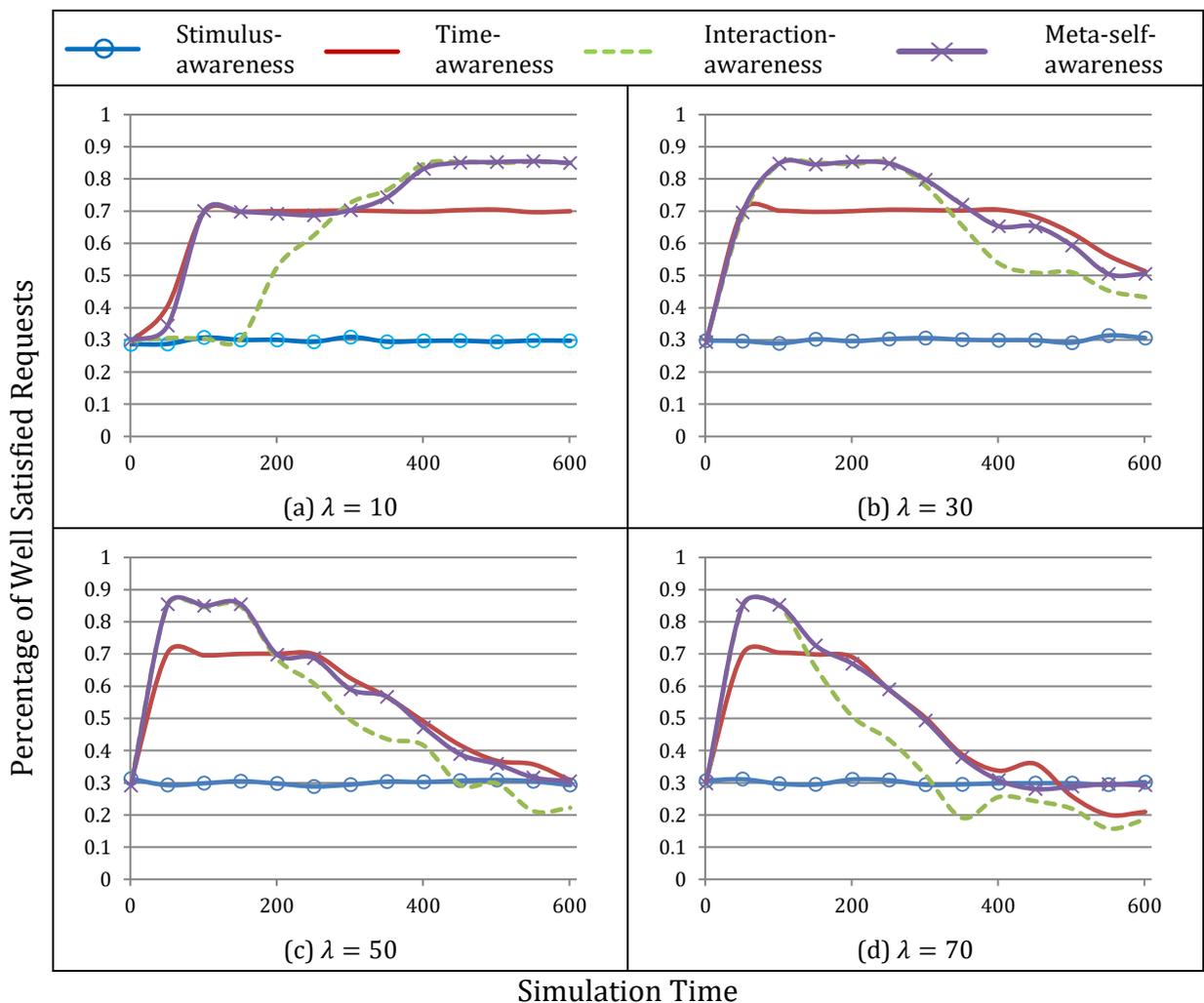


Figure 6.5: Comparison in Percentage of Satisfied Requests

case. Another example in Figure 6.5 (d), the PSR of the meta-self-awareness case will be close to the stimulus-awareness case (after simulation time =400) which has the best PSR at that point. Therefore, having the advantage of switching between the awareness levels by the meta-self-awareness approach results in satisfying a higher number of requests.

**Comparison in RW**      The second set of experiments compares the average resources waste over simulation time. Figure 6.6(a), (b), (c), and (d) shows the average RW for varying arrival rates  $\lambda$ . The figures show that the average RW of the stimulus-awareness case is higher than the interaction- and the time-awareness cases in the initial period of simulation time. However, the average RW of the interaction- and the time-awareness rises when the requests arrival rate gets high. More importantly, the figures show that the average RW of the meta-self-aware case matches the average RW of the adopted awareness approach. For example, in Figure 6.6 (a), the RW of the meta-self-awareness case is the same as the time-awareness until simulation time = 300. This is because the adopted awareness level in this period is the time-awareness level. 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 (6.3). Figure 6.6 (a) shows also that the RW of the meta-self-awareness is the same as the interaction-awareness case after simulation time = 300, which is the least RW among the other approaches. In Figure 6.6 (b) and (c) the RW of the meta-self-awareness is the same as the interaction-awareness, which is the best one in that case until simulation time is 300 and 200 respectively. Another example in Figure 6.6 (d), the RW of the meta-self-awareness case will be close to the stimulus-awareness case (after simulation time = 400) even though it is not the least RW at that point. The reason is that the PSR has a higher weight in the experimentations setup of the what-if simulations. This higher

weight makes the suggestion decisions biased towards suggesting the level that is expected to have higher PSR. However, having the advantage of switching between the awareness levels, by the meta-self-awareness approach, has the advantage of reducing the RW if the difference in the expected PSR of the levels is not significant (in case the PSR has a higher weight than RW).

