The Journal of Systems and Software 213 (2024) 112051

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

New Trends and Ideas

A conceptual and architectural characterization of antifragile systems ☆

Vincenzo Grassi [a], Raffaela Mirandola [b,*], Diego Perez-Palacin [c]

[a] Università di Roma Tor Vergata, Roma, Italy
[b] Karlsruhe Institute of Technology (KIT), Germany
[c] Linnaeus University, Växjö, Sweden

ARTICLE INFO

Keywords:
Antifragility
Dependability
Uncertainty
Software architecture

ABSTRACT

Antifragility is one of the terms that have recently emerged with the aim of indicating a direction that should be pursued toward the objective of designing Information and Communications Technology systems that remain trustworthy despite their dynamic and evolving operating context. We present a characterization of antifragility, aiming to clarify from a conceptual viewpoint the implications of its adoption as a design guideline and its relationships with other approaches sharing a similar objective. To this end, we discuss the inclusion of antifragility (and related concepts) within the well-known dependability taxonomy, which was proposed a few decades ago with the goal of providing a reference framework to reason about the different facets of the general concern of designing dependable systems. From our conceptual characterization, we then derive a possible path toward the engineering of antifragile systems.

1. Introduction

Information and Communications Technology (ICT) systems represent a vital infrastructure for our society and, consequently, we are increasingly concerned with their trustworthiness. In this respect, it is widely recognized that the ability of these systems to cope with the more and more dynamic and evolving contexts where they operate is a key element to achieving the desired level of confidence about them.

How to design and build ICT systems that remain trustworthy despite changing operating conditions is a long-standing concern and different lines of work have been proposed and pursued to this end (Weyns and others, 2013; Ghezzi, 2016). In parallel with them, it has emerged the need for conceptual frameworks that can help in shedding light on the different facets of this general concern, to clarify specific (and possibly different) objectives that could be pursued, methodologies and techniques that are most suitable to achieve them, and their mutual relationships. In this respect, a fundamental contribution is the taxonomy mainly developed during the '80s and '90s under the *dependability* umbrella term (Avizienis et al., 2004), where dependability is defined as *"[...] the ability [of an ICT system] to deliver service that can justifiably be trusted"* (as an alternative definition, the same paper proposes *"[...] the ability to avoid service failures that are more frequent and more severe than is acceptable"*). That taxonomy, starting from the dependability root term, provided a common vocabulary and conceptual reference for reasoning about the different facets of what impairs dependability, how to cope with it, and how to assess the resulting dependability.

In the next years, other terms besides dependability (and the more specific terms and concepts derived from it in that taxonomy) have been introduced to characterize possible approaches aimed at building ICT systems able to cope with changes. Two of these terms that have emerged over others are *resilience* and, more recently, *antifragility*. Both terms have their origin in areas other than ICT systems, but their usage within this latter area has become increasingly popular, somehow obfuscating any reference to the previous dependability taxonomy from a conceptual and terminology perspective. However, the scope of these terms and underlying concepts and their relationship with other terms and concepts that can be encountered in related literature is sometimes not very clear. It has also been argued (Avižienis, 2017), referring in particular to *resilience*, that this term, as it appears to be used in the ICT literature, does not actually bring with it visions or concepts different from those that could be already derived from the dependability taxonomy (Avizienis et al., 2004).

In this respect, we believe that a primary need for a software engineer involved in the design of trustworthy systems able to cope with changes is to have a commonly agreed-on repertoire of terms and underlying concepts, which makes clear which system aspects each term intends to capture, whether some term is a specialization (qualifications) of some other, or if it denotes a means for attaining a property indicated by another term. From this perspective, our position is that the crisp conceptual reference provided by the dependability taxonomy

☆ Editor: P. Lago.

* Corresponding author.
E-mail addresses: vincenzo.grassi@uniroma2.it (V. Grassi), raffaela.mirandola@kit.edu (R. Mirandola), diego.perez@lnu.se (D. Perez-Palacin).

https://doi.org/10.1016/j.jss.2024.112051
Received 2 October 2023; Received in revised form 5 February 2024; Accepted 1 April 2024
Available online 8 April 2024
0164-1212/© 2024 The Authors. Published by Elsevier Inc. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

should not be lost or obfuscated but rather updated and expanded, if necessary. As a contribution in this direction, in this paper, we revisit the taxonomy presented in Avizienis et al. (2004) and propose an extension of its scope that allows us to give first-class status to issues that were somewhat overlooked in the original proposal. As a result of this extension, we argue that the *antifragility* term and the underlying concepts can be integrated into that taxonomy, maintaining in this way its role of a unified place where the relationships among different goals and approaches aimed at designing and building ICT systems able to cope with changes can be better understood and compared.

Then, based on this conceptual clarification, as a second contribution of this paper, we discuss how to promote the engineering of this antifragility vision. To this end, we first present a reference model for antifragile ICT systems inspired by the three-layer reference model for self-managing systems proposed in Kramer and Magee (2007), and then we delineate a path based on the Digital Twin technology for the realization of antifragile systems.

This paper builds upon the preliminary ideas on this topic presented in Grassi et al. (2023), where a first proposal of including the antifragility concept in the dependability taxonomy was presented, and extends it as follows:

(i) we provide in Section 4 an enhanced characterization of the systems under study including the impact of uncertainty and a refinement of the concept of satisfaction metric as an overall measure of dependability;
(ii) in Section 5.2.1 we introduce and explain the concept of dependability lifecycle to better define the antifragility attribute;
(iii) in Section 6 we include the mapping of the proposed definition into a reference model and present a path toward its realization with a mapping to the Digital Twin standard reference architecture;
(iv) we extensively use the running example in the paper to illustrate the introduced concepts and notation;
(v) we include in Section 7 an extended related work section;
(vi) we include a discussion on open issues in a new section;
(vii) we add an appendix about resilience to complete the analysis of the terminology;
(viii) we include a new appendix with preliminary experiments with the running example about the tractability of antifragility.

Paper organization The rest of the paper is organized as follows. Section 2 briefly reviews the taxonomy presented in Avizienis et al. (2004), highlighting its main concepts, and presents the high-level idea of our contribution. In Section 3, we outline an example system we will use throughout the paper to illustrate some of the concepts we introduce. In Section 4, we present the conceptual framework we use as a basis for our discussion. Then, in Section 5, we present our extensions to the original taxonomy, showing how, thanks to this, the antifragility concept can find a proper placement within that taxonomy. In Section 6, we discuss the engineering implications of the antifragility vision and outline a reference model and a possible realization direction for antifragile ICT systems. Finally, we present related works in Section 7, discuss some open issues related to our proposal in Section 8, and draw some final conclusions and hints for future work in Section 9. To complete the terminology analysis, in Appendix A, we briefly present some considerations about the *resilience* term, and in Appendix B we presents some preliminary experiments with the running example about antifragility tractability.

2. Background

Antifragility. The notion of *antifragility* originates from N. N. Taleb's book (Taleb, 2012), where the following often-cited definition is given: *"Antifragility is beyond resilience or robustness. The resilient resists shocks and stays the same; the antifragile gets better."*. An antifragile system is thus a system that thrives and improves when facing events with a negative impact. As noted in Taleb (2012), the immune system is a typical example of such a system taken from the natural world.

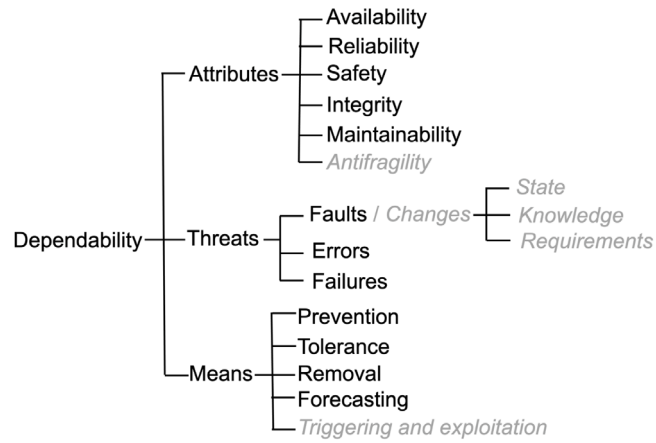

Fig. 1. The dependability taxonomy and the proposed extensions.
Source: Elaborated from Avizienis et al. (2004).

Taleb's book, however, mainly presents the general vision of antifragility, with no focus on specific domains like ICT systems.

The dependability taxonomy. The taxonomy presented in Avizienis et al. (2004) (the black parts of Fig. 1 illustrate its first levels) starts from the *dependability* root term, which identifies the context this taxonomy refers to, as stated in the definitions reported in Section 1. At the immediately next level of the taxonomy tree, we find three different branches: *attributes*, which define a vocabulary of terms that can be used to characterize from (partially) different perspectives the properties an ICT system should exhibit to consider it dependable; *means*, which identify different (not mutually exclusive) possible categories where we can group approaches that can be adopted to attain dependability, as described by its attributes; *threats*, which specify what these approaches should cope with, which can impair dependability, distinguishing the different facets captured by the concepts of *fault*, *error*, and *failure*. In particular, these latter three concepts are related to each other through the identification of the "chain of dependability threats": $\cdots \rightarrow$*fault*$\rightarrow$*error*$\rightarrow$*failure*$\rightarrow$*fault*$\rightarrow \ldots$. In this chain, a *failure* is the manifestation at the system interface of a deviation of the delivered service from the correct service, an *error* is the part of the system's total state that may lead to a failure, while a *fault* is defined as *"[...] The adjudged or hypothesized cause of an error [...]"*. In particular, faults can be classified as *internal* when they originate within the system boundary, or *external* when they originate outside the system boundary (i.e., in the system environment).

