POLITECNICO DI MILANO
SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

TESI DI LAUREA MAGISTRALE IN
COMPUTER SCIENCE AND ENGINEERING

TITLE

Author:
Paolo Paterna

Student ID (Matricola):
852548

Supervisor (Relatore):
Prof. Raffaela Mirandola

Co-Supervisor (Correlatore):
Ph.D. Diego Perez-Palacin

A.Y. 2017/2018

Contents

List of Figures

List of Tables

Acknowledgments

Abstract (Italian version)

Abstract

CHAPTER 1

Introduction

CHAPTER 2

State of the Art

2.1 Self-Adaptive Software Systems

In modern-day applications, software complexity has extremely increased thanks to the spread of highly available and faster wireless connection such as in the Internet of Things (IoT) ambit. Since software is often deployed in dynamic contexts, where requirements, environment assumptions and usage profiles varies continuously, software complexity increased over time to the point where it is often composed by a number of sub-components and/or sub-services that work together in order to offer a service to the users. This is the case of service-oriented applications – also called Service Based Systems (SBS) – that are composed by multiple *services* and *components*. In these systems, services offered by third-party providers are dynamically composed into workflows to deliver complex functionalities, so SBSs rely on self adaptation to cope with the uncertainties associated with third-party services as the loose coupling of services makes a reconfiguration feasible. Without adaptation, the application is prone to degraded performance because of faulty components, messages lost between services or delays due to an increasing number of users.

During the past decade a lot of research has been made in this scope but the engineering of adaptive systems remains a incredible challenge.[1] In order to

solve the problem, **Self-Adapting Software Systems (SASS)** are born. These are flexible systems that can adapt themselves to their contextual needs and can do so with the highest performance and availability. General discussion concerning the issue and the state of the art in the design and implementation have been presented.[1][2][3][4][5][6][7]

These kind of systems have some fundamental properties called auto-managing that are:

- Auto-configuration
- Auto-recovery in case of failure
- Auto-optimization
- Auto-protection

All these properties can be grouped in two more abstract concepts which are self-awareness and context-awareness.

Self-Awareness is the ability of the system to be able to monitor itself in terms of available resources and behavior.

Context-Awareness is the ability of the system to understand the environment where it is working, using the information provided by its components, and adapt itself to all the changes that can occur during its normal operational status. To better understand how a SASS works we need to answer some simple questions:

- Who is adapting?
- Which adaptation is required?
- When is necessary to adapt?
- Where is needed to change something
- Why is needed an adaptation?
- How we achieve this goal?

During the past years have been developed some dimensions that help to answer all this simple questions: *Time*, *Reason*, *Level*, *Technique* and *Adaptation Control* shown in figure 2.1.

Who is adapting? As the name suggests, it's the system itself that changes something in order to preserve some given constraint.