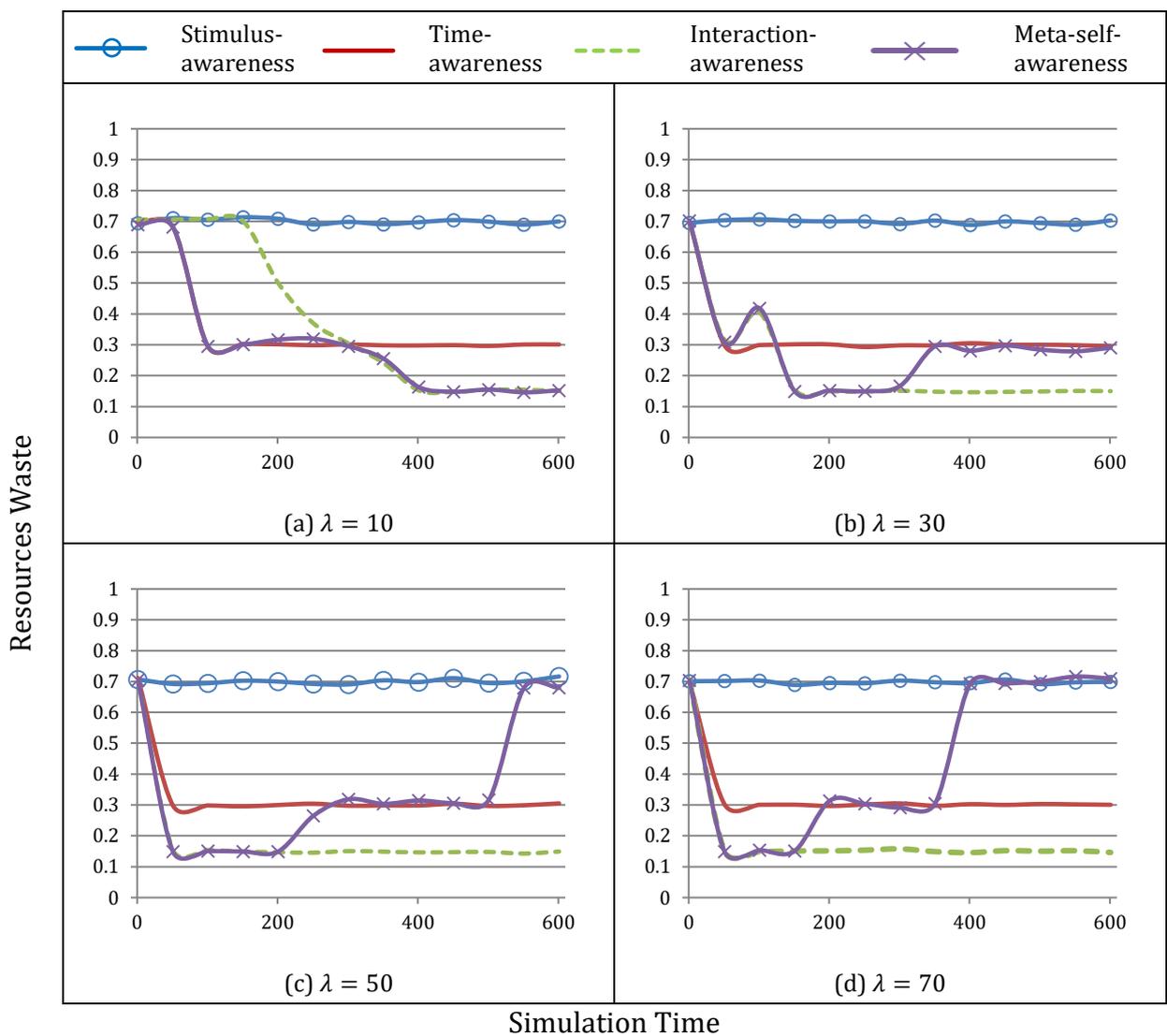


Figure 6.6: Comparison in Resources Waste

**Comparison in WT**      The third set of experiments compares the average waiting time WT waste over simulation time. Figure 6.7 (a), (b), (c), and (d) shows the average WT for varying arrival rates  $\lambda$ . The figures show that the average WT of the stimulus-awareness case is always smaller than the interaction- and the time-awareness cases. The average RW of the interaction- and the time-awareness rises more rapidly, but linearly, when the requests arrival rate gets high. 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 Figure 6.7 (a), the WT of the meta-self-awareness case is the same as the time-awareness until simulation time = 300. This is because the adopted awareness level in this period is the time-awareness level. Figure 6.7 (a) shows also that the WT of the meta-self-awareness is the same as the interaction-awareness case after simulation time = 300, the period in which is the interaction-awareness level is adopted. In Figure 6.7 (c), the WT of the meta-self-awareness case will drop close to the stimulus-awareness case (after simulation time = 500) which is adopted in that case. Another example in Figure 6.7 (d), the WT of the meta-self-awareness case will be close to the stimulus-awareness case (after simulation time = 400) which is adopted in that case. Therefore, the switching between the awareness levels by the meta-self-awareness approach results in changes in the waiting time according to the adopted level and reducing the waiting time when the requests arrival rate is high.

The experimentation results show that the meta-self-awareness approach enables the switching between the different levels of awareness based on the expected performance of each of the levels and on the queue state. The results show that the meta-self-awareness tends to select the interaction-awareness level when the arrival rate is low and after a period of the simulation time. Such period is required to acquire and refine

the knowledge on the performance of the services. However, since the adoption of this level results in high overhead when the arrival rate is high, the meta-self-awareness switches to another level of awareness in order to reduce the waiting time.

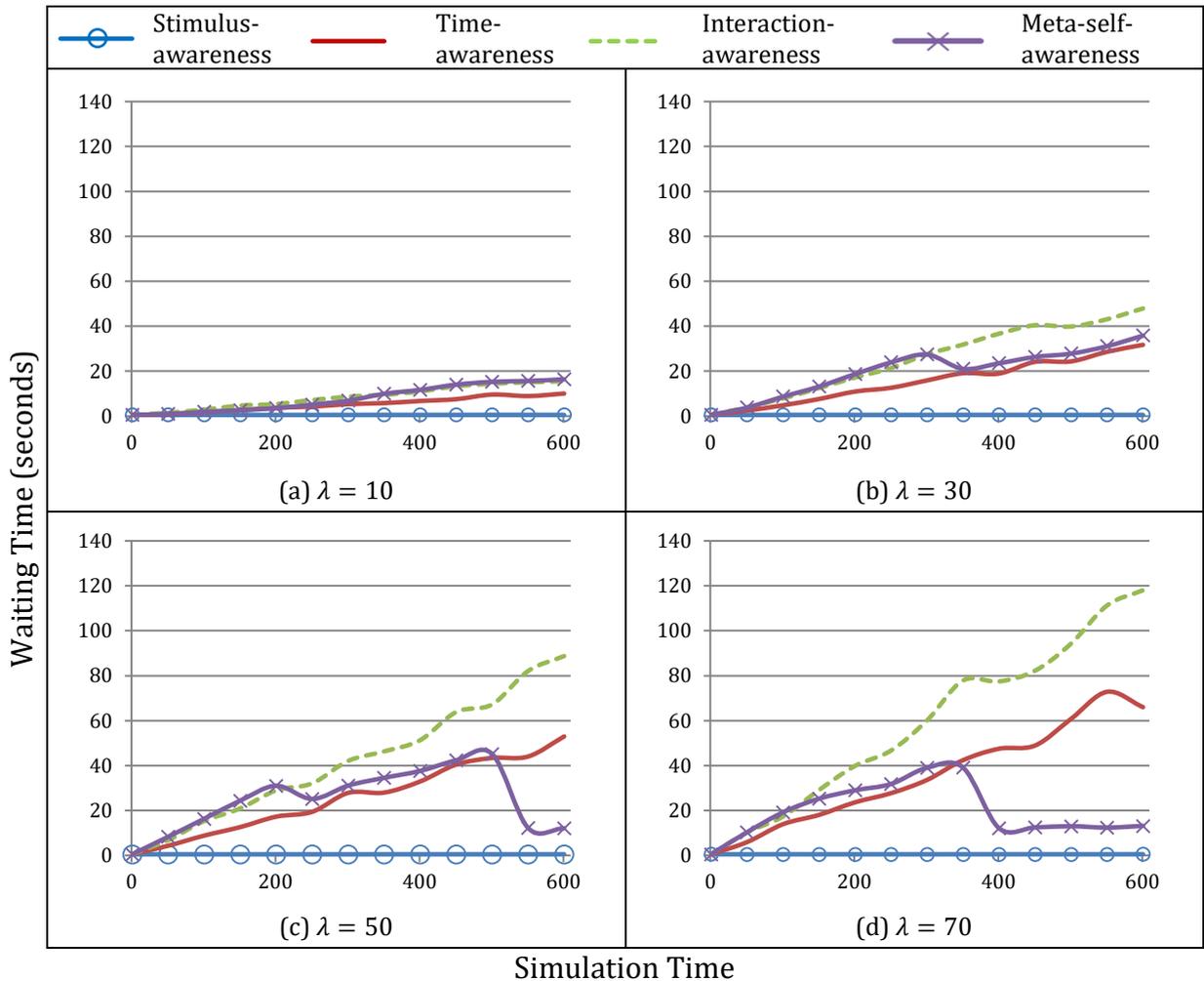


Figure 6.7: Comparison in Waiting Time

The results also 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 an awareness level. As we mentioned in regards to Figure 6.6(a), the adopted level in the period where simulation time  $\leq 300$  was the time-awareness although the

stimulus-awareness exhibited the minimum waste. That is because we assigned a higher weight to the PSR metric in these experiments.

#### **6.4.2 Overhead of Meta-self-awareness**

Naturally, although the symbiotic simulation provides support of self-adaptation at the meta-level, it adds computational overhead to the use of the self-aware framework. The computational overhead results mainly from re-computing the selection and adaptation decisions using the different levels of awareness during the what-if analysis. Figure 6.8 compares the computational time of meta-self-aware and the non-meta-self-aware approaches. The computation time is measured as the time consumed for processing all the requests divided by the number of requests. The figure shows that the computational time of the meta-self-awareness approach exhibits high computation time compared to the other approaches. The figure shows also that the computation time increases with the increase of the requests arrival rate, as more computation is needed to serve increasing number of requests to search for and adapt the composite services. Furthermore, the figure shows that the computation time increases over time. This increase is due to the increase in the size of the knowledge as more knowledge about the performance of the services is accumulated. However, further research is required to investigate the acceptance of this overhead compared to the benefits of the meta-self-awareness level. In relation to that, exploring the optimal placement of symbiotic simulator, i.e. on the same machine running the self-aware framework or on dedicated computing machine to parallelise the computation and reduce the computation overhead, is another area of research for further work.