In the end, as stated in the definition reported in the Introduction, it is the occurrence of failures beyond some acceptable frequency and severity that impairs the trust we have in the system capacity to deliver an acceptable service, i.e., its dependability. Hence, over a narrow horizon, a fault is the cause of an error, while over a broader horizon, it is the cause that triggers the chain of events that shape our perception of the system's dependability.

Our contribution. With respect to the background outlined in the two paragraphs above, the extension we propose involves all three branches of the taxonomy. We aim to find a proper collocation to the antifragility concept within that taxonomy, which clarifies its relationships and interactions with other taxonomy concepts and suggests directions for designing and operating an antifragile ICT system. To this end, we extend the scope of possible *threats* to dependability, by expanding the concept of *fault* into the more general concept of *change*, which includes in particular what we call *knowledge change*. Then, we propose adding *antifragility* to the class of possible dependability *attributes* as a

new attribute focusing on assessing a system's capacity to cope with knowledge changes. Finally, as a possible suitable way to cope with this new kind of threat, we extend the categories of *means* to achieve dependability, including in it the new category of *changes triggering and exploitation*.

The gray italics parts of Fig. 1 summarize this extension, which is discussed in detail in Section 5. This extension then provides the conceptual basis for the path toward the engineering of antifragile ICT systems presented in Section 6.

3. Running example

To illustrate some of the main concepts introduced in the paper we adopt an example inspired by the case provided in Hole (2016) on malware spreading over a networked system. The system consists of distributed nodes that need to connect with each other to accomplish the system mission, where nodes can be computers, mobile robots, IoT devices, etc. The network topology is dynamic in the sense that nodes can appear/disappear, and the system operates according to a self-organizing principle, where each node decides to which other nodes it should connect to accomplish the current mission. Software hosted by nodes is susceptible to malware attacks, where the infection can originate from an external attack (outbreak of malware) or from an internal propagation from another infected node. Regarding the infection propagation, an unhealthy node will infect all its directly connected nodes with which it shares a vulnerability. Different software applications and different versions of the same application may have different vulnerabilities. Nodes can realize that they have been infected and apply a self-healing operation, but recognizing the infection is not immediate. Fig. 2 illustrates a network of ten nodes of five different types, where each geometric shape (e.g., ○, △, □, …) represents a possible type, characterized by the specific set of applications (and relative versions) it hosts. Dot filled nodes 1-3 are infected. Node 2 may have already propagated its infection to node 3, and node 3 could propagate its infection to nodes 4 and 10.

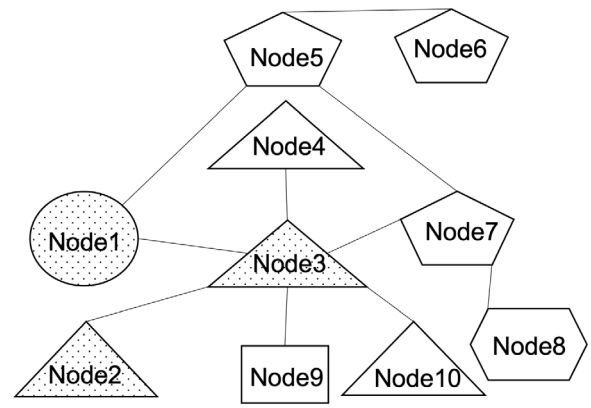

Fig. 2. Example of a networked system with different types of nodes.

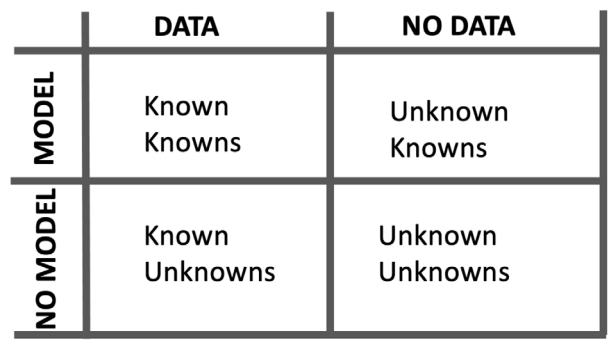

Fig. 3. Four-quadrants uncertainty model.
Source: Adapted from Casti (2011).

4. Conceptual framework

In this section we first discuss (Section 4.1) the impact of uncertainty on dependability, then we present (Section 4.2) the notation we use throughout the paper, and finally (Section 4.3) we introduce a possible measure of dependability.

4.1. Characterization of uncertainty

As recalled in the Introduction, the taxonomy presented in Avizienis et al. (2004) stresses the point that a system is dependable if we can have a *justifiable trust* about its ability to deliver the service it is intended for. This trust can be undermined by the presence of *threats*, categorized in Avizienis et al. (2004) in the three distinct classes of *faults*, *errors*, and *failures*. We believe that, besides these threats, another important factor deserves to be explicitly considered when talking about the dependability of a system: the *uncertainty* we have about the threats that could affect the system, concerning, for example, what they really are, and when, where, and how they could occur. This uncertainty heavily contributes to shaping the trust we have in a system, i.e., our perception of its dependability.

This uncertainty mainly derives from a lack of knowledge, and different terminologies and taxonomies (Perez-Palacin et al., 2014; Weyns and others, 2013) have been defined to characterize the spectrum of its possible manifestations. In this paper, we choose to refer to a characterization along the two dimensions of *model* and *data*, derived from Casti (2011). The first dimension (*model*) refers to the availability (or lack) of a body of knowledge about some fragment of the real world we are interested in: in our context, this fragment includes a system and its operating environment. The model provides a framework, based on sources including past experience, theoretical assumptions, and field observations, which allows us to interpret or forecast the current or future system behavior and its interactions with the environment. In particular, it is thanks to this model that we can assess our perception of the system dependability. The second dimension (*data*) refers to the availability (or lack) of information that can be exploited to instantiate or refine a model, and/or to validate it. Depending on what we have along these two dimensions, we can give the following (coarse) characterization of the uncertainty scenarios we could have to cope with de Lemos et al. (2013) Casti (2011), as shown in Fig. 3:

- *Known Knowns* quadrant: in this case, both a model and the data to instantiate and analyze it are available [Model and Data];
- *Known Unknowns* quadrant: in this case, data are available, but we lack a model that fits those data and supports their interpretation [No Model and Data];
- *Unknown Knowns* quadrant: in this case, a model is available, but we lack some data that would be necessary to "fill" it [Model and No Data];
- *Unknown Unknowns* quadrant: this case is also called *deep uncertainty* (Walker et al., 2003), or *latent uncertainty* (Baruwal Chhetri et al., 2019), that is an uncertainty of which there is no awareness: neither a model nor data to instantiate it are present [No Model and No Data].

4.2. Notation

We now introduce the notation we will use to refer to the fragment of the real world we are interested in:

- S: an ICT *system* (in the following simply called: system) made of hardware and software components that together implement some domain-specific logic;
- E: the *environment* with which the system interacts through a suitable interface, where the boundary between the system and the environment is based on the perspective and context of an observer.
- R: the *requirements* that express the expectations that the system's stakeholders have about what is the correct behavior of the system within its environment. These expectations are stated as a set of properties that must/should hold, concerning what the system must/should do (functional properties) and how it should do it (non-functional properties, e.g., concerning the system performance, security, etc.).
- M: a *model* that captures at some suitable abstraction level the current level of knowledge and understanding we have of S and E.

The quality of S (and hence the trust we can have in it) depends on its ability to fulfill R in the environment E. However, the assessment we can make of this quality strongly depends on M. We remark that M represents the *partial* knowledge we have about S and E, as what we know and understand about them can be affected by different types of uncertainty or even complete ignorance (Perez-Palacin and Mirandola, 2014). Hence, M captures only a subset of the whole set of elements S and E are actually made of. Referring to the four-quadrants classification of the uncertainty scenarios discussed above, the lowermost two quadrants in Fig. 3 are out of the scope of M, and the uncertainty they reflect thus represents a serious threat to our trust assessment. In turn, for the subset of S and E covered by M (uppermost two quadrants in Fig. 3), M not only captures the current status but also represents the effect of the changes/faults of which there exist knowledge. This representation can be "imprecise" (e.g., because of the lack of enough data needed to estimate some model parameters or because of the adopted abstraction level) and partially uncertain (e.g., there is no observed data that supports or bring full awareness of the effect of a change).

Example 1. In the malware spreading example, S includes nodes, their connections, vulnerabilities, healing processes, etc., while E includes the set $\mathcal{A}$ of cyberattack vectors that could infect nodes of S.

R could include the following two quality requirements: (i) the system should keep, on average, at least 95% of its nodes healthy (i.e., non-infected) along time, and (ii) each node should be connected to at least two nodes.