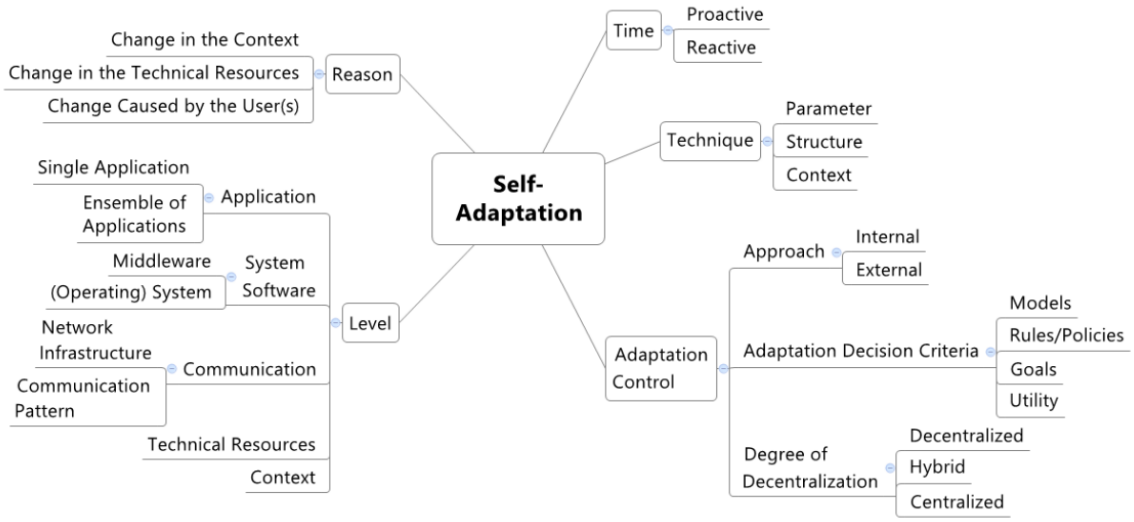

Figure 2.1: *The Dimensions to analyze adaptation.[8]*

Which adaptation is required? The *Technique* dimension is the one that answers this question in fact the software engineer can change either the parameters or the system can be considered as a set of components. The former case allows to fine tuning the system at the expense of an higher complexity, the latter is called composite vision and permits the systems to cooperate exchanging algorithms and much more important, reusing components which improve performance because failed or defected components can be replaced.

When is necessary to adapt? The *Time* dimension is crucial in this situation. There are three typical approaches: the reactive one is the more traditional one which states that an adaptation is needed only after a causative event. The other two approaches are more interesting and they are predictive and proactive. The former studies the system before any event and calculate the need of an adaptation, the latter applies and adaptation despite an event and improves the performance. From the user perspective the proactive approach is the best because it doesn't interrupt the operation of the system in any load but it is the more complicated to implement. Monitoring continuously the system is a costly task to do, on the other side an adaptive monitoring is simple that analyze only specific aspect and/or resources and intervenes only if needed.

2.1.1 The SOLAR Framework

However working on the adaptability of a system can impact other quality attributes such as performance, reliability or maintainability and in the worst case improving adaptability can decrease part, if not all, of these attributes as stated

in [9]: *quality attributes can never be achieved in isolation, the achievement of any one will have an effect, sometimes positive and sometimes negative, on the achievement of others*.

Find a balance between these quality attributes is often a challenging task because sometimes they're conflicting each other, e.g. lower cost and higher availability, so find an adaptability value that can meet all the requisites is, as a consequence, a challenging task too.

Bibliography

[1] R. de Lemos et al. *Software engineering for self-adaptive systems: A second research roadmap*, volume 7475 of *Lecture Notes in Computer Science*. Springer, 2013.

[2] M.C. Huebscher and J.A. McCann. A survey of autonomic computing – degrees, models, and applications. *ACM Comput. Surv. 40*, 3, 2008.

[3] M. Salehie and L. Tahvildari. Self-adaptive software: landscape and research challenges. *ACM Trans. Auton. Adapt. Syst. 4*, 2:1–42, 2009.

[4] B.H Cheng et al. *Software Engineering for Self-Adaptive Systems: A Research Roadmap*, volume 5525. Springer, 2009.

[5] J. Andersson, R. de Lemos, S. Malek, and D. Weyns. Modeling dimensions of self-adaptive software systems. In *Software engineering for self-adaptive systems*, volume 5525, pages 27–47. Springer, 2009.

[6] P. Oreizy, M.M. Gorlick, R.N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D.S. Rosenblum, and A.L. Wolf. An architecture-based approach to self-adaptive software. *IEEE Intell. Syst. 14*, 3:54–62, 1999.

[7] R. Calinescu, C. Ghezzi, M.Z. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Commun. ACM 55*, 9:69–77, 2012.

[8] Felix Maximilian Roth, Christian Krupitzer, et al. A survey on engineering approaches for self-adaptive systems. In *Pervasive and Mobile Computing*, volume 17, pages 184–206. Elsevier, 2015.

[9] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice, 2nd edn. SEI Series in software engineering*. Addison-Wesley Pearson Education, Boston, 2003.