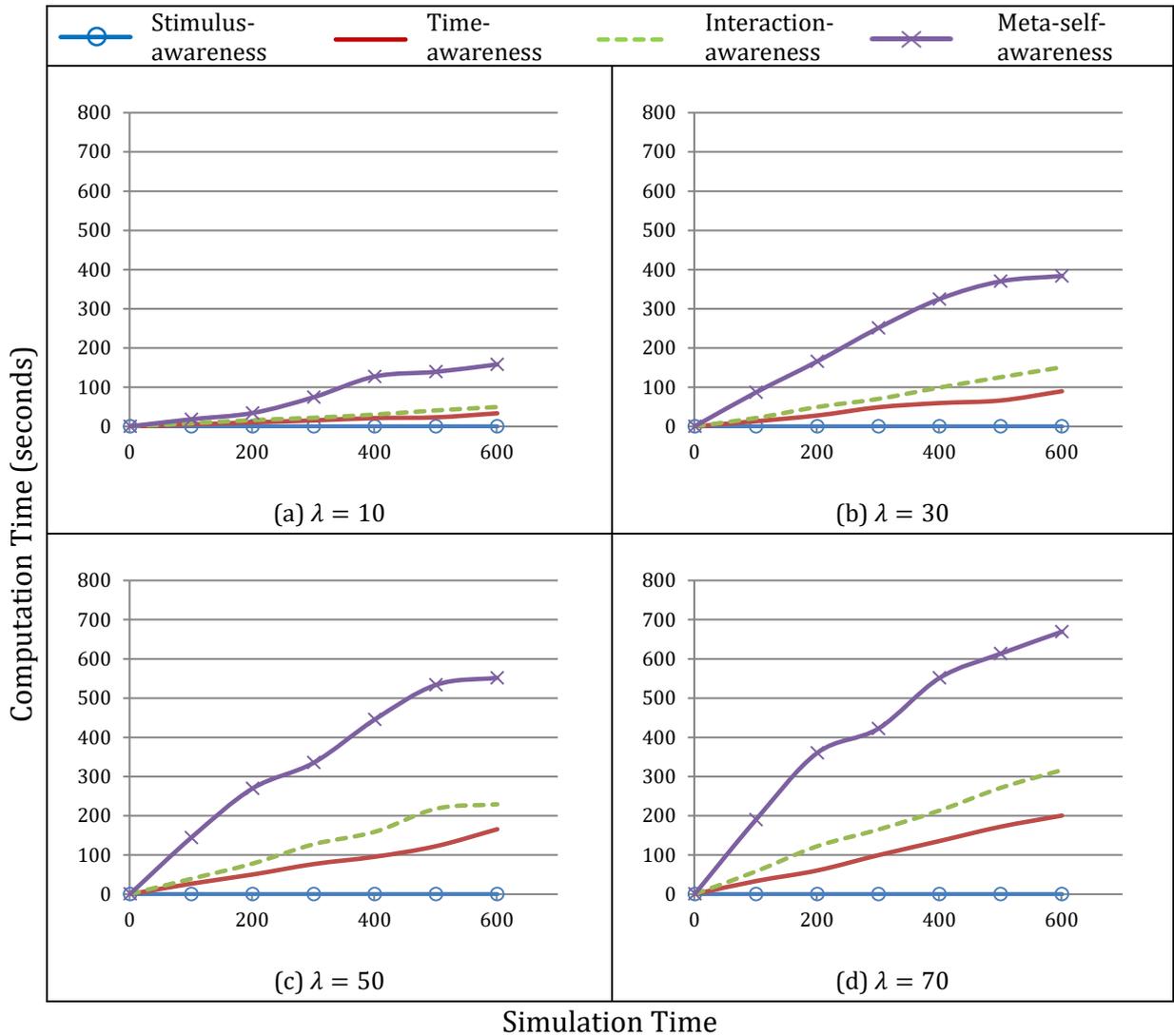


Figure 6.8: Comparison in Computation Time

## 6.5 Conclusion

In this chapter, we have answered the thesis' research question related to the management of the trade-off between the different levels of awareness by proposing an architecture and providing a quantitative approach for self-adaptation at the meta-level. This meta-level of awareness enables systematic switching between the levels of awareness based on their advantages and disadvantages.

The mechanism of the meta-self-awareness approach is two-fold; on the one hand, the approach evaluates the performance of the different awareness levels and the alternative decisions that could have been taken by the other levels of awareness. For this purpose, a symbiotic simulation decision support system is used to perform what-if experiments of the alternative decisions. The what-if experiments enable evaluating the quality of the collected knowledge by simulating the alternative selection and adaptation decision which could have been taken using that knowledge. Then the performance measures of the awareness levels are used to find out the optimum awareness level. This level will be passed to the *Decision maker* component as a suggestion.

On the other hand, the meta-self-awareness approach takes into consideration the system state in terms of the queue stability in order to avoid the rapid growth of the queue. The stability is defined according to Loynes' theorem [154] which appries that the queue is stable if the requests arrival rate is less than the requests serving rate. Subsequently, using the queue state and the suggestion of the awareness level, the meta-self-awareness approach selects the optimum awareness level and effects this decision on the system.

The chapter evaluates the performance of the meta-self-awareness approach in relative to non-meta-self-awareness approaches. The objective of the evaluation is to investigate the behaviour of the system in the presence of the meta-self-awareness. For this purpose, the evaluation compares the meta-self-awareness with the stimulus-, time-, and interaction-aware approaches. The results show that the performance of the system when using the meta-self-awareness approach is convergent with the best case performance of the system when using the other approaches. In other words, when using

the stimulus-, time-, or interaction-awareness, the performance of the system exhibits degradation in some circumstances. Whereas, in the meta-self-awareness case, the system avoids such degradation, thanks to the capability of adapting the awareness level provided by the meta-self-awareness. However, the usefulness of the meta-self-awareness comes with overhead in terms of computational time, an area that requires further future research.



# CHAPTER 7

## REFLECTIONS, CONCLUSION REMARKS, AND FUTURE WORK

### 7.1 Reflections

In this section, we reflect on the thesis using different qualitative criteria for dynamic software systems, including complexity, scalability, overhead, and practical deployment.

#### 7.1.1 Complexity

As mentioned in chapter 1, the complexity of managing dynamic software systems is attributed to *dynamism*, *uncertainty*, and *heterogeneity* of the environment [10] [98] [33]. To manage complexity and react to continuous changes, self-adaptivity has been acknowledged as a solution. The autonomic behaviour needs to be supported with knowledge about the different components that contribute to the complexity of the system. Once the knowledge about these components is captured and modelled, it provides the decision-making process with the possibility to take more intelligent decisions in response to the changes. However, the knowledge can evolve and change over time. Thus, continuous knowledge management is necessary to ensure better

support for the adaptation decision making. This thesis provides a framework for handling such situations and demonstrates its capabilities using the volunteer storage services scenario.

- ***Dealing with dynamism.***

- The self-awareness framework provides the primitives for monitoring the behaviour of dynamic software systems, where the volunteering environment was taken as an example. The internal and external changes will be reported to the self-expression component, which reacts by an adaptation action. The reaction is supported by the knowledge modelled by the different levels of awareness, as shown in section 5.6. Furthermore, the dynamism is also related to the changes in the knowledge itself. For example, the performance patterns of the volunteers may change. The continuous update of the knowledge, e.g. the continuous maintenance of the dynamic histograms (see section 5.5), ensures having up-to-date knowledge for better informing the adaptation decisions. In addition to that, the framework is able to deal with the dynamism related to the changes in the workload through the meta-self-awareness level, which enables the switching between the levels (see section 6.3). For example, a spike in the workload resulting from a sharp increase in the requests arrival rate will motivate the usage of the stimulus-awareness to provide faster processing of the requests.
- This thesis assumes that the values of the thresholds and the weights used in the approach are specified by the system administrator or a domain expert. Potential extensions can look at how the thresholds can be adjusted at runtime

i.e. how to make the thresholds dynamic. For this purpose, the context, the changes in the context, and the attributes which are related to the changes in the context, should be taken into account.

- ***Dealing with uncertainty.*** Service provision in dynamic environments exhibits fluctuations which may result in unsatisfying the users' requirements. The proposed knowledge management approaches attempted to quantify and measure the deviations from the promised quality of services provision exhibited by the providers, i.e. the volunteers in our case, in terms dependability (see section 5.6.2). Then the dependability is then incorporated in the selection and adaptation processes in order to increase the probability of selecting more dependable services, thus, reducing the uncertainty associated with the service provision. Furthermore, as the uncertainty can be associated with the collected knowledge, the meta-self-awareness provides a means for analysing the knowledge through the symbiotic-based approach, as shown in section 6.3. The what-if analysis provides a way to judge on the quality of the knowledge to investigate the expected outcome of using the collected knowledge. Consequently, the meta-self-awareness level deals with the uncertainty in the decision-making process by switching between the awareness levels.
- ***Dealing with heterogeneity.*** The heterogeneity of the infrastructure contributes to the complexity of the system since different computing resources have varied capabilities. In our case, we decided to deal with heterogeneity through utility and dependability. The proposed utility model deals with the heterogeneity by expressing the capabilities of the contributed resources as utilities (see section 4.3.3). Then the utilities are used to reason about the selection decisions. After

that, the knowledge management approaches deal with the heterogeneity by quantifying and measuring the dependability of the services (see section 5.6.2). The approaches do not require prior knowledge about the contributed computing machines since the dependability information will reflect the ability of those resources to satisfy the requirements.

Collectively, the above points summarise our approach in dealing with the complexity of the dynamic software systems, as quantitatively shown in the previous chapters. The dynamic knowledge management provides concrete grounds for continuous optimisation of the quality of the system.

### **7.1.2 Scalability**

The scalability of a system expresses its performance in the presence of relatively large amounts of data or workload. In this section, we elaborate on the effects of the amount of historical knowledge and the amount of workload on the scalability of the proposed self-aware approach.

- ***The effect of the amount of historical knowledge.***
  - The size of the dynamic histograms, in terms of the number of data points, may endlessly increase during the operation of the system. Massive increase will influence the scalability of the system as this will require more computations for estimating the services dependabilities and for maintaining the dynamic histograms. This can be treated by adopting a *pruning* strategy to get rid of the less important data points. An example is the *forget* strategy that we used to drop the old data points when a bucket cannot be split, as shown in section 5.5.2. An alternative *forget* strategy can be to drop the *outlier* data points, e.g.

the data points that have values far from the average values of the other data points. A combination of the two strategies is a third possible strategy.