M represents what we actually know of S and E at a given time. This body of knowledge could include:

- an instance of the metamodel in Fig. 4 that represents the current knowledge of the system and environment;
- the assumption that nodes of different types do not share any vulnerability because their software binaries are different (represented in Fig. 4 by setting to one the multiplicity of the *VulnerableSoftware* association); as a consequence, an infected node can propagate the infection only to its directly connected nodes of the same type;
- a set $A \subset \mathcal{A}$ of known attack vectors ($\mathcal{A} \setminus A$ is thus the set of attack vectors not (yet) known to system designers);
- a stochastic model of the behavior of nodes health lifecycle, from their infection and malware propagation to their recovery by the self-healing process; it could simply consist of a random sampling based on the *probAttack* probability or *meanHealingT* value in Fig. 4: they refer, respectively, to the probability that an attack successfully exploits some vulnerability of a node's software and the mean time that the node takes to heal from the infection.

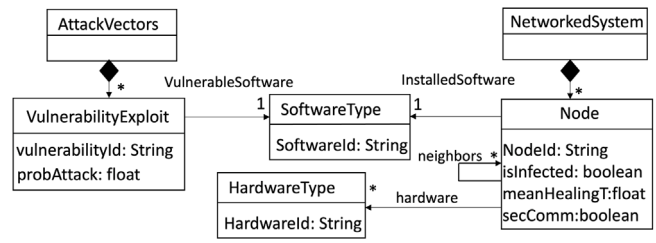

Fig. 4. Metamodel part of the body of knowledge in M.

Based on this body of knowledge embedded in M, we can assess the system's ability to fulfill R. This assessment may be affected by different types of uncertainty. Some uncertainties can be located within the upper two quadrants ((Un-)Known Known) in Fig. 3: they could concern, for example, the value of the *probAttack* probability for known instances of *AttackVectors*, or even the composition itself of the set of *AttackVectors* instances. In the former case, we likely already have data to estimate the probability value, but more data would allow a more precise and updated estimation. In the latter case, we could not have yet collected evidence of the presence of some new attack vector (lack of data), but our model is anyway ready to take it into account when it will be possibly detected. Other uncertainties can be located within the lower two quadrants ((Un-)Known Unknowns). They could concern, for example, the adequacy of the adopted model for the infection propagation: it is based on assumptions that, in the future, could be revealed to be inadequate, but, at present, we are not able to understand which of them should be possibly corrected and in which way.

4.3. A measure of dependability

The taxonomy presented in Avizienis et al. (2004) does not provide suggestions about possible "measures" of the overall level of trust we could have in a given system, i.e., of its dependability. It proposes instead a list of *attributes* (e.g., reliability, availability, safety, …) that can be used to measure from different perspectives the properties that the system should manifest to allow us to trust it. We propose an expansion of this perspective, starting from the argument that, in the end, we trust a system if we are confident that (i) it is able to do what it was designed for (functional requirements), (ii) it is able to do it well (non-functional requirements), (iii) notwithstanding possible threats to its ability to do that. In other words, we trust a system if we are *reasonably confident* that it is able to fulfill its requirements, where these requirements include, in general, statements about functions to be performed, and about quality expectations we have about those functions. In particular, the latter could concern different aspects of the delivered service, including its performance (e.g., response time), its readiness (e.g., availability Avizienis et al., 2004), or its continuity (e.g., reliability Avizienis et al., 2004). In this respect, we note that Avizienis et al. (2004) focuses only on attributes like the latter two.

Based on these considerations, we introduce the concept of *satisfaction function* $Q_R(M) \in \mathcal{U}$, as a possible overall measure of the degree of trust we have in a system, i.e., of its dependability, where $\mathcal{U}$ is some suitable (possibly multi-dimensional) metric space with an associated ordering relation $\preceq$. In our vision, $Q_R(M)$, together with the metric space $(\mathcal{U}, \preceq)$ it is defined on, is intended to play a double-sided role:

- on the one side, it assesses to what extent the system S is able to fulfill in the environment E the properties stated in R, based on what we know and understand about them, i.e., M. In this respect, we point out the different roles that the "parameters"

R and M play in the definition of $Q_R(M)$: R defines the (multidimensional) "space" within which a given system S is to be evaluated, and specifies the boundaries of the acceptable region within that space; M allows to identify the "point" within that space corresponding to S. Based on this, $Q_R(M)$ assesses whether the point represented by M lies within the acceptable region, or how distant it is from it (where this general concept of "distance" should be suitably refined according to the domain-specific characteristics);

- on the other side, as discussed in Section 4.1, $Q_R(M)$ should also embed information about the uncertainty affecting the provided assessment of the distance between the "point" represented by M and the acceptable region specified by R. This uncertainty results from the combination of different factors, including the lack of precision in the measurement of a given quantity or the degree of confidence or belief we assign to the modeling assumptions underlying the provided assessment.

Hence, given a set of requirements R and two models M' and M'' corresponding to two different systems S' and S'' intended to fulfill R, $Q_R(M') \preceq Q_R(M'')$ means that, based on the knowledge we have, S'' can be considered more dependable than S', taking into account both the estimated distance from R and the uncertainties about this estimation.

Example 2. Let us assume that for the considered example system we are only interested in how many of the two requirements stated in Example 1 the system fulfills, with a slight preference for requirement (i) in case of partial fulfillment. Moreover, in case of the unfulfillment of a requirement, no relevance is given to how far the system is from its fulfillment. Under these assumptions, using fictional values for the sake of exemplification, a possible (very simplistic) *distance function* measuring the distance from R of the system modeled by M could be defined as follows (where lower is better):

$$dist(R, M) = \begin{cases} 0 & \text{if both requirements are fulfilled} \\ 0.3 & \text{if only requirement (i) is fulfilled} \\ 0.5 & \text{if only requirements (ii) is fulfilled} \\ 1.0 & \text{otherwise} \end{cases}$$

Moreover, let us assume that we have a separate measure of our confidence in the distance estimation (due to uncertainties affecting the distance assessment), based on a 0-1 scale, where 0 and 1 mean no confidence and complete confidence, respectively.

A possible (again very simplistic) *satisfaction metric* could be thus defined as follows:

$$Q_R(M) = (1.0 - dist(R, M)) \cdot confidence$$

With this definition, the metric space $\mathcal{U}$ corresponds to the set $\mathbb{R} \in [0, 1]$, and the ordering relation $\preceq$ corresponds to the usual $\leq$ relation defined on this set.

Actually evaluating $Q_R(M)$ defined in this way for a given system modeled by M requires carrying out an analysis procedure based on the model M outlined in Example 1 (where M includes, for example, the stochastic model of infection propagation, and the assumptions made about how infection propagates). As an example, this analysis for a given model instance M' could result in assessing that the system it represents fulfills both requirements, but the confidence associated with this assessment is only 0.6; hence, our overall trustworthiness degree is expressed by $Q_R(M') = 1.0 \cdot 0.6 = 0.6$. On the other hand, for another model instance M'' referring to a different system, the analysis carried out using M'' could reveal that this alternative system fulfills only requirement (i) in Example 1, but the confidence about this assessment is high, being equal to 0.95. Hence, for this second system, we have $Q_R(M'') = 0.7 \cdot 0.95 = 0.665$, which represents a higher overall trustworthiness degree with respect to the former system.

5. Extending the dependability taxonomy

Based on the considerations discussed in the previous section, we revisit in this section the dependability taxonomy (Avizienis et al., 2004) with the aim of extending and refining its scope. In particular, this extension and refinement will allow us to give a suitable definition and collocation of the *antifragility* concept. To this end, we discuss in the next three subsections issues concerning the three main branches of the original taxonomy: *threats* (Section 5.1), *attributes* (Section 5.2) and *means* (Section 5.3).

5.1. Dependability threats

Within the conceptual framework provided by Avizienis et al. (2004), *faults* play the role of fundamental triggering element of the "chain of dependability threats": $\cdots \rightarrow$*fault*$\rightarrow$*error*$\rightarrow$*failure*$\rightarrow$*fault*$\rightarrow \ldots$. Indeed, they denote the cause that makes S and E enter or remain within an error state, i.e., a state where S could be partially or totally unable to fulfill some of the requirements stated in R. Then, the actual manifestation of this inability at the S's interface represents a failure. Hence, making a system dependable basically means devising a satisfactory way to cope with possible faults and their consequences.

We revisit in this section these concepts, broadening their scope. In particular, we extend besides *faults* the class of possible triggers of the chain of dependability threats, including in this class other elements that were somehow overlooked in the original taxonomy. We give to the elements of this broadened class the general name of *changes* that a system has to cope with and that could impair its dependability.

As a first step, we introduce the following classification into three different types of possible changes that could occur, depending on which of the elements we have identified in our conceptual framework (i.e., S, E, R, or M) they affect:

1. *state changes*: they affect S and/or E by causing the modification of the *state* of some of their elements so as to possibly enter or remain within an error state. They thus include the original concept of fault introduced in Avizienis et al. (2004);
2. *requirement changes*: they affect R, adding to and/or removing from R statements about properties of the system and/or its environment;
3. *knowledge changes*: they affect M, as a result of a modification of the knowledge and understanding we have about S and/or E. We point out that a change of this type does not directly correspond, in general, to a modification of S or E (which we classify as *state* change). They could rather (but not necessarily) be the consequence of the accumulation of observed state changes, which better fit some new knowledge framework obtained by modifying M.

