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.

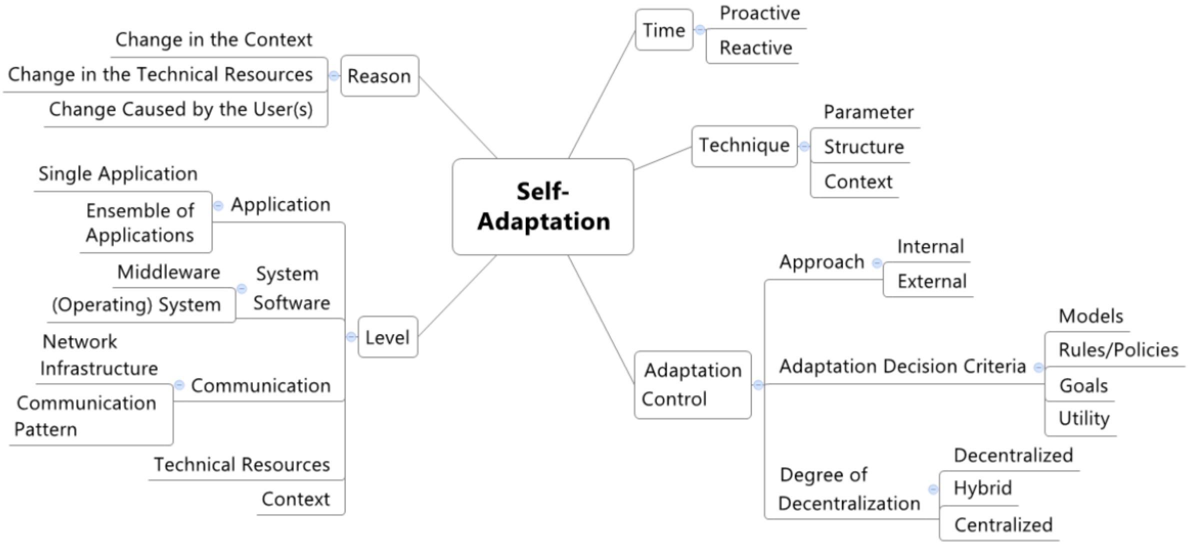

Figure 2.1: *The Dimensions to analyze adaptation.[8]*

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.

Where is needed to change something? In general a SASS is composed by two main part: the the Adaptability Logic (AL) and the Managed Resource (MR). The former in general doesn't change, the latter is composed at the base of the hardware and of the software such as the operating system or, in case of distributed systems, the middleware that control the hardware; at a higher level of the application. These are the parts that require adaptation. To answer this question is needed to decide at which level the operation has to be applied without neglecting the relationship between the MR and the AL which is composed by the network that connects them and/or the view of the communication patterns. Thus *Level* is the considered dimension.

Why is needed an adaptation? In this case, *Reason* is the right dimension. There can be one or more reason because a system needs adaptation such as a change in the available resources, a change in the environment or a change in the user base of the system.

How we achieve this goal? The answer to this question is more complicated than the others because it needs a new topic called *Adaptation Control*.

2.1.1 Adaptation Control

In the literature can be found 2 approaches: the *internal approach* that intertwine the adaptation logic with the system resources, which has problems with the maintainability and scalability of the system, and the *external approach* that splits the system into adaptation logic and managed resources, which increases maintainability and scalability through modularization.

The control unit needs a metric in order to decide how to adapt and in literature are present different metrics: models, rules and policies, goal or utility functions [9].

Another aspect of the adaptation logic is the degree of decentralization. If we have a system with limited resources then a centralized adaptation logic has to be preferred but with greater systems a decentralized AL can improve performance and every sub-system can communicate with another with different patterns of communication. Of course hybrid technique can be made mixing the previous approaches.