- In the case of interaction-awareness, the pair-wise knowledge about the services interactions can grow fast. This may influence the decision making of the meta-self-awareness level by avoiding the switching of the interaction-awareness level. A possible way to deal with this situation is to group the services into clusters based on the interaction information. That is, services that exhibit frequent interactions will belong to the same cluster. Then the interaction-awareness level can limit the knowledge modelling to the scope of the clusters, i.e. by considering the interactions of a service with the services in its cluster only.

- ***The effect of the workload***

- The number of services in the service repository influences the scalability of the system as more computation will be carried out when the number of services grows. As the evaluation results show, the effect of the increase in the number of services on the scalability is generally limited. This is because the computation of the utility models and the dependabilities is efficient. However, this may affect the computation of the interaction-awareness case, as explained above.
- Increasing the number of services' attributes (storage, availability, security, etc.) will have a larger effect on the scalability. This is due to the need of finding the non-dominant set of services iteratively in the search for composite services. Although we use an efficient algorithm [155] for finding the non-dominant set, the performance can be improved further (in the high scale

case) by using genetic algorithms (e.g. based on Based on NSGA-II) to find a sub-optimal approximation of the non-dominant set of services.

- The increase in the workload due to an increase in the requests arrival rate also affects the scalability especially in the time- and interaction-awareness cases. In extreme case, this may result in dropping some of the requests if the system's queue is fully occupied. The meta-self-awareness responds to such situations by switching to the stimulus-awareness level in order to process the requests faster and notifying the system administrator to increase the queue capacity.

### **7.1.3 Overhead**

The main overhead of our proposed approach can result from the use of the symbiotic simulation, as discussed in section 6.4.2. This is because investigating alternative decisions by performing the what-if analysis requires recomputing the selection and adaptation decisions using the different levels of awareness. This overhead can increase significantly when both the requests arrival rate and the historical data size increase. In such case, a dedicated machine can be used for the symbiotic simulation. This leads to an interesting research to dynamically optimising the placement of the symbiotic simulation based on the state of the system in terms of the scale and the computing requirements of the symbiotic simulator.

### **7.1.4 Practical Deployment**

In this thesis, our work is on the fundamentals of engineering self-aware software systems with knowledge management. Therefore, the experimentation has been done using simulations due to scalability issues. In other words, having a reasonable number

of volunteers for iterative experimentation is impractical. Further development and research will be required for applying our proposal in real life application. Nevertheless, the architecture of the volunteer storage system (presented in chapter 5) can be an example of how the proposed approach can be deployed in a real scenario. For this purpose, a real application has been developed in which the storage services has been implemented as RESTful web services using the Jersey framework [156]. The system's queue has been realised using the Java Message Service [157]. The service repository has been realised as a PostgreSQL database [158]. The GlassFish server [159] has been used as a web server to host the web services at the volunteers' machines. Figure 7.1 shows an example of the user interface where the request is served by a composite service that involves two services.



Figure 7.1: Example Showing The Interface of A Subscriber Using A Composite Service of Two Volunteer Services

## 7.2 How the Research Questions are Addressed

In this section, we review how the research questions, introduced in chapter 1, have been addressed throughout the thesis.

### **RQ1) How to characterise self-awareness and what motivated the research and applications of self-awareness in software engineering for dynamic and scalable software systems?**

In chapter 2, we have conducted a systematic literature review for the purpose of getting a deep understanding of the computational self-awareness. The review aimed also at exploring the motivations behind incorporating the self-awareness concept in software systems. The findings of the SLR show that there is no unified definition for self-awareness, as this research topic is still evolving. However, the progress in this topic is visible and demonstrated in the contributions of the EU Proactive Initiative Self-Awareness in Autonomic Systems [16] and road-mapping agenda of the Dagstuhl seminar [17]. Based on these works, we proposed the following definition of computational self-awareness, which is adapted from the definitions of [16] and [17]:

*A software system is self-aware if it:*

- *possesses knowledge about its internal state and its environment,*
- *supports fine-grained knowledge management,*
- *able to capture the performance patterns of its components (internal and external),*
- *supports both autonomic reactive and proactive adaptation at different levels, and*
- *is able to predict the likely effect of the adaptation actions/decisions.*

Our proposed approach is in line with this definition as follows:

- The adopted self-awareness framework is able to capture knowledge about the internal state of the system by monitoring the stability of the systems' queue and by observing the performance of the different levels of awareness.
- The framework is able to capture knowledge about the environment by monitoring the performance of the services involved in composite services.
- The proposed dynamic knowledge management approach separates the knowledge concerns and provides concrete algorithms for managing and using the knowledge to inform the decision making.
- The framework is able to reactively respond to the changes by replacing the services that violate the requirements and switching between the awareness levels. The framework also supports proactive adaptation by using the captured performance patterns and replacing the services if the performance is expected to drop in the next timeslot(s). Also, monitoring the queue stability proactively prevents destabilising the queue when making the switching decision.
- The mechanism of the meta-self-awareness considers the likely effect of the adaptation actions by (a) analysing the adequacy and quality of the captured knowledge, through the what-if experimentations, and (b) avoiding the awareness level that may affect the stability of the queue, even though the knowledge at this level is adequate.

With respect to the motivations behind the adoption of computational self-awareness in dynamic systems, the SLR findings show that the general purpose is to achieve better autonomy for software systems. This is fulfilled in our approach by enriching the self-adaptation capabilities by the different knowledge management approaches.

**RQ2) What are the requirements for enacting self-awareness in dynamic software systems and how can these requirements be addressed?**

In chapter 3 we have demonstrated that self-awareness is required to enrich the self-adaptation capabilities in dynamic environments, e.g. the volunteer computing environment. However, enacting self-awareness requires the following:

- An approach for dynamic knowledge management that is able to provide in-depth information to support adaptation. This approach should be able to separate the knowledge concerns through being able to capture different types of knowledge from the collected data. Therefore, in chapter 5 we proposed fundamental improvements to one of the self-awareness approaches, i.e. the EPiCS approach. The improvements included proposing concrete algorithms for multi-level knowledge management, specifically, the stimulus-, the time-, and the interaction-awareness levels.
- The knowledge management approaches should take into consideration the specific characteristic of the environment in which the system operates. Since we are using VC as a steering scenario and that the volunteer services exhibit periodic and correlated performance patterns (chapter 3), we used dynamic histograms as dynamic data structures that are able to capture the performance patterns and the correlation of the volunteer services.

- The knowledge management approaches should treat the knowledge as moving targets. That means knowledge continuously evolve as that data arrives continuously and the performance patterns of the services may change at runtime. This requires flexible knowledge representation that allows frequent updates of the knowledge. Therefore in chapter 5, we proposed algorithms for dynamic maintenance of the dynamic histograms in order to keep them able to capture the latest performance patterns.

**RQ3) How can the system seamlessly switch between various levels of awareness while considering the adequacy and quality of the knowledge in enabling self-awareness? What mechanisms can help in coordination?**

Dynamic software systems can benefit from the dynamic knowledge that relates to the levels of awareness. The orchestration of this knowledge hopes to unlock new potentials for managing dependability, maintaining its levels and reasoning about its runtime tradeoffs, considering the current benefits at each state and the operational overheads. Specifically, in chapter 6 we have addressed the problem of managing the trade-offs between the levels of awareness by the meta-self-awareness level. This level assists in the problem of “systematic” switching between the different levels of awareness. It enables the self-aware system to adapt itself in response to the environmental changes which is a necessity for long-life systems where adaptation design decisions are difficult to be planned a priori and/or beyond the systems’ designers’ capabilities. The critical challenge of self-adaptation at the meta-level is to assess the quality of the dynamic knowledge, including its recency and decay, which is acquired and represented at each

of the self-awareness levels. Another challenge is that the knowledge can be inadequate making it difficult to anticipate the likely outcome of the adaptation decisions. Furthermore, performing online adaptations based on “best bet” or “trial and error” can be expensive exercise; it can lead to suboptimal adaptation and/or destabilise the system through unnecessary adaptations etc. To address this additional problem, we proposed a novel symbiotic-simulation-based approach to engineer the meta-self-awareness level. The symbiotic simulator is periodically fed by real-time knowledge to perform ‘what-if’ experiments in order to investigate the likely quality of alternative adaptation decisions, using each of the awareness levels. The symbiotic simulator level will then recommend the desirable decision and it is then passed to the *Decision maker* component which takes into consideration the recommendation along with the system state (in terms of the request arrival rate and the queue state). Afterwards, the *Decision maker* component decides whether to adapt to a different level of awareness or not.

**RQ4) How does VC, as a representative paradigm, render itself as a sensible environment for understanding and demonstrating potential improvements related to knowledge management in self-aware systems?**

The findings of the SLR show that the motivation behind the adoption of self-awareness in the increasing complexity of the current systems, which is due the dynamism, uncertainty and heterogeneity of the computing environments. Therefore, a case that obviously exhibits such characteristics is required to demonstrate the engineering of self-awareness. In addition to that, the environment specifics should be taken into account when designing the concrete knowledge management approaches for enacting self-awareness. For these reasons, we have selected the VC environment, for steering the

presentation of our approach. This environment exhibits high complexity (as shown in chapter 3), due to the lack of strict SLAs and dilution of control. Such situation allows the service providers (volunteers) to withdraw their services when they opt to or when they are unable to continue providing their resources. The consequences include unsatisfying the users' requirements and hence limited application of the VC as a computing paradigm [5].