We now argue that besides *state changes*, which, as stated above, include the concept of fault in Avizienis et al. (2004), also the latter two types of changes (*requirement* and *knowledge changes*) can be considered as triggers of the chain of dependability threats:

- *requirement changes*: a change of this type is likely to cause a misalignment between S (operating within E) and the new version of R, as it, in a sense, re-shapes the boundary that delimits the subset of *error states* within the overall set of possible states for S and E. As a consequence, this may bring S and E to an error state, which could lead to the manifestation of a failure, thus resulting in the following "chain of threats":
  requirement change$\rightarrow$*error*$\rightarrow$*failure*$\rightarrow \ldots$;
- *knowledge changes*: at first sight, it could seem that they can hardly be considered as the cause of an error (and then of a failure that could impair the system dependability), as they do not refer to a modification of any of S, E or R, but only to the acquisition of a new (hopefully better) understanding about them. However,

this new understanding can lead to the realization that what was considered as a correct state of S and E with respect to R is actually incorrect, i.e., it is an error that can ultimately lead to the manifestation of a failure. Hence, the assessment we could have made of the system dependability before the occurrence of the knowledge change was fallacious, and after a new assessment, we may realize that the system dependability is actually different (lower). In other words, a knowledge change can generate an error by transforming a hidden error we were unaware of into a manifest error. As a consequence, knowledge changes can also be considered as the starting point of a "chain of threats": *knowledge change→error→failure→...*

From this discussion, the "chain of dependability threats" can be thus extended in this way:

$$\cdots \rightarrow change \rightarrow error \rightarrow failure \rightarrow change \rightarrow \ldots,$$

where *change* belongs to any of the three types we have introduced and includes in it the concept of *fault* of the original taxonomy.

However, we remark that even if these three types of changes can all be encompassed under the common notion of dependability threats, they are related in different ways to the uncertainty characterization outlined in Section 4.1. On the one side, *state* and *requirement changes* are the elements that "fill" the four quadrants of that characterization; in this respect, a given model M represents a specific partitioning among these quadrants of the set of all state and requirement changes that could possibly occur. On the other side, *knowledge changes* enact a modification in the partitioning of this set among the four quadrants, as reflected in the updated/new model they lead to.

Example 3.

- The infection of a healthy node because of a known attack $a' \in A$ is an example of state change originated in the environment E, whose uncertainty status can be situated within the Known Knowns quadrant. Indeed, we may assume to have a model for a' (represented by a specific instance of the *VulnerabilityExploit* metaclass) and data about it (which could include the historical series of past attacks of that type) that allows us to instantiate the class parameters (e.g., the *probAttack* attribute). We remark that some degree of uncertainty is anyway present also in this case, represented by the stochastic model (based on the *probAttack* attribute) that we use to make forecasts about future occurrences of this state change. Similar considerations could be made for the connection/disconnection of a node to/from the system because of an autonomous decision taken by the node itself, which is an example of state change originated within S;
- The infection of a healthy node because of an unknown attack $a'' \in \mathcal{A} \backslash A$ is instead an example of state change whose uncertainty status can be situated within the Unknown Knowns quadrant. Indeed, differently from the previous example, we lacked the necessary data to build a suitable instance of the *VulnerabilityExploit* metaclass corresponding to a''. A consequence of this ignorance is that our current assessment of $Q_R(M)$ does not take into account the impact of this attack;
- Continuing the above example, when we gain awareness of the new attack a'' (after its manifestation), this results in a knowledge change, represented by the update of M through the addition of a suitable instance of the *VulnerabilityExploit* metaclass corresponding to a''. This knowledge change reshapes the boundary between the Known Knowns and Unknown Knowns quadrants: the concepts of vulnerability and corresponding exploitation probability were already present in the model (i.e., they were part of the Knowns embedded in the model, as exemplified by the metamodel in Fig. 4), but we lacked some (Unknown) data about them;
- The realization of the fallacy of the assumption that an infection propagates only to nodes of the same type is instead an example of *knowledge change* that reshapes the boundary of the (Un)known Unknowns quadrants: the metamodel in Fig. 4 constraints to one the multiplicity of the *VulnerableSoftware* association, which means that the possibility of infections propagating to nodes of different types was completely beyond our awareness horizon. No model instance can be created from that metamodel that can appropriately fit the data that we now observe. This knowledge change has thus a deeper impact than the previous one, as it will lead to the definition of a different metamodel;
- An example of requirement change whose uncertainty can be situated in the Known Knowns quadrant can be, for example, the change toward always using secure communication between nodes, enforced by a law announced in advance. In this case, indeed, we have both the model (the boolean `secComm` attribute included in the Node class that flags whether the node implements secure communication) and the data (derived from testing secure communication between nodes and observing the nodes that already implemented it before it was enforced by law);
- An example of requirement change in the (Un)known Unknowns quadrants could be the addition of a new requirement to the set R stated in Example 1 (leading to a new set of requirements R'), caused by the emergence of an unforeseen new stakeholder. The new stakeholder could bring into the system sustainability concerns, requiring, for example, that the nodes average energy consumption remains below a given threshold θ.

5.2. Dependability attributes

In this section, we start with a characterization of the lifecycle of a system from the perspective of its dependability. Then, based on this characterization, we give a definition of *antifragility* and motivate its addition to the list of *attributes* that the taxonomy (Avizienis et al., 2004) proposes as possible qualifications of the system dependability.

5.2.1. The dependability lifecycle

From a dependability perspective, the lifecycle of a system S can be seen as consisting of a sequence of successive *phases*. Each phase corresponds to a single realization of the *outer dependability cycle* in Fig. 5. A phase starts with the release of a *version* of S that is deemed adequate to fulfill R in a given E, based on the knowledge we have about them (i.e., M). The adequacy of the current version of S means that we are satisfied by the level of trust we can put in it, as measured by the value of $Q_R(M)$. When this adequacy no longer holds (as discussed below), the current phase ends and a new version of S is released, which marks the beginning of the next phase (next cycle of the outer cycle).

Within each phase, S and E are not "static", as they are subject to what we have called state changes (as depicted by the self-loop of the state labeled *S, E, M, R* in Fig. 5). However, these changes do not modify $Q_R(M)$, i.e., the assessment we have made of the S's dependability, as long as their occurrence does not lead to the modification of one of the "pillars" $Q_R(M)$ is based on, i.e., R or M, resulting in a modification of its value.

Requirement or knowledge changes can also occur within a phase (transitions labeled as *R-change* and *K-change* in Fig. 5), possibly leading to a modification of the $Q_R(M)$ value. If this modification does not make $Q_R(M)$ exit its acceptable region, this means that S is adequate also for the new knowledge/requirement scenario, and the system remains within the *inner dependability cycle* evidenced as dotted arrows in Fig. 5 (which thus includes only state changes and K- or R-changes that keep $Q_R(M)$ to an acceptable value). On the other hand, if $Q_R(M)$ drops below the acceptable threshold because of an *R-change* or *K-change*, this motivates the release of a new version of S (transitions labeled *R-triggered* and *K-triggered evolution*, respectively), which starts

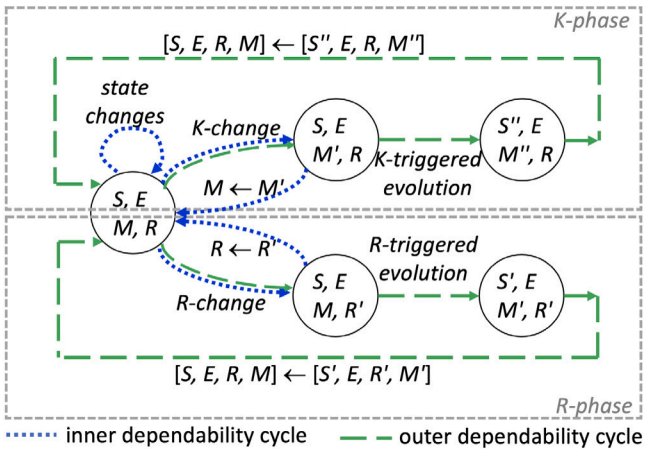

Fig. 5. Dependability cycle.

a new phase (outer cycle) where $Q_R(M)$ has been brought again to an acceptable value.

Fig. 5 evidences, in the succession of phases that constitute the outer cycle (dashed arrows), the distinction between two types of phases, named *R-phase* and *K-phase*, depending on whether they terminate because of the occurrence of a *R-change* or *K-change*, respectively. Both types of phases start with given S, E, R and M (resulting from the previous phase), and the consequent value $Q_R(M)$, which assesses the trust we have on S. From that point on, they evolve differently, as briefly discussed below.

R-phase. When an *R-change* occurs, it modifies R to some new R', but leaves M unchanged, as it does not impact, in general, the understanding we have of S and E. However, the *R-change* leads to a reassessment of the system dependability, measured by a new value $Q_{R'}(M)$. If this new dependability value turns out to be unsatisfactory, this motivates (as shown in the lower part of Fig. 5) the release of a new version of the system, denoted as S', with the consequent update of the model M to a new M' that embeds the novelties introduced in S' with respect to S, and (hopefully) guarantees a satisfactory dependability value $Q_{R'}(M')$. R' and M' thus represent, respectively, the new requirements and the updated knowledge that characterize the beginning of the next phase.