Adaptation Logic Issues

As said before a SASS is composed of managed resources and the adaptation logic; it can be represented by the tuple `SASS = (AL, MR)`. *AL* = $a_1, \ldots, a_n$, with a_i representing a logic element, monitors the environment (M), analyzes the data for change (A), plans adaptation (P) and control the execution of the adaptation (E): these are known as *MAPE cycle* or *MAPE functionality*[10]. *MR* = $mr_1, \ldots, mr_n$, with mr_i representing a resource, is the set of resources such as hardware with software, smart-phones, robotics or unmanned vehicles. Figure 2.2 shows a SASS where the dashed line represent the system border. The di-

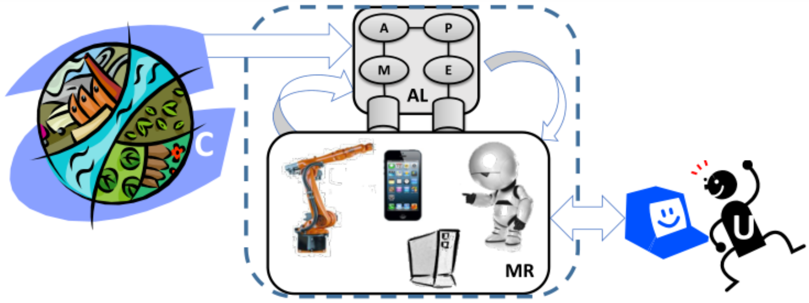

Figure 2.2: *A SASS (AL = Adaptation Logic, MR = Managed Resources, U = User(s), C = Context, M,A,P,E = MAPE functionality).[8]*

mension can therefore be mapped to the MAPE functionalities as shown in table 2.1.

	Time	Reason	Level	Technique
Monitoring	Continuos	What to monitor	Identification of the levels	—
Analyzing	Algorithms depend on reactive or proactive dimension	Where to analyze	—	—
Planning	—	What should be influenced by planning	Adaptation plans address these levels	Plans for performing the techniques
Executing	—	—	Execution of the change on the levels	Execution of the change on the levels

Table 2.1: *Relation of the MAPE Activities and the Dimensions*

2.2 Tele Assistance: A Self-Adaptive Service-Based System

Tele Assistance System, also known as TAS, is a research, done by the University of York [11], on self-adaptation in the domain of service-based systems. Originally introduced in [12] has already been used in the evaluation of several self-adaptation solutions [12], [7], [13], [14], [15], albeit based on ad-hoc implementations, scenarios and evaluation metrics that make the comparison of these solutions and its use to evaluate other solution very difficult.

To address these limitation the University of York implemented TAS on their Research Service Platform (ReSeP) in conjunction with concrete scenarios in order to have an immediate use in evaluation of the self-adaptation solutions.

The system provides health support to chronic condition sufferers within the comfort of their homes. TAS uses a combination of sensors embedded in a wearable device and remote services from healthcare, pharmacy and emergency service providers. As shown in Figure 2.3, the TAS workflow takes periodical measurements of the vital parameters of a patient and employs a third-party medical service for their analysis. The analysis result may trigger the invocation of a pharmacy service to deliver new medication to the patient or to change his/her dose of medication, or the invocation of an alarm service leading, e.g., to an ambulance being dispatched to the patient. The same alarm service can be invoked directly by the patient, by using a panic button on the wearable device.