In addition to the above, recent analysis studies reported specific characteristic related to the volunteers' performance. These studies [5] [111] [112] reported the presence of periodic and correlated performance patterns of the volunteers. Based on that, the awareness of such patterns can inform the selection, allocation, and adaptation of the volunteer resources. It enables the prediction of the volunteer hosts' performance, which helps to reason about the selection and adaptation decisions. The presences of these reports, along with the mentioned complexity of VC make this environment an interesting case for the development of self-awareness. On these grounds, in chapter 4 we developed a utility model for quantifying the contributed resources of the volunteer services. Then we developed an approach for selecting the services for composite services, as a basis for introducing self-awareness with knowledge management to support the adaptation.

### **7.3 Future Work**

Several future works that can be built on the work we have done in this thesis. In this section, we sketch some ideas that may be of interest for research.

- **Further approaches for capturing knowledge patterns.** In this thesis, we developed dynamic knowledge management approaches which take into

consideration the characteristics of VC. In more complex environments, other approaches may be required to provide better support for self-adaptation. For example, in scenarios where VC is integrated with a commercial cloud [116], multiple approaches can be developed to realise a certain level of awareness, e.g. regression-based and/or learning-based approaches in addition to the approaches presented in this thesis. Then the meta-self-awareness level can be extended to support the selection of the different algorithms realising the same level of awareness, in addition to its basic functionality of switching between the awareness levels.

- **Overhead management.** The overhead of utilising the symbiotic simulation leads to the need for further investigation in order to specify whether this overhead is acceptable relative to the usefulness of the symbiotic. The investigation may also involve developing an approach for deciding on the placement of the symbiotic simulator, whether on dedicated machines or on the same machine of the central manager, based on the required and consumed resources for each of them.
- **Experiments.** The evaluation in this thesis reports on controlled experiments based on simulations to compare the performance of the proposed approaches with basis related approaches. However, other kinds of experiments using real scenarios would be advised for putting the outcomes of this research in industrial settings. Although such experimentations will be complicated and costly, they will be required to provide a complete validation of the approach. Additionally, in the experimental evaluations, we fixed the thresholds and weights values. A possible extension is to investigate approaches for adjusting the thresholds and

weights dynamically taking into consideration the environmental changes and the system's goals.

- **Evaluating the presence of self-awareness in a system.** Existing approaches have evaluated self-awareness in relation to primitives that enable self-awareness and enrich the adaptivity. As an example, the EPiCS project [18] has structured the proposed framework around levels for self-awareness benefiting from stimuli, goal, knowledge, interaction, yet evaluating self-awareness presence is challenging to achieve. This calls for novel metrics, qualitative and quantitative frameworks for evaluating the presence/absence of self-awareness. This can consequently beg questions like: To what extent a system is self-aware? How can we claim that a system is more self-aware than its competitor?
- **Investigating performance patterns in the different computing environments.** The engineering of self-awareness with knowledge management can be supported by the availability of knowledge that gives indications about the system performance and the environment changes. Such indication can be obtained from many sources e.g. data traces of real systems. For this purpose, another possible area of future research would be to carrying out further work in monitoring and collecting real data from the real system operating in the different computing environments. Then collect data can be analysed or mined to inform the development of self-aware and knowledge management research.
- **Developing energy-aware approaches for volunteer computing.** Despite the advantages of providing free resources in the volunteer computing paradigm, optimising the energy consumption is still a challenge that limits the wide adoption of this paradigm. Future works that aim at developing self-aware

approaches for reducing the energy consumption in VC are required to make this paradigm more practical.

#### **7.4 Closing Remarks**

This thesis makes novel contributions to the field of engineering self-awareness in software systems. The thesis introduces a self-aware framework with dynamic knowledge management approaches, which benefit from reports of analysed traces of real data, with the absence of closely relevant work. The findings of this thesis can provide a better understanding of the computational self-awareness and the requirements of dynamic knowledge management when engineering self-awareness. The conducted experiments show evidence on the effectiveness of the framework in dealing with the complexity of the environments. We hope that the findings of this work along with the mentioned future research directions can stimulate research in advancing the state-of-art and practice of this category of systems.



# APPENDIX A

## GLOSSARY

Table A.1: The Acronyms in the thesis

<b>Acronyms</b>	<b>Description</b>
VC	Volunteer Computing
VS	Volunteer Service
CS	Composite Service
VSS	Volunteer Storage Service
VSC	Volunteer Storage Composition
WS	Web Service
WSC	Web Service Composition
RW	Resources Waste
PSR	Percentage of Satisfied Requests
WT	Waiting Time
PDR	Percentage of Dropped Requests
SLA	Service Level Agreement
SLR	Systematic Literature Review

## REFERENCES

- [1] R. Buyya, *High Performance Cluster Computing: Programming and Applications*: Prentice Hall PTR, 1999.
- [2] S. Distefano and A. Puliafito, "Cloud@Home: Toward a Volunteer Cloud," *IT Professional*, vol. 14, pp. 27-31, 2012.
- [3] J. Rao and X. Su, "A survey of automated web service composition methods," presented at the Proceedings of the First international conference on Semantic Web Services and Web Process Composition, San Diego, CA, 2005.
- [4] Z. Liangzhao, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for Web services composition," *IEEE Transactions on Software Engineering*, vol. 30, pp. 311-327, 2004.
- [5] 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, pp. 281-296, 2012.
- [6] G. H. Alférez and V. Pelechano, "Facing Uncertainty in Web Service Compositions," in *2013 IEEE 20th International Conference on Web Services*, 2013, pp. 219-226.
- [7] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, pp. 41-50, 2003.
- [8] R. de Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, *et al.*, "Software Engineering for Self-Adaptive Systems: A Second Research Roadmap," in *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*, R. de Lemos, H. Giese, H. A. Müller, and M. Shaw, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1-32.
- [9] 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, pp. 7267-7279, 2013.

- [10] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, pp. 1-42, 2009.
- [11] R. Laddaga and P. Robertson, "Self adaptive software: A position paper," in *SELF-STAR: International Workshop on Self-\* Properties in Complex Information Systems*, 2004, p. 19.
- [12] M. Nouman Durrani and J. A. Shamsi, "Volunteer computing: requirements, challenges, and solutions," *Journal of Network and Computer Applications*, vol. 39, pp. 369-380, 2014.
- [13] S. Parsons, R. Bahsoon, P. R. Lewis, and X. Yao, "Towards a Better Understanding of Self-Awareness and Self-Expression within Software Systems," University of Birmingham, School of Computer Science, Birmingham, UK April 2011.
- [14] F. O. Faniyi, "Self-aware software architecture style and patterns for cloud-based applications," University of Birmingham, 2015.
- [15] S. Dustdar, C. Dorn, F. Li, L. Baresi, G. Cabri, C. Pautasso, *et al.*, "A roadmap towards sustainable self-aware service systems," in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, 2010, pp. 10-19.
- [16] E. C. (FP7). (2010). *FP7: FET Proactive Initiative: Self-Awareness in Autonomic Systems (AWARENESS)*. Available: [http://cordis.europa.eu/fp7/ict/fet-proactive/aware\\_en.html](http://cordis.europa.eu/fp7/ict/fet-proactive/aware_en.html)
- [17] S. Kounev, X. Zhu, J. O. Kephart, and M. Kwiatkowska, "Model-driven algorithms and architectures for self-aware computing systems (Dagstuhl Seminar 15041)," *Dagstuhl Reports*, vol. 5, 2015.
- [18] T. Chen, F. Faniyi, R. Bahsoon, P. R. Lewis, X. Yao, L. L. Minku, *et al.*, "The Handbook of Engineering Self-Aware and Self-Expressive Systems," 2014.
- [19] S. Kounev, N. Huber, F. Brosig, and X. Zhu, "Model-Based Approach to Designing Self-Aware IT Systems and Infrastructures," *IEEE Computer Magazine*, 2016.
- [20] T. Becker, A. Agne, P. R. Lewis, R. Bahsoon, F. Faniyi, L. Esterle, *et al.*, "EPiCS: Engineering proprioception in computing systems," in *Computational Science and Engineering (CSE), 2012 IEEE 15th International Conference on*, 2012, pp. 353-360.
- [21] A. Elhabbash, M. Salama, R. Bahsoon, and P. Tino, "Self-Awareness in Software Engineering: A Systematic Literature Review," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, under review., 2017.