K-phase. When a *K-change* occurs, it modifies M to some new M', while R remains unchanged. This leads to a reassessment of the system dependability, measured by a new value $Q_R(M')$. If this new dependability value turns out to be unsatisfactory, this again motivates (as shown in the upper part of Fig. 5) the release of a new version of the system (S''), and the consequent update of the already updated model M' to a newer M'', which embeds the novelties introduced in S'' with respect to S. M'' thus represents the updated knowledge that characterizes the beginning of the next phase, while the requirements R remain unchanged.

Example 4.

- The knowledge changes mentioned in Example 3 (e.g., the discovery of a new type of attack, or the realization of the existence of attacks affecting nodes of different types simultaneously) lead to the definition of a new model M' that includes the new sources of risk for the system. This may lead to realizing that the system S trustworthiness measured by $Q_R(M')$ has decreased below the acceptable value, thus triggering the release of a new version S'' (with the associated model M'') embedding suitable countermeasures that bring back the system trustworthiness to the previous acceptable value, or even better (i.e., $Q_R(M) \preceq Q_R(M'')$). These countermeasures could include the introduction of patches in the affected software, which at least partially remedy the newly discovered vulnerability or the implementation of new proactive healing procedures.
- The requirement change mentioned in Example 3 (modification of the set R into a new set R', because of the addition of a new requirement concerning energy saving) triggers the release of a new version S' (with the associated model M'), in case the existing system S results unable to fulfill the new requirement (i.e., it results $Q_{R'}(M)$ below the acceptable threshold). We point out that the fulfillment of this new requirement could imply the need for a trade-off with the fulfillment of previous requirements: for example, this would be the case when energy saving is achieved by reducing the frequency of the periodic preventive self-healing procedures, which could increase the risk of undetected infection propagation, with a negative impact on the requirement concerning the node healthiness. As a consequence, it could result in $Q_{R'}(M) \prec Q_{R'}(M')$ (comparison with respect to the extended requirements), while $Q_R(M') \prec Q_R(M)$ (comparison with respect to the original requirements).

5.2.2. Antifragility

The concept of *dependability cycle* introduced in the previous section gives us the basis for the characterization of the *antifragility* concept. To this end, we start from the definition of antifragility for ICT systems proposed, for example, in Russo and Ciancarini (2016), Hole (2016), which largely recalls the general one already quoted in Section 2 (Taleb, 2012): *"Antifragility is beyond resilience or robustness. The resilient resists shocks and stays the same; the antifragile gets better"*. This definition strictly connects antifragility with the vision of a system that does not simply resist "shocks" (i.e., changes, in the terminology we have introduced) but rather exploits them to continuously *improve* itself. However, this concept of "improvement" remains quite generic. To give it a more precise characterization, we connect it with the idea of "dependability growth": in our vision, *a system improves itself if it is able to improve the trust its stakeholders put in it (i.e., its dependability)*. According to the discussion in Section 4.3, this amounts to saying that a system improves itself if it is able to get closer and closer to the complete fulfillment of the requirements stated in R and to reduce our uncertainty about that. In our framework, this definition of system improvement can be made more precise as follows:

Definition 1. Given a *succession* of *phases*:

$[phase_0, phase_1, \ldots, phase_n]$, where R^i and M^i denote, respectively, the requirements and model at the beginning of phase i, and $Q^i_{R^i}(M^i)$ denotes the corresponding satisfaction function (trustworthiness level), a system *improves* itself in the timespan covered by those phases if it holds:

$$Q^i_{R^i}(M^i) \preceq Q^{i+1}_{R^{i+1}}(M^{i+1}), 0 \le i \le n-1 \tag{1}$$

A succession of phases $[phase_0, phase_1, \ldots, phase_n]$ fulfilling the condition stated in expression (1) is said to be *monotone*.

Definition 1 implicitly relies on the meaningfulness of the comparison stated in expression (1). In this respect, we recall from Section 4.3 the different roles that R and M play in the definition of $Q_R(M)$: R defines the (multidimensional) evaluation space for a given system S, and specifies the boundaries of the acceptable region within that space; M identifies the point within that space corresponding to S. Finally, $Q_R(M)$ assesses whether this point lies within the acceptable region or how far it is from it. From this perspective, expression (1) makes sense only if M^i and M^{i+1} identify points within the *same space*, and if $Q^i_{R^i}(M^i)$ and $Q^{i+1}_{R^{i+1}}(M^{i+1})$ assess their position with respect to the *same acceptable region*. In other words, expression (1) is meaningful only if it holds:

$$R^i = R^{i+1}, 0 \le i \le n-1 \tag{2}$$

From the discussion at the end of the previous subsection, we see that the condition stated in expression (2) holds only if the succession $[phase_0, phase_1, \ldots, phase_n]$ consists exclusively of *K-phases*. Hence, all these considerations lead to state the following definition of an antifragile system:

Definition 2. A system is *antifragile* if it implements monotone successions of *K-phases*.

An antifragile system is thus a system that is able *to exploit knowledge changes* to improve its dependability. Such a virtuous behavior is evidently a property that could be required for a system to consider it trustworthy in specific scenarios. However, none of the dependability attributes listed in Avizienis et al. (2004) (e.g., availability, reliability, etc.) fits well with this kind of desirable behavior. From the perspective of the dependability taxonomy (Avizienis et al., 2004), our suggestion is thus to extend it by adding *antifragility* to the list of *attributes* that can be used to qualify a system as dependable, as the behavior captured by this new attribute (stated in Definition 2) can, in some contexts, be considered relevant for the trustworthiness of a system.

5.3. Means to attain dependability

Possible means that can be exploited to attain the various attributes of dependability are grouped in Avizienis et al. (2004) into four major categories: *prevention*, *tolerance*, *removal*, and *forecasting*.

In the original proposal, these four categories refer to different ways of coping with the threat represented by *faults*. In Section 5.1 we have discussed the extension of this concept of threat to the wider class of *changes*, which includes in it also the concepts of *requirement change* and *knowledge change*, besides faults. In this section, we discuss whether the four categories of means identified in Avizienis et al. (2004) are also well suited to cope with this extended concept of threat. To this end, we base our discussion on the consideration made in Avizienis et al. (2004) that the four categories of means can be grouped into two different classes, depending on the *aim* they have in coping with a given threat: (i) means aimed at **avoiding** the occurrence of a threat, which include *prevention* and *removal*, and (ii) means aimed at **accepting** the occurrence of a threat, which include *tolerance* and *forecasting* (as shown in the black part of Fig. 6). Considering now the extended concept of changes we have introduced, both these classes appear as reasonable ways of coping with the impact of these changes on the dependability of a given system, as we argue in the following.

In the case of requirement changes, *avoiding* may include approaches aiming at conducting a thorough requirements elicitation phase as much as possible, while *accepting* could include the adoption of approaches fostering the so-called *technical credit*, with the adoption of principles that prevent the introduction of (known or unknown) "negative lifecycle impacts" (Berenbach, 2014).

In the case of knowledge changes, *avoiding* could include approaches aiming at gaining a deep and complete knowledge of the system and its environment as much as possible, keeping the uncertainty about them at a minimum, while *accepting* could include, also in this case, the adoption of technical credit principles, investing in designing and engineering systems able to anticipate as much as possible future emergent situations (Berenbach, 2014).

However, with regard to *knowledge changes*, we point out that such changes should (hopefully) lead to a better understanding of the system and its environment and, possibly, to the release of a better system version. For this reason, another goal to be pursued (besides avoiding or accepting them) should also be to *encourage* their occurrence and their exploitation to evolve the system into an improved version. Hence, besides the means already introduced in Avizienis et al. (2004), we should certainly devise means specifically aimed at achieving this goal, i.e., fostering the occurrence and exploitation of knowledge changes.

Based on these considerations, we suggest the addition of a new category to the four means to attain dependability already introduced in Avizienis et al. (2004), specifically tailored to deal with knowledge changes, taking advantage from them: we call it *change triggering and exploitation*. This category encompasses approaches such as fault injection techniques or adversarial machine learning methods that may help to understand the system's behavior when facing stressful situations and to exploit this knowledge, so realizing system behaviors as described by Definition 2 aimed at attaining *antifragility*.

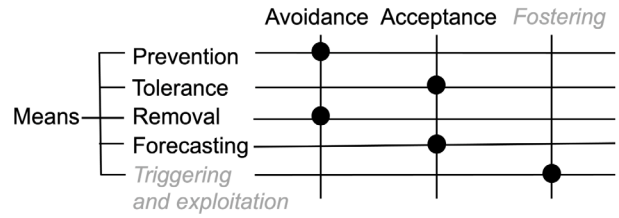

Fig. 6. Classification of means according to their **aim** w.r.t. to changes.
Source: Elaborated from Avizienis et al. (2004).

Fig. 6 evidences (in gray italics) the addition of the *fostering* class of means, besides *avoiding* and *accepting*, and the introduction of the new mean of *change triggering and exploitation*.

Finally, Fig. 1, already presented in Section 2, summarizes the extensions (highlighted in gray italics) of the dependability taxonomy that we have introduced in this section.