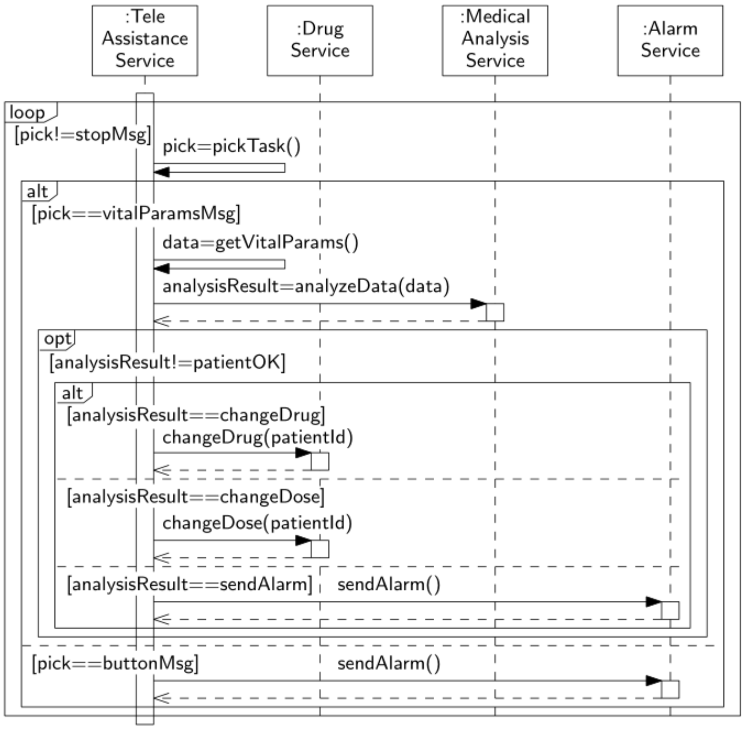

Figure 2.3: *TAS workflow*

They also device some generic adaptation scenarios shown in Table 2.2 and some metrics shown in table 2.3

In conclusion TAS is a reference implementation of a service based system and generic adaptation scenarios associated with different types of uncertainty. First, it aims to promote research and understanding among multiple researchers and research groups, through enabling the comparison of different self-adaptation approaches, without favouring any particular approach. Second, TAS aims to serve the advance of single research efforts by reducing the time required to evaluate self-adaptation solutions. Finally, it aims to contribute to advancing the practice of engineering self-adaptive systems, by being a realistic example of a widely used type of software system.

Scenario	Type of uncertainty	Type of adaptation	Type of requirements
S1	Unpredictable environment: service failure	Switch to equivalent service; Simultaneous invocation of several services for idempotent operation	QoS: Reliability, cost
S2	Unpredictable environment: variation of service response time	Switch to equivalent service; Simultaneous invocation of several services for idempotent operation	QoS: Performance, cost
S3	Incomplete information: new service	Use new service	QoS: Reliability, performance, cost
S4	Changing requirements: new goal	Change workflow architecture; Select new service of the change on the levels	Functional: new operation
S5	Inadequate design: wrong operation sequence	Change workflow architecture	Functional: operation sequence compliance

Table 2.2: *Generic adaptation scenarios for service-based systems*

Quality Attribute	Metrics
Reliability	Number of failed service invocations Number of specific operation sequence failures Mean time to recovery
Performance	Number of specific operation sequences exceeding allowed execution time
Cost	Cumulative service invocation cost over given time period
Functionalities	Number of faulty process executions

Table 2.3: *Quality attributes and metrics for the evaluation and comparison of SBS self-adaptation solutions*

2.3 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 [16]: *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.

The SOLAR (SOftware quaLities and Adaptability Relationships) framework [17] helps the software architect to select the best set of components in order to fulfill the requirements trying to achieve a minimum level for some quality attribute such as availability and/or cost. This tool helps the software architect to build a suitable architecture for his needs but is not a *"solution for every situation"*.

2.3.1 The SOLAR Metrics

All the metrics in SOLAR are greatly inspired by [18] and are all defined at an architecture level and static perspective.

To define them, the software architecture relies on a component-and-connector view (C&C view). In this C&C view *components* are principal computational elements present at runtime. The representation uses the UML diagram. In Figure 2.4 is shown an example of component and their respective connections.