- [22] K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee, "A Design Science Research Methodology for Information Systems Research," *J. Manage. Inf. Syst.*, vol. 24, pp. 45-77, 2007.
- [23] D. Donjerkovic, Y. Ioannidis, and R. Ramakrishnan, "Dynamic Histograms: Capturing Evolving Data Sets," in *Data Engineering, 2000. Proceedings. 16th International Conference on*, 2000, pp. 86-86.
- [24] A. Elhabbash, R. Bahsoon, P. Tino, and P. Lewis, "Symbiotic-based Meta-self-awareness for Self-adaptive Systems: A Case for Volunteer Services," *Submitted to the IEEE Transaction on Software Engineering*, 2017.
- [25] A. Elhabbash, R. Bahsoon, and P. Tino, "Self-awareness for dynamic knowledge management in self-adaptive volunteer services," in *The 24th IEEE International Conference on Web Services (ICWS 2017)*, to appear, 2017.
- [26] A. Elhabbash, R. Bahsoon, and P. Tino, "Interaction-awareness for Volunteer Service Composition," presented at the The 10th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2016), Augsburg, Germany, 2016.
- [27] A. Elhabbash, R. Bahsoon, P. Tino, and P. R. Lewis, "Self-Adaptive Volunteered Services Composition through Stimulus-and Time-Awareness," in *Web Services (ICWS), 2015 IEEE International Conference on*, 2015, pp. 57-64.
- [28] A. Elhabbash, R. Bahsoon, P. Tino, and P. R. Lewis, "A Utility Model for Volunteered Service Composition," presented at the Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, 2014.
- [29] A. Elhabbash, R. Bahsoon, and P. Tino, "Towards Self-Aware Service Composition," in *High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICSS), 2014 IEEE Intl Conf on*, 2014, pp. 1275-1279.
- [30] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering – A systematic literature review," *Information and Software Technology*, vol. 51, pp. 7-15, 2009.
- [31] M. Salehie and L. Tahvildari, "Towards a Goal-driven Approach to Action Selection in Self-adaptive Software," *Softw. Pract. Exper.*, vol. 42, pp. 211-233, February 2012.
- [32] J. C'amara, G. A. Moreno, and D. Garlan, "Reasoning About Human Participation in Self-adaptive Systems," in *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ed. Florence, Italy: IEEE Press, 2015, pp. 146-156.

- [33] 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.
- [34] T. Patikirikorala, A. Colman, J. Han, and L. Wang, "A systematic survey on the design of self-adaptive software systems using control engineering approaches," in *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, ed, 2012, pp. 33-42.
- [35] S. Dobson, S. Denazis, A. Fern'andez, D. Ga'iti, E. Gelenbe, F. Massacci, *et al.*, "A Survey of Autonomic Communications," *ACM Trans. Auton. Adapt. Syst.*, vol. 1, pp. 223-259, December 2006.
- [36] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, pp. 46-54, 2004.
- [37] J. Kramer and J. Magee, "Self-Managed Systems: An Architectural Challenge," in *2007 Future of Software Engineering*, ed. Washington, DC, USA: IEEE Computer Society, 2007, pp. 259-268.
- [38] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, and A. Agarwal, "Seec: A framework for self-aware computing," Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA2010.
- [39] 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*, ed. Santa Fe, New Mexico, USA: ACM, 2010, pp. 7-16.
- [40] S. Dustdar, C. Dorn, F. Li, L. Baresi, G. Cabri, C. Pautasso, *et al.*, "A Roadmap Towards Sustainable Self-aware Service Systems," in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, ed. Cape Town, South Africa: ACM, 2010, pp. 10-19.
- [41] F. O. Faniyi, "Self-aware software architecture style and patterns for cloud-based applications," ed: University of Birmingham, 2015.
- [42] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," Keele University, Report2007.
- [43] P. Bozzelli, Q. Gu, and P. Lago, "A systematic literature review on green software metrics," VU University Amsterdam, Department of Computer. Science, The Netherlands, Report2013.
- [44] D. Weyns, M. U. Iftikhar, S. Malek, and J. Andersson, "Claims and supporting evidence for self-adaptive systems: a literature study," in *7th International*

*Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, ed: IEEE Press, 2012, pp. 89-98.

- [45] O. P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *Journal of Systems and Software*, vol. 80, pp. 571-583, 2007.
- [46] T. Dyba, T. Dingsoyr, and G. Hanssen, "Applying Systematic Reviews to Diverse Study Types: An Experience Report," in *1st International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ed, 2007, pp. 225-234.
- [47] T. Greenhalgh and R. Peacock, "Effectiveness and efficiency of search methods in systematic reviews of complex evidence: audit of primary sources," *BMJ*, vol. 331, pp. 1064-1065, 2005.
- [48] T. Dyba and T. Dingsoyr, "Empirical studies of agile software development: A systematic review," *Information and Software Technology*, vol. 50, pp. 833-859, 2008.
- [49] D. S. Cruzes and T. Dyba, "Recommended Steps for Thematic Synthesis in Software Engineering," in *International Symposium on Empirical Software Engineering and Measurement*, ed, 2011, pp. 275-284.
- [50] JabRef Development Team, "JabRef," ed, 2016.
- [51] A. Bronstein, J. Das, M. Duro, R. Friedrich, G. Kleyner, M. Mueller, *et al.*, "Self-aware services: using Bayesian networks for detecting anomalies in Internet-based services," in *Proceedings of IEEE/IFIP International Symposium on Integrated Network Management*, ed, 2001, pp. 623-638.
- [52] P. Andras and B. G. Charlton, "Self-aware software - Will it become a reality?," in *Self-Star Properties In Complex Information Systems: Conceptual and Practical Foundations*, 2005, pp. 229-259.
- [53] M. Mitchell, "Self-awareness and control in decentralized systems.," in *AAAI Spring Symposium: Metacognition in Computation*, ed, 2005, pp. 80-85.
- [54] R. Abbott and C. Sun, "Abstraction Abstracted," in *Proceedings of the 2nd International Workshop on The Role of Abstraction in Software Engineering (ROA)*, ed. New York, NY, USA: ACM, 2008, pp. 23-30.
- [55] M. D. Santambrogio, H. Hoffmann, J. Eastep, and A. Agarwal, "Enabling technologies for self-aware adaptive systems," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, ed, 2010, pp. 149-156.
- [56] I. Breskovic, C. Haas, S. Caton, and I. Brandic, "Towards Self-Awareness in Cloud Markets: A Monitoring Methodology," in *IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC)*, ed, 2011, pp. 81-88.

- [57] F. Faniyi and R. Bahsoon, "Engineering Proprioception in SLA Management for Cloud Architectures," in *9th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, ed, 2011, pp. 336-340.
- [58] F. Zambonelli, N. Biccocchi, G. Cabri, L. Leonardi, and M. Puviani, "On Self-Adaptation, Self-Expression, and Self-Awareness in Autonomic Service Component Ensembles," in *5th IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW)*, ed, 2011, pp. 108-113.
- [59] E. Vassev and M. Hinchey, "Awareness in Software-Intensive Systems," *IEEE Computer*, vol. 45, pp. 84-87, 2012.
- [60] D. B. Abeywickrama, F. Zambonelli, and N. Hoch, "Towards Simulating Architectural Patterns for Self-Aware and Self-Adaptive Systems," in *IEEE 6th International Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW)*, ed, 2012, pp. 133-138.
- [61] F. Faniyi, P. R. Lewis, R. Bahsoon, and X. Yao, "Architecting Self-Aware Software Systems," in *IEEE/IFIP Conference on Software Architecture (WICSA)*, ed, 2014, pp. 91-94.
- [62] E. Vassev and M. Hinchey, "Knowledge Representation for Adaptive and Self-aware Systems," in *Software Engineering for Collective Autonomic Systems: The ASCENS Approach*, 2015, pp. 221-247.
- [63] M. Hölzl and T. Gabor, "Reasoning and Learning for Awareness and Adaptation," in *Software Engineering for Collective Autonomic Systems: The ASCENS Approach*, 2015, pp. 249-290.
- [64] P. R. Lewis, A. Chandra, S. Parsons, E. Robinson, K. Glette, R. Bahsoon, *et al.*, "A Survey of Self-Awareness and Its Application in Computing Systems," in *5th IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW)*, ed, 2011, pp. 102-107.
- [65] S. Kounev, F. Brosig, and N. Huber, "The Descartes Modeling Language," Technical report, Department of Computer Science, University of Wuerzburg 2014.
- [66] T. Becker, A. Agne, P. R. Lewis, R. Bahsoon, F. Faniyi, L. Esterle, *et al.*, "EPiCS: Engineering Proprioception in Computing Systems," in *IEEE 15th International Conference on Computational Science and Engineering (CSE)*, ed, 2012, pp. 353-360.
- [67] R. De Nicola, M. Loreti, R. Pugliese, and F. Tiezzi, "A Formal Approach to Autonomic Systems Programming: The SCEL Language," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 9, pp. 7:1-7:29, 2014.
- [68] T. Chen and R. Bahsoon, "Toward a Smarter Cloud: Self-Aware Autoscaling of Cloud Configurations and Resources," *IEEE Computer*, vol. 48, pp. 93-96, 2015.