6. Antifragility engineering

In the previous sections, we have evidenced the need to give first-class status to *knowledge changes* (K-changes) and we have defined *antifragility* as an attribute that can be used to characterize a (positive) attitude in dealing with this kind of changes. We have also added a new category to the means to attain dependability, called *change triggering and exploitation*, as a suitable means that can be used to improve the antifragility of a system by encouraging and exploiting the occurrence of K-changes.

In this section, to promote the engineering of antifragile systems, we first discuss (Section 6.1) a mapping of this characterization of antifragility into a reference model, whose goal is to help separate different concerns and identify more precisely the main functionality and the open problems that are key in progressing toward the design of antifragile systems. Then, as a further step toward realizing this reference model in practice, we present (Section 6.2) a possible path from the proposed reference model to a concrete reference architecture by leveraging existing Digital Twin frameworks. Specifically, we highlight how the main functionalities of the proposed framework can be managed by the entities in the Digital Twin reference architecture, while also providing considerations about potential advantages and disadvantages (Section 6.3).

6.1. Reference model

The reference model we propose is an extension of the three-layer architectural reference model for self-managing systems presented in Kramer and Magee (2007). This choice brings with itself the idea that antifragility can be seen as an extension of the self-managing design principle, which also includes the system's ability to learn from changes and improve itself. Fig. 7 illustrates our model, evidencing the extension with respect to the original model. In particular, the extension mainly concerns the uppermost layer of the model, while the role of the other two layers remains substantially unchanged, as outlined below.

Bottom and middle layer. In the original proposal, within a self-managing perspective, the role of the bottom and middle layers is to cope with variations in the system itself or in its operating environment by applying pre-defined adaptation plans. What differs between the two layers is the scope of these plans, which in the bottom layer is limited to single components (component tuning and adaptation), while in the middle layer it may generally embrace the whole architecture or relevant parts of it. Both layers also perform event and state monitoring activities and reporting to the respective upper layer, to enable adaptation decisions that go beyond their respective limited scope.

We retain these roles in our reference model. With respect to the conceptual framework discussed in the previous sections, this means that the bottom and middle layers mainly cope with single component or system-wide *state changes*, respectively, whose uncertainty characterization fits in both cases the Known-Knowns quadrant. These changes are handled in the inner dependability cycles within a single phase, as discussed in Section 5.2.1. According to these considerations, in Fig. 7, we name these layers *component-level state change management* and *system-level state change management*, respectively.

Upper layer. The role of this layer includes, as in Kramer and Magee (2007), the responsibility of devising new action plans in response to: (i) the introduction of new goals or (ii) the signaling from the layer below of unforeseen operating states for which the existing plans reveal to be inadequate. In terms of our conceptual framework, this corresponds to the ability to cope with *R-changes* and *K-changes*, respectively (in Fig. 7 we name this layer *R- and K-change management*). However, we point out that our concept of K-change is broader than what is envisioned by point (ii) above, as it includes not only the realization of the existence of unforeseen state changes but also the achievement of a better understanding of the existing system, environment, or requirements, as they are.

Given this broad concept of K-changes, the extension we propose aims at exploiting the potential of these changes to improve the system, in the perspective of devising an antifragile system. To achieve this goal, we thus suggest including in the upper layer the following features, according to what was discussed in Section 5.3:

- *change triggering* mechanisms able to favor (possibly offline, e.g., working with a virtual duplicate of the real system) the occurrence of K-changes;
- *learning and exploitation* mechanisms allowing both the improvement of the system quality (by making it able to cope with new operating conditions or to better cope with already known conditions) and the learning of new aspects that were not predefined.

These two sets of mechanisms should work together to enact the "evolution part" of each dependability cycle by triggering K-changes and exploiting through suitable learning activities the newly acquired knowledge.

Fig. 7 explicitly evidences the features introduced in the upper layer. As a final remark, we point out that, as in Kramer and Magee (2007), the distinction of concerns among the three layers reflects a time scale differentiation: immediate feedback actions are at the lowest layer while the more time-consuming actions (e.g., requiring the definition of new action plans, or system evolution as a consequence of knowledge change) are at the uppermost layer.

Example 5. A possible example of managing logic at the bottom layer (component-level state change management) are the self-check and self-healing operations that nodes can carry out independently to realize that they are infected and solve it. Another example at this layer is the implementation of the circuit-breaker pattern (Richardson, 2018) together with the application of the fail-fast principle to create regions of nodes in quarantine when a node realizes that it is infected and notifies its neighboring nodes. In turn, an action under the responsibility of the middle layer (system-level state change management) may be to frequently analyze the system topology to find nodes that act as a hub (i.e., many other nodes are connected to it) and send them a signal to proactively self-heal (i.e., change their software binaries even if they have not detected that they are infected), since a long-term infection of these nodes is critical to the spread of the malware along the whole system. Finally, an action that can be taken at the upper layer (R- and K-change management) is to apply Chaos Engineering (Basiri et al., 2016) and study its effects in order to gain new knowledge about vulnerabilities of connected nodes and how malware actually spreads across system nodes.

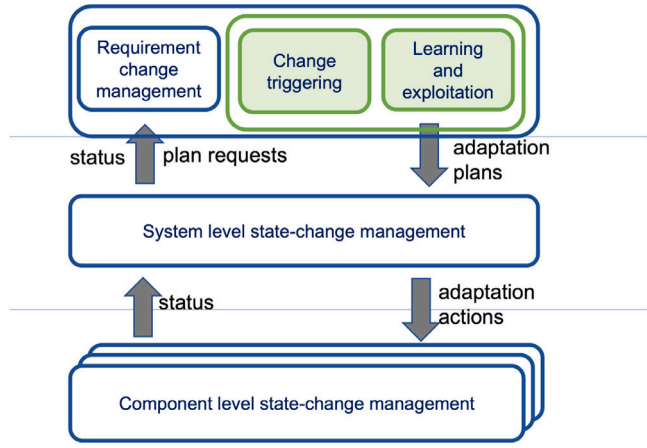

Fig. 7. Reference model.

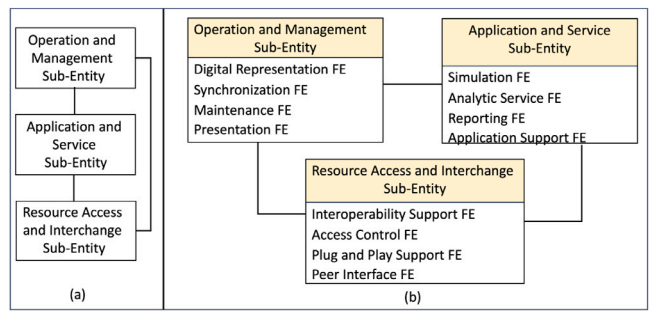

Fig. 8. ISO 23247 Digital Twin Reference Architecture entity (ISO, 2021); (a) Proposed sub-entities, (b) sub-entities functional view.

6.2. From a reference model to a reference architecture

Nowadays, the Digital Twin is a popular technology to collect information and reason about actions in a system. Digital Twins keep their *digital representations* synchronized with the real-world elements. In the following, we leverage the entities defined in the Digital Twin reference architecture specified in the ISO-23247 standard (ISO, 2021) to outline a possible reification of the reference model in Fig. 7, with the aim of suggesting how an antifragile system could be concretely designed. In this respect, we remark that we may reasonably expect that (complex) systems already incorporate technologies for model simulation, behavior planning, etc., leveraging models or digital representations of their components. These technologies clearly enable the full adoption of the Digital Twin technology, thus paving the way toward the incremental achievement of the antifragility goal.

Fig. 8(a) shows the three sub-entities in the Digital Twin reference architecture proposed by ISO-23247—namely *Operation and Management*, *Application and Service*, and *Resource Access and Interchange*—whereas Fig. 8(b) details the corresponding *functional entities* (*FE*).

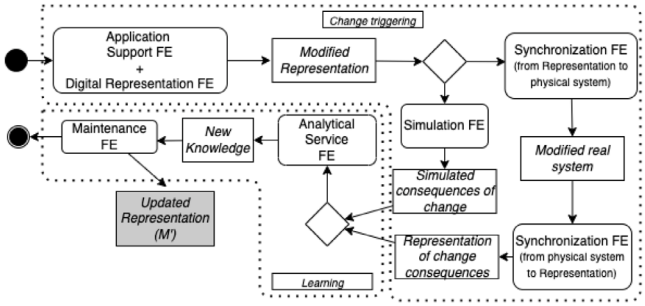

Fig. 9. Collaboration between Digital Twin functional entities to achieve change triggering and exploitation.

With respect to our conceptual framework, the Operation and Management Sub-Entity maintains the information about the real world (S and E) by means of its functional entities (*Synchronization FE*, *Maintenance FE*, *Presentation FE*), while the *Digital Representation FE* handles the model M and the requirements R.

The *Synchronization FE* handles *state changes* affecting directly S and/or E, reflecting the new state in the corresponding digital representation. *Maintenance FE* handles *knowledge changes* affecting M, "repairing" possibly discovered anomalies in the digital representation (we explain below how the new knowledge could be obtained). Finally, *requirement changes* come from a user entity, which is part of the Digital Twin Framework in ISO (2021) and can interact with the Digital Twin entity, through a collaboration with the *Maintenance FE* that fixes the representation of the new requirements in the *Digital Representation FE*.