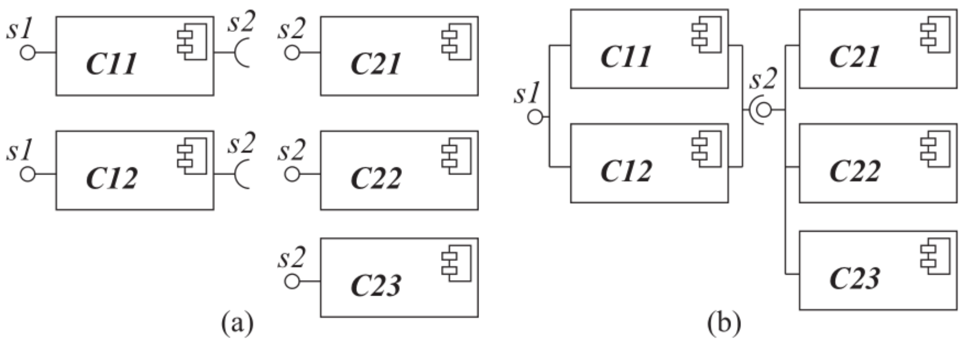

Figure 2.4: *(a) A set of components and their interfaces and (b) the C&C view of the components in (a).[17]*

Components have interfaces attached to ports. *Connectors* are pathways of interaction between components and also have interfaces or roles. In Figure 2.5 is shown an example of an architecture, the used components are highlighted in gray; this example will be used to explain the metrics later in this chapter.

In the architecture of Figure 2.5:

- the service S1 is provided by component C11 that is unique and must be in use. S1 is the provided service of this architecture.
- component C11 requires S2 and S3 services.

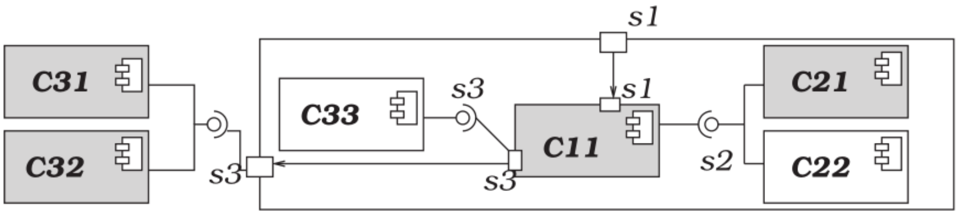

Figure 2.5: *C&C view: discovered components and used components (in gray).[17]*

- service S2 is provided by C21 and C22 where only C21 is in use.
- service S3 is provided by C31, C32 and C33 but only C31 and C32 are in use.

Absolute adaptability of a service (AAS)

This metric measures the number of used components for providing a given service.

$$AAS \in \mathbb{N}^n | AAS_i = |UC_i|$$

Quantifies how much adaptable a service is by counting the different alternatives to execute the service (1 no adaptable, >1 adaptable), where the service adaptability grows according to the number of components able to provide it.

Referring to the example in Figure 2.5, we observe that $AAS = [1, 1, 2]$.

Relative adaptability of a service (RAS)

This metric measures the number of used components that provide a given service with respect to the number of components actually offering such service.

$$RAS \in \mathbb{Q}^n | RAS_i = \frac{|UC_i|}{|C_i|}$$

It describes how each service stresses its adaptability choices and it informs how much more adaptable the service could be. RAS vector values near to one mean that the service is using almost all the adaptability potentially reachable.

Referring to the example in Figure 2.5, we observe that $RAS = [1, 0.5, 0.6]$.

Mean of absolute adaptability of services (MAAS)

This metric measures the mean number of used components per service.

$$MAAS \in \mathbb{Q} | MAAS = \frac{\sum_{i=1}^{n} AAS_i}{n}$$

This metric offers insights into the mean size and effort needed to manage each service.

Referring to the example in Figure 2.5, $MAAS = 4/3 = 1.3$.

Architectures with more adaptable services have higher values of MAAS. Besides, a $MAAS > 1$ means that the architecture includes adaptable services (at least one of the components $AAS_i > 1$). For $MAAS \leq 1$, there may be adaptable services or not (AAS should be checked in this case).