- [69] R. Sterritt, "Autonomic computing," *Innovations in Systems and Software Engineering*, vol. 1, pp. 79-88, 2005.
- [70] D. B. Abeywickrama, N. Hoch, and F. Zambonelli, "SimSOTA: Engineering and Simulating Feedback Loops for Self-adaptive Systems," in *Proceedings of the International C\* Conference on Computer Science and Software Engineering (C3S2E)*, ed. Porto, Portugal: ACM, 2013, pp. 67-76.
- [71] N. Serbedzija, T. Bures, and J. Keznikl, "Engineering Autonomous Systems," in *Proceedings of the 17th Panhellenic Conference on Informatics (PCI)*, ed. Thessaloniki, Greece: ACM, 2013, pp. 128-135.
- [72] D. Dannenhauer, M. T. Cox, S. Gupta, M. Paisner, and D. Perlis, "Toward Meta-level Control of Autonomous Agents," *Procedia Computer Science*, vol. 41, pp. 226 - 232, 2014.
- [73] R. Gioiosa, G. Kestor, D. J. Kerbyson, and A. Hoisie, "Cross-Layer Self-Adaptive/Self-Aware System Software for Exascale Systems," in *IEEE 26th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, ed, 2014, pp. 326-333.
- [74] E. Riccobene and P. Scandurra, "Formal Modeling Self-adaptive Service-oriented Applications," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC)*, ed. Salamanca, Spain: ACM, 2015, pp. 1704-1710.
- [75] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, and A. Agarwal, "A generalized software framework for accurate and efficient management of performance goals," in *Proceedings of the International Conference on Embedded Software (EMSOFT)*, ed, 2013, pp. 1-10.
- [76] E. E. Veas, K. Kiyokawa, and H. Takemura, "Self-aware Framework for Adaptive Augmented Reality," in *Proceedings of the International Conference on Augmented Tele-existence*, ed. Christchurch, New Zealand: ACM, 2005, pp. 70-77.
- [77] N. Bicocchi, D. Fontana, and F. Zambonelli, "A self-aware, reconfigurable architecture for context awareness," in *IEEE Symposium on Computers and Communications (ISCC)*, ed, 2014, pp. 1-7.
- [78] M. Salama and R. Bahsoon, "Quality-Driven Architectural Patterns for Self-Aware Cloud-Based Software," in *IEEE 8th International Conference on Cloud Computing (IEEE CLOUD)*, ed, 2015, pp. 844-851.
- [79] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, and A. Agarwal, "SEEC: a general and extensible framework for self-aware computing," Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA2011.
- [80] E. Vassev and M. Hinchey, "Knowledge Representation and Reasoning for Intelligent Software Systems," *IEEE Computer*, vol. 44, pp. 96-99, 2011.

- [81] C. H. Huang, J. S. Shen, and P. A. Hsiung, "A Self-Adaptive Hardware/Software System Architecture for Ubiquitous Computing Applications," in *Ubiquitous Intelligence And Computing* vol. 6406, ed, 2010, pp. 382-396.
- [82] F. Zambonelli, G. Castelli, L. Ferrari, M. Mamei, A. Rosi, G. D. Marzo, *et al.*, "Self-aware Pervasive Service Ecosystems," *Procedia Computer Science*, vol. 7, pp. 197 - 199, 2011.
- [83] G. Castelli, M. Mamei, A. Rosi, and F. Zambonelli, "Engineering Pervasive Service Ecosystems: The SAPERE Approach," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 10, pp. 1:1-1:27, 2015.
- [84] Y. Su, F. Shi, S. Talpur, Y. Wang, S. Hu, and J. Wei, "Achieving self-aware parallelism in stream programs," *Cluster Computing-The Journal Of Networks Software Tools And Applications*, vol. 18, pp. 949-962, 2015.
- [85] A. Griffiths, "ldquo;self rdquo;-conscious objects in Object-Z," in *Proceedings of Technology of Object-Oriented Languages and Systems (TOOLS)*, ed, 1997, pp. 210-224.
- [86] A. G. Beltran, P. Milligan, and P. Sage, "Heterogeneity-aware distributed access structure," in *Fifth IEEE International Conference on Peer-to-Peer Computing (P2P)*, ed, 2005, pp. 152-153.
- [87] O. Nierstrasz, M. Denker, T. Gîrba, A. Lienhard, and D. Röthlisberger, "Change-Enabled Software Systems," in *Software-Intensive Systems and New Computing Paradigms: Challenges and Visions*, Berlin, Heidelberg, 2008, pp. 64-79.
- [88] S. Kounev, F. Brosig, N. Huber, and R. Reussner, "Towards Self-Aware Performance and Resource Management in Modern Service-Oriented Systems," in *Proceedings of IEEE International Conference on Services Computing (SCC)*, ed, 2010, pp. 621-624.
- [89] S. Kounev, F. Brosig, and N. Huber, "Self-aware QoS Management in Virtualized Infrastructures," in *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC)*, ed. Karlsruhe, Germany: ACM, 2011, pp. 175-176.
- [90] A. Egyed, "Architecture Differencing for Self Management," in *Proceedings of the 1st ACM SIGSOFT Workshop on Self-managed Systems (WOSS)*, ed. Newport Beach, California: ACM, 2004, pp. 44-48.
- [91] R. B. France and B. Rumpe, "Model-driven Development of Complex Software: A Research Roadmap," in *Future of Software Engineering, 2007. FOSE '07*, ed, 2007, pp. 37-54.
- [92] G. Blair, N. Bencomo, and R. B. France, "Models@ run.time," *Computer*, vol. 42, pp. 22-27, 2009.

- [93] E. Riccobene, P. Scandurra, and F. Albani, "A Modeling and Executable Language for Designing and Prototyping Service-Oriented Applications," in *37th EUROMICRO Conference on Software Engineering and Advanced Applications*, ed. 2011, pp. 4-11.
- [94] D. Garlan and M. Shaw, "An introduction to software architecture," *Advances in software engineering and knowledge engineering*, vol. 1, 1993.
- [95] M. C. Huebscher and J. A. McCann, "A Survey of Autonomic Computing — Degrees, Models, and Applications," *ACM Computing Surveys*, vol. 40, pp. 7:1-7:28, 2008.
- [96] E. Vassev, M. Hinchey, and B. Gaudin, "Knowledge Representation for Self-adaptive Behavior," in *Proceedings of the Fifth International C\* Conference on Computer Science and Software Engineering*, ed. Montreal, Quebec, Canada: ACM, 2012, pp. 113-117.
- [97] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, pp. 263-277, 2007.
- [98] J. Camara, R. de Lemos, C. Ghezzi, and A. Lopes, *Assurances for self-adaptive systems : principles, models, and techniques*. Berlin New York: Springer, 2013.
- [99] E. Yuan, N. Esfahani, and S. Malek, "A Systematic Survey of Self-Protecting Software Systems," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 8, pp. 1-41, 2014.
- [100] 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*, 2010, pp. 741-750.
- [101] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, 2004, pp. 4-10.
- [102] G. Fedak, C. Germain, V. Neri, and F. Cappello, "Xtremweb: A generic global computing system," in *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, 2001, pp. 582-587.
- [103] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@ home: an experiment in public-resource computing," *Communications of the ACM*, vol. 45, pp. 56-61, 2002.
- [104] A. L. Beberg and V. S. Pande, "Storage@ home: Petascale distributed storage," in *2007 IEEE International Parallel and Distributed Processing Symposium*, 2007, p. 482.
- [105] A. L. Beberg, D. L. Ensign, G. Jayachandran, S. Khaliq, and V. S. Pande, "Folding@ home: Lessons from eight years of volunteer distributed computing," in *Parallel &*

- Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on, 2009*, pp. 1-8.
- [106] A. Cuomo, G. D. Modica, S. Distefano, M. Rak, and A. Vecchio, "The Cloud@Home Architecture - Building a Cloud Infrastructure from Volunteered Resources," presented at the Proceedings of the 1st International Conference on Cloud Computing and Services Science, Netherlands, 2011.
- [107] A. Chandra and J. Weissman, "Nebulas: using distributed voluntary resources to build clouds," presented at the Proceedings of the 2009 conference on Hot topics in cloud computing, San Diego, California, 2009.
- [108] K. Chard, K. Bubendorfer, S. Caton, and O. F. Rana, "Social Cloud Computing: A Vision for Socially Motivated Resource Sharing," *Services Computing, IEEE Transactions on*, vol. 5, pp. 551-563, 2012.
- [109] S. Sebastio, M. Amoretti, and A. L. Lafuente, "AVOCLOUDY: a simulator of volunteer clouds," *Software: Practice and Experience*, 2015.
- [110] J. R. Douceur, "Is remote host availability governed by a universal law?," *SIGMETRICS Perform. Eval. Rev.*, vol. 31, pp. 25-29, 2003.
- [111] D. Kondo, A. Andrzejak, and D. P. Anderson, "On correlated availability in internet-distributed systems," in *Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing*, 2008, pp. 276-283.
- [112] B. Javadi, D. Kondo, J. M. Vincent, and D. P. Anderson, "Discovering Statistical Models of Availability in Large Distributed Systems: An Empirical Study of SETI@home," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, pp. 1896-1903, 2011.
- [113] M. Bakkaloglu, J. J. Wylie, C. Wang, and G. R. Ganger, "On correlated failures in survivable storage systems," DTIC Document2002.
- [114] B. Abbott, R. Abbott, R. Adhikari, P. Ajith, B. Allen, G. Allen, *et al.*, "Einstein@ Home search for periodic gravitational waves in LIGO S4 data," *Physical Review D*, vol. 79, p. 022001, 2009.
- [115] S. Siva Sathya and K. Syam Babu, "Survey of fault tolerant techniques for grid," *Computer Science Review*, vol. 4, pp. 101-120, 2010.
- [116] V. D. 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, 2009*, pp. 134-139.
- [117] V. D. Cunsolo, S. Distefano, A. Puliafito, and M. Scarpa, "Applying Software Engineering Principles for Designing Cloud@Home," in *Cluster, Cloud and Grid*

- Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, 2010, pp. 618-624.
- [118] S. Distefano, M. Fazio, and A. Puliafito, "The Cloud@Home Resource Management System," in *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, 2011, pp. 122-129.
- [119] S. Distefano, A. Puliafito, M. Rak, S. Venticinque, U. Villano, A. Cuomo, *et al.*, "QoS Management in Cloud@Home Infrastructures," in *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2011 International Conference on*, 2011, pp. 190-197.
- [120] R. Aversa, B. Di Martino, N. Mazzocca, and S. Venticinque, "MAGDA: A Mobile Agent based Grid Architecture," *Journal of Grid Computing*, vol. 4, pp. 395-412, 2006/12/01 2006.
- [121] M. Ryden, A. Chandra, and J. Weissman, "Nebula: Data Intensive Computing over Widely Distributed Voluntary Resources," Dept. of Computer Science and Eng., Univ. of Minnesota tech. report TR 13-007, 2013.
- [122] S. Kannan, A. Gavrilovska, and K. Schwan, "Cloud4Home -- Enhancing Data Services with @Home Clouds," presented at the Proceedings of the 2011 31st International Conference on Distributed Computing Systems, 2011.
- [123] K. Chard, S. Caton, O. Rana, and K. Bubendorfer, "Social Cloud: Cloud Computing in Social Networks," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, 2010, pp. 99-106.
- [124] A. Jula, E. Sundararajan, and Z. Othman, "Cloud computing service composition: A systematic literature review," *Expert Systems with Applications*, vol. 41, pp. 3809-3824, 2014.
- [125] U. K. C. O. E. a. R. Group. (2013, 14/02/2017). *End User Device Strategy: Security Framework & Controls*. Available: [https://www.gov.uk/government/uploads/system/uploads/attachment\\_data/file/261980/EUD\\_Security.pdf](https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/261980/EUD_Security.pdf)
- [126] D. G. Kendall, "Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded Markov chain," *The Annals of Mathematical Statistics*, pp. 338-354, 1953.
- [127] D. Kondo, D. P. Anderson, and J. M. Vii, "Performance evaluation of scheduling policies for volunteer computing," in *e-Science and Grid Computing, IEEE International Conference on*, 2007, pp. 415-422.
- [128] D. P. Anderson, "Emulating volunteer computing scheduling policies," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, 2011, pp. 1839-1846.

- [129] H. Ehtamo, R. P. Hämäläinen, P. Heiskanen, J. Teich, M. Verkama, and S. Zionts, "Generating pareto solutions in a two-party setting: constraint proposal methods," *Management Science*, vol. 45, pp. 1697-1709, 1999.
- [130] B. Donassolo, H. Casanova, A. Legrand, and P. Velho, "Fast and scalable simulation of volunteer computing systems using simgrid," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, 2010, pp. 605-612.
- [131] U. o. C. a. B.-B. group. *Open-source software for volunteer computing*. Available: <https://boinc.berkeley.edu/>
- [132] *OpenNebula home page*. Available: <http://www.opennebula.org/>
- [133] D. Nurmi, R. Wolski, C. Grzegorzczuk, G. Obertelli, S. Soman, L. Youseff, *et al.*, "The Eucalyptus Open-Source Cloud-Computing System," in *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, 2009, pp. 124-131.
- [134] H. Casanova, A. Legrand, and M. Quinson, "SimGrid: A Generic Framework for Large-Scale Distributed Experiments," in *Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on*, 2008, pp. 126-131.
- [135] R. Buyya and M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency and computation: practice and experience*, vol. 14, pp. 1175-1220, 2002.
- [136] S. Kounev, F. Brosig, N. Huber, and R. Reussner, "Towards Self-Aware Performance and Resource Management in Modern Service-Oriented Systems," in *Services Computing (SCC), 2010 IEEE International Conference on*, 2010, pp. 621-624.
- [137] Microsoft. (14/02/2017). *Microsoft Baseline Security Analyzer*. Available: <https://msdn.microsoft.com/en-us/library/ff647642.aspx>
- [138] (15/02/2117). *WHAT IS Self-Awareness in Autonomic Systems?* Available: [http://cordis.europa.eu/fp7/ict/fet-proactive/aware\\_en.html](http://cordis.europa.eu/fp7/ict/fet-proactive/aware_en.html)
- [139] M. Wirsing, M. Hölzl, M. Tribastone, and F. Zambonelli, "ASCENS: Engineering Autonomic Service-Component Ensembles," in *Formal Methods for Components and Objects: 10th International Symposium, FMCO 2011, Turin, Italy, October 3-5, 2011, Revised Selected Papers*, B. Beckert, F. Damiani, F. S. de Boer, and M. M. Bonsangue, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1-24.
- [140] F. Zambonelli, G. Castelli, L. Ferrari, M. Mamei, A. Rosi, G. D. Marzo, *et al.*, "Self-aware Pervasive Service Ecosystems," *Procedia Computer Science*, vol. 7, pp. 197-199, 2011.

- [141] V. Poosala, P. J. Haas, Y. E. Ioannidis, and E. J. Shekita, "Improved histograms for selectivity estimation of range predicates," *SIGMOD Rec.*, vol. 25, pp. 294-305, 1996.
- [142] H. Mousavi and C. Zaniolo, "Fast and accurate computation of equi-depth histograms over data streams," presented at the Proceedings of the 14th International Conference on Extending Database Technology, Uppsala, Sweden, 2011.
- [143] C. Siang Yew, P. Tino, and Y. Xin, "Measuring Generalization Performance in Coevolutionary Learning," *Evolutionary Computation, IEEE Transactions on*, vol. 12, pp. 479-505, 2008.
- [144] R. Fujimoto, W. H. Lunceford, E. H. Page, and A. M. U. (editors), "Grand challenges for modeling and simulation: Dagstuhl report," Technical report 350, 2002.
- [145] O. O. Onolaja, "Dynamic data-driven framework for reputation management," Ph.D., School of Computer Science, University of Birmingham, 2012.
- [146] H. Aydt, S. J. Turner, W. Cai, and M. Y. H. Low, "Symbiotic Simulation Systems: An Extended Definition Motivated by Symbiosis in Biology," in *2008 22nd Workshop on Principles of Advanced and Distributed Simulation*, 2008, pp. 109-116.
- [147] H. Aydt, S. J. Turner, W. Cai, and M. Y. H. Low, "Research issues in symbiotic simulation," in *Proceedings of the 2009 Winter Simulation Conference (WSC)*, 2009, pp. 1213-1222.
- [148] B. Tjahjono and X. Jiang, "Linking symbiotic simulation to enterprise systems: framework and applications," presented at the Proceedings of the 2015 Winter Simulation Conference, Huntington Beach, California, 2015.
- [149] V.-A. Vu, G. Park, G. Tan, and M. Ben-Akiva, "A simulation-based framework for the generation and evaluation of traffic management strategies," presented at the Proceedings of the 2014 Annual Simulation Symposium, Tampa, Florida, 2014.
- [150] S. Abar, P. Lemarinier, G. K. Theodoropoulos, and G. M. P. OHare, "Automated Dynamic Resource Provisioning and Monitoring in Virtualized Large-Scale Datacenter," in *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*, 2014, pp. 961-970.
- [151] F. Darema, "Dynamic data driven applications systems: A new paradigm for application simulations and measurements," in *International Conference on Computational Science*, 2004, pp. 662-669.
- [152] D. Ardagna and B. Pernici, "Adaptive Service Composition in Flexible Processes," *IEEE Transactions on Software Engineering*, vol. 33, pp. 369-384, 2007.

- [153] C.-L. 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.
- [154] R. M. Loynes, "The stability of a queue with non-independent inter-arrival and service times," in *Mathematical Proceedings of the Cambridge Philosophical Society*, 1962, pp. 497-520.
- [155] D. Lixin, Z. Sanyou, and K. Lishan, "A fast algorithm on finding the non-dominated set in multi-objective optimization," in *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, 2003, pp. 2565-2571 Vol.4.
- [156] O. a. o. i. affiliates. (2013). *The Java EE 6 Tutorial - Web Services*. Available: <http://docs.oracle.com/javaee/6/tutorial/doc/giqsx.html#gkcaw>
- [157] O. a. o. i. affiliates. (2013). *The Java EE 6 Tutorial - Java Message Service Concepts*. Available: <http://docs.oracle.com/javaee/6/tutorial/doc/bnceh.html>
- [158] P. G. D. Group. *PostgreSQL* Available: <https://www.postgresql.org/>
- [159] O. C. a. o. i. affiliates. *GlassFish*. Available: <https://glassfish.java.net/>