We now consider in particular *knowledge changes*. The knowledge they bring into the system may come from different sources: directly from the *Analytic Service FE* (part of the Application and Service Sub-Entity), which analyzes the accumulated data collected during previous *state changes* (e.g., referring to the running example, it recalculates the probability of successful attack); from the user entity when a knowledge change has not been internally generated; or from the change triggering, learning and exploitation mechanisms we are envisioning. We focus on the latter one and outline a process for its realization involving different functional entities of the ISO-23247 standard. Fig. 9 sketches the process. First, the *Application Support FE* generates the change to be triggered and studied by accessing the *Digital Representation FE* and producing a modified representation that includes the change effects. Then, the consequences of the change should be unveiled. Two alternative options arise:

- Through the observation of the modified real system. In this case, the *Synchronization FE*, whose functionality is bidirectional, synchronizes the modified representation in the real system. After that, the *Synchronization FE* is again responsible for reflecting in the *Digital Representation FE* the observed consequences of the change that was applied to the system (i.e., how the modified system behaves). This option is related to the Chaos Engineering discipline and has advantages and disadvantages. An obvious advantage is that it is possible to observe the true consequences of any change. A weakness of this alternative is that its application is constrained to learn from changes in S; learning about environment E changes is more complicated because the actuators may not be able to modify E but only S. Another obvious disadvantage is that experimenting with changes in the real system to learn their consequences can take the system down and even threaten safety and security.
- Through offline simulation. In this case, the *Simulation FE* receives the modified representation and predicts the consequences of the change in the system behavior. This alternative presents some advantages: the consequences of a change are obtained safely through a simulation; we can also simulate environment changes; it is possible to make a *sensitivity* analysis or evaluate *what-if* scenarios over a range of intensities for a change. A disadvantage of this alternative is that we need to rely on the results of a simulator, which may not be well-tested for boundary behaviors; moreover, the unveiling of *unknown unknowns* is limited, since a simulator cannot operate using inputs different from those it has been engineered around.

Once the consequences of a change are available, the *Analytical Service FE* is in charge of analyzing them and creating new knowledge. This is the main functional entity where the *learning* is carried out. After that, the *Maintenance FE* updates the representation in the *Representation FE* according to the new knowledge, making it more accurate with respect to the behavior of the real system S and the properties of the environment E.

As a final remark, we point out that we have not mentioned in our presentation all the functional entities included in the ISO-23247 reference architecture standard (ISO, 2021). This does not mean that they are irrelevant, but we do not delve into them as they appear to be out of the scope of change triggering and learning mechanisms, which is our focus.

6.3. Engineering considerations

We discuss here some potential impediments in the suggested path toward antifragility engineering. We first focus on the situations where *change triggering* fosters K-changes, either by modifying the real system or by simulation (Fig. 9). The potential impediments in the observation of the consequences of changes in the real system stem from the usual tasks carried out in the data cleaning area. It is necessary to identify incorrect data and fix the data set for, subsequently, learning from it. Incorrect data may refer to corrupted, duplicated, outliers, gaps over time, or unexpected formats in part of the data in the dataset. This task, which is commonly performed by data quality analysts, requires leveling up its automation to approach the expected level of autonomy in antifragile ICT systems. Another potential impediment is related to the synchronization activities between the model (or digital representation) and the real system and this is one of the challenges in digital twin deployments that are currently receiving research attention.

On the other hand, when the consequences of changes are obtained through simulation, the impediments to observing such consequences include the limitations of the simulation engine and its degree of coupling with the system model. The engineers need to clearly understand what is being simulated and which information in the model is ignored by the simulation engine. Moreover, the simulation engine most probably uses a language different from the one adopted for the system representation. In such a case, the Model-Driven Engineering field provides solid engineering techniques for integrating these two languages through Model-to-Model transformations. However, the engineering team needs expertise in Model-Driven Engineering to appropriately use the model transformation languages.

Regarding the system improvement on behalf of the *learning and exploitation mechanisms*, the evolution task carried out after a K-change or R-change (Fig. 5) may be engineered following disparate methods, from the pure manual evolution of the system to the autonomous self-evolving software, passing through semi-automatic methods where the system may autonomously find a potentially higher quality evolution being guided and supervised by an engineer. Automated approaches may take advantage of faster system simulation to compute the quality of different potential evolution alternatives of the system and propose the subset of options with the expected best quality. This makes the simulation engine (the *Simulation FE* in the Digital Twin reference architecture) be accessed from different parts of the antifragility process. The experiments illustrated in Appendix B implement this possibility where a simulator is used by the *change triggering* and by the *learning and exploitation* during the exploration of alternative networked systems to calculate the expected quality of each candidate.

7. Related work

In this section, we briefly summarize works on dependability, resilience, and antifragility on which we leveraged (Section 7.1) and existing architectural solutions dealing with (un)known (un)knowns (Section 7.2).

7.1. Dependability, resilience, and antifragility

The word *dependability* is often used broadly to indicate a set of properties that guarantee that a system is functioning as required. In 2004, Avizienis et al. (2004) defined a taxonomy that formalizes the definition of dependability as a set of attributes (e.g., reliability, safety, availability), means to attain dependability (e.g., fault prevention, fault tolerance) and threats to dependability (faults, errors, and failures). From 2004 until now, several other terms have emerged as, for example, robustness, resilience, and self-healing (Andersson et al., 2021; Ghosh et al., 2007; Laprie, 2008; Schmeck et al., 2010; Guelfi, 2011; Woods, 2015; Weyns and others, 2013), to characterize the system ability to deal with faults and changes. Many of these papers provide conceptual frameworks that assist in identifying the current state of the art, relationships among different approaches, and promising research avenues. However, they focus on some of these terms without considering their collocation into a general framework like the dependability taxonomy (Avizienis et al., 2004).

With the growing number of terms referring to concepts whose boundaries are not always well defined, Avizienis in 2017 discussed this "jungle of terminology" (Avižienis, 2017) to compare the different terms and find common ground. This study suggests that the use of the traditional taxonomy is able to cover the new terms that emerged in this field.

At the same time, the use of the term *antifragility* introduced in 2012 in Taleb (2012) as the ability of a system to work in an open, uncertain environment being able to absorb changes, react meaningfully even to unforeseen events and get even better afterward, is gaining momentum. A recent comprehensive survey analyzing antifragility and resilience with the aim of shedding light on approaches able to manage disruptions in business ecosystems is presented in Ramezani and Camarinha-Matos (2020). A classification of disruption sources and drivers is considered, complemented by a list of strategies to handle disruptions and examples of engineered systems implementing promising approaches to increase resilience and antifragility. In the area of ICT systems, Hole (2016) analyzes the concept of antifragility and illustrates several examples of how often today's ICT systems are fragile to downtime and other large-impact events. The book then discusses and illustrates, with some examples, a set of principles like modularity, redundancy, and diversity, to cite a few, which could help move toward the engineering of antifragile systems. A similar set of principles has also been outlined in Russo and Ciancarini (2016, 2017) focusing on software engineering and software architectures, respectively. The need for a change of perspective when designing new antifragile systems is also outlined in Bakhouya and Gaber (2014), de Bruijn et al. (2020), Gorgeon (2015), which emphasizes the need to embrace the change from the system conception phase instead of facing it.

With respect to these works, we have chosen to maintain the dependability taxonomy (Avizienis et al., 2004) as the basic conceptual reference model for the design of trustworthy systems, and to integrate in it newly emerged concepts such as antifragility.

7.2. Architecting antifragility

As concerns existing *architectural solutions* underlying systems whose goal is dealing with changes, we review here a set of recent architectural proposals that can be considered a step forward toward a systematic approach to the design of antifragile systems (e.g., Gheibi and Weyns, 2022; Klös et al., 2018a; Perrouin et al., 2012; Baruwal Chhetri et al., 2019; Bouchenak et al., 2011; Zavala et al., 2020; Roth et al., 2015; Baruwal Chhetri et al., 2018). For all of them, their design choices can be understood in terms of the extended three-layer reference model we have proposed in Section 6.1. However, we note that in most of the cases, the extensions they propose (mainly concerning the uppermost layer of the model) only partially cover all the concerns that would be required for a full-fledged antifragile system. Most of them, indeed, include a learning component but do not include the change triggering and exploitation capability.

These proposed architectures share a common conceptual vision for the bottom and middle layers. In particular, the middle layer implements the system-wide self-adaptation managing logic, mainly architected according to the well-known *MAPE-K* model (Kephart and Chess, 2003), where the K(knowledge) component stores the predefined adaptation plans that are then implemented by the other *MAPE* components.

As remarked above, the definition of the upper layer functionalities is the characterizing aspect of these architectures, as summarized in the remainder of this section and in Table 1.

Work in Gheibi and Weyns (2022) presents an architecture proposal for a lifelong self-adaptive system, where the new layer goal is to detect and manage new knowledge through learning mechanisms, e.g., by updating and evolving the learning models and adaptation plans implemented by the underlying layers. In Klös et al. (2018a), the upper layer includes an evaluation, a learning, and a verification component whose goal is to adjust the adaptation logic implemented by the underlying layer when this is deemed necessary on the basis of newly acquired knowledge. The idea of an upper layer managing self-improvement through learning can also be found in Baruwal Chhetri et al. (2019). An upper layer, organized with a feedback loop based on the $MAPE-K$ model, is introduced in Roth et al. (2015), Perrouin et al. (2012), and in Zavala et al. (2020) with the aim to adapt the adaptation logic of the managing system. The definition of a layer implementing self-self management by adopting a recursive design and mirroring and replicating the autonomic components is presented in Bouchenak et al. (2011) and exploited in Baruwal Chhetri et al. (2018) and Uzunov et al. (2021) with the addition of learning capabilities.