Mean of relative adaptability of services (MRAS)

This metric represents the mean of RAS.

$$MRAS \in \mathbb{Q} | MRAS = \frac{\sum_{i=1}^{n} RAS_i}{n}$$

This metric informs about the mean utilization of the potential components for each service. Values of this metric range between zero and one.

Referring to the example, $MRAS = (1 + 0.5 + 0.6)/3 = 0.72$.

The higher the MRAS of an architecture, the more adaptable its services are, on average. The maximum value of this metric is obtained when $RAS_i = 1 \ \ \forall i \in [1, \ldots, n]$, which is in turn obtained when all services are as much adaptable as possible because they use all the available components. Therefore, a value close to one for MRAS means that, on average, services are as much adaptable as possible. A value close to zero means that:

1. services can be much more adaptable (adding components not yet used)
2. different architecture alternatives with the same quantity of adaptability can be created

Level of system adaptability (LSA)

This metric measures the number of components used to make up the system with respect to the number of components that the most adaptable architecture would use.

$$LSA \in \mathbb{Q}0..1 | LSA = \frac{\sum_{i=1}^{n} AAS_i}{\sum_{i=1}^{n} |C_i|}$$

The value of this metric ranges between zero and one. For LSA, a value of one means that the system is using all existing components for each service, i.e., $AAS_i = |C_i| \forall i \in 1, \ldots, n$, and then its adaptability is already to the maximum. A value close to one means that the market offers few choices to increase the system architectural adaptability. When a new component is bounded to the architecture, LSA increases in a constant value $(1/\sum i = 1^n |C_i)$ irrespective of the number of components already considered for the same service.

Referring to the example in Figure 2.5, $LSA = 4/(1+2+3) = 0.6$.

Table 2.4 summarizes the five metrics and their values for the example in Figure 2.5.

Name	Range	Value	Example in Fig. 2.5
AAS	$\mathbb{N}^n$	$\lvert UC_i \rvert$	$[1, 1, 2]$
RAS	$\mathbb{Q}^n \in 0, \ldots, 1$	$\frac{\lvert UC_i \rvert}{\lvert C_i \rvert}$	$[1, 0.5, 0.6]$
MAAS	$\mathbb{Q}_+$	$\frac{\sum_{i=1}^{n} AAS_i}{n}$	1.3
MRAS	$\mathbb{Q}^n \in 0, \ldots, 1$	$\frac{\sum_{i=1}^{n} RAS_i}{n}$	0.72
LSA	$\mathbb{Q}^n \in 0, \ldots, 1$	$\frac{\sum_{i=1}^{n} AAS_i}{\sum_{i=1}^{n} \lvert C_i \rvert}$	0.6

Table 2.4: *Summary of the metrics.[17]*

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. 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] P. Lalanda, J. A. McCann, and A. Diaconescu. Autonomic computing, 2013.

[10] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer 36*, 1:41–50, 2003.

[11] Danny Weyns and Radu Calinescu. Tele assistance: A self-adaptive service-based system examplar.

[12] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini. Validation of web service compositions. *IET*, 1(6):219–232, 2007.

[13] R. Calinescu, Lars Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic qos management and optimization in service-based systems. *Software Engineering, IEEE Transactions on*, 37(3):387–409, 2011.

[14] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Model evolution by run-time parameter adaptation. In *International Conference on Software Engineering*, 2009.

[15] A. Filieri, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Conquering complexity via seamless integration of design-time and run-time verification. In *Conquering Complexity*. Springer, 2012.

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

[17] Diego Perez-Palacin, Raffaela Mirandola, and José Merseguer. On the relationships between qos and software adaptability at the architectural level. *The Journal of Systems and Software*, 87, 2014.

[18] N. Subramanian and L. Chung. Process-oriented metrics for software architecture adaptability. *IEEE Computer Society*, page 311, 2011.