The approach that appears to be closer to our vision is the one presented in Hashmi et al. (2022), which focuses on secure systems. The authors do not present explicitly a software architecture for their approach but plan to exploit the AWARE (Baruwal Chhetri et al., 2018) framework and enhance it to encompass self-exploration, self-learning and self-training activities. Specifically, they envision systems able to self-explore (through adversarial search approaches) the potential impact of the cyber-attacks to evaluate suitable responses. Self-learning abilities could help both the exploration and the understanding of the attacks, while self-training capabilities, based on reinforcement learning techniques, aim to help the evolution and the improvement of the system as a result of the attacks. These new activities nicely fit the reference model outlined in Fig. 7.

In summary, all the revised solutions can be characterized as partial instances of the three-layer reference model we have proposed, as they extend the model in Kramer and Magee (2007) by explicitly including in the upper layer learning capabilities aimed at better enabling the system to cope with modifications of the operating conditions. However, what we want to stress with our reference model that extends the model in Kramer and Magee (2007) is that the full vision of an antifragile system (i.e., a system that learns from experience and improves itself) would require exploiting learning capabilities also in the absence of significant variations of the operating conditions, with the goal of gaining a better understanding (knowledge change) of how the system performs in a given environment, and how this performance could be improved (which could also possibly lead to uncovering hidden parts of the system itself or its environment). To this end, mechanisms like fault injection and adversarial techniques in the uppermost layer could help, and should be added to the proposed architectural solutions to make them fully antifragile.

Table 1
Upper layer roles in proposed architectures.

Name	Upper layer
Lifelong Computing (Gheibi and Weyns, 2022)	Meta-learning
Self-learning (Klös et al., 2018b)	Evaluation, Learning and Verification
ALM (Roth et al., 2015)	MAPE-K loop with Prediction and learning
DAS (Perrouin et al., 2012)	MAPE-K loop and variability models
HaFLoop (Zavala et al., 2020)	Centralized and decentralized MAPE-K loops
Self-Improving (Baruwal Chhetri et al., 2019)	Agent-based learning
JADE (Bouchenak et al., 2011)	Autonomic component mirroring and replication
AWARE (Baruwal Chhetri et al., 2018)	Learning
AWARE2 (Uzunov et al., 2021)	
Adversarial search (Hashmi et al., 2022)	Exploration, learning and training

8. Discussion

In this section, we discuss the connection between antifragility and requirement changes (Section 8.1), the growth of the design space to explore (Section 8.2), and some issues underlying the definition of $Q_R(M)$ (Section 8.3).

8.1. Requirement changes and antifragility

The discussion in Section 5 establishes a direct connection between *antifragility* and *knowledge changes*. Indeed, it characterizes antifragility as a system attribute that describes its ability to cope with knowledge changes, taking advantage of them to improve its dependability (as measured by the satisfaction function $Q_R(M)$), where the "improvement" is observed with respect to an invariant set of requirements R. We note that, indirectly, this characterization establishes a connection also between antifragility and *state changes*, as in general, it is the accumulation of observations about (possibly unexpected) state changes that, in the end, triggers a knowledge change.

What remains out of this characterization is the third type of changes we have considered, i.e., *requirement changes*. We are aware that these changes do occur during the system's lifespan and that the ability of a system to cope with them successfully can contribute to building our trust in it. However, at present, we have doubts about the suitability of antifragility as an attribute that characterizes "success" in coping with requirement changes. Other attributes could be more suitable to capture relevant aspects of this success (see, for example, the work in the area of technical credit (Berenbach, 2014)). We have yet to reach a firm position about this point and thus prefer to leave it as an open issue requiring further consideration.

8.2. Design space growth

One of the research challenges in model-based analysis is the growth of the design space that needs to be explored. In the reference model presented in Section 6.1, the antifragile system faces this challenge at least twice: in the upper layer *change triggering* that modifies M and in the *learning and exploitation*.

The change triggering mechanisms in the upper layer generate data about the system quality in a set of unprecedented circumstances to, subsequently, learn from them. If the change triggering is guided by humans, the design space size is limited to the ranges of the variables the human desires to explore. A larger challenge comes when the change triggering is autonomous. The objective is that this autonomous mechanism unveils something that is currently unknown, and thus, it should not only explore variants whose elements hold a tiny difference with known environment and system situations. However, as an engineered mechanism under some ground assumptions, it is designed with limits in the types of variants it can create (e.g., the types of models that the metamodel supports); it cannot explore situations that were totally unexpected in the design. Referring to Section 4.1, this mechanism can resolve Unknown–Knowns since it produces data on circumstances that can be represented by the current models, but it cannot assist in the obscure task of exploring variants of pure Unknown–Unknown circumstances. Nevertheless, with careful design of the change triggering, it is possible to partially penetrate the Unknown–Unknowns quadrant, as motivated in Garlan (2021) (e.g., using the "assume less" strategy introduced in Garlan (2021) which, in our running example, would make less assumptions about prior knowledge of the attack effects and vulnerabilities). The latter possibility brings great benefits since it enables becoming antifragile to unanticipated circumstances, but it also exacerbates the challenge of the size of the design space to explore.

The learning and exploitation mechanisms carry out the system improvement. If the middle layer makes decisions using machine learning models, the improvement may refer to finding knowledge from the new data in the explored configurations in terms of further training the used learner. These learners encompass modern artificial neural networks and more traditional rule-based learners, such as decision trees. In this case, the challenges in these fields, such as under and over-fitting or changing data formats, affecting the model size, appear. The system improvement may also refer to an optimization problem in terms of modifying some system elements to increase the system overall quality, now also including the newly explored scenarios. This optimization may suffer again from the problem of exploring the design space. The use of heuristics, like genetic algorithms, attempts to overcome the challenge and still find the optimal result.

8.3. Satisfaction function

We have introduced in Section 4.3 the concept of *satisfaction function* $Q_R(M)$, as an overall measure of the system dependability, underlining its double role: on the one side, it should provide an assessment of the "distance" of a given system (modeled by M) from the fulfillment of the requirements stated in R; on the other side, it should also provide information about the "precision" (uncertainty) of this assessment, as both aspects underpin our trust in a system. We have also given in *Example* 2 a possible instantiation of $Q_R(M)$. Still, we would like to remark that it had only exemplification purposes, giving a very simple concrete example of our general idea of satisfaction function. Devising a suitable $Q_R(M)$ for a real system is a far more complex task, requiring contributions from different fields, which include:

- Methodologies to estimate the set of "observables" involved in the definition of R. In the case of an existing system, these methodologies can be based on data collected on the field, which, depending on domain-specific situations, could require to be suitably cleaned and elaborated. In the case of a still-to-be system, they should rely on data derived from other sources, like suitable simulation models. In both cases, more or less sophisticated analytic or simulation models could be necessary to infer from these data the observables' value;
- Methodologies that, given an estimate of the observables' value, assess their distance from the acceptable region identified by R. Several proposals about this issue have emerged in the recent past, concerning, e.g., the assessment of measures of the partial fulfillment of the requirements in R or of the distance from some target values (e.g. Lacerda et al., 2019; Klös et al., 2018b);
- Methodologies that allow the explicit annotation and estimation of the confidence related to the assessment of the satisfaction function $Q_R(M)$. To this end, the adoption of languages and data types that allow the representation of measurement uncertainties,

for example, with the addition of the degree of *belief* associated with that measurement as in Bertoa et al. (2020), Troya et al. (2021) or using ranges, distribution probabilities, or confidence intervals, could represent a possible solution to this issue;

- Methodologies to suitably combine the two "sides" of our $Q_R(M)$ concept, i.e., "distance" and "precision" of the assessment: referring again to *Example* 2, we have given there the idea of combining these two sides through a simple product of separate measures of these two aspects. Also, in this case, we remark that this choice has only exemplification purposes, and it does not mean in any way that we are suggesting its adoption as a general methodology to get an overall single figure of merit.

9. Conclusion

We have presented a conceptual characterization of antifragility, funded on an extension of the dependability taxonomy (Avizienis et al., 2004). In this characterization, antifragility is an *attribute* that captures the ability of a system to cope with a particular type of *changes* (knowledge changes), taking advantage of their occurrence. We have then suggested change triggering and exploitation as a suitable *mean* to achieve this ability.

Then, based on this characterization, we have delineated a reference model for antifragile systems, have shown how recently proposed architectures for advanced self-adaptive systems can fit this reference model, and how an emerging technology such as Digital Twin can foster their realization.

As a next step of this work, we plan to tackle the issue of how we can assess the "antifragility degree" of an ICT system. Core elements of this issue are the definition of suitable antifragility metrics and methodologies for their evaluation. In this respect, we note that such metrics should allow us to quantify to what extent a system is able to cover two distinct facets of antifragility, according to the characterization we have given in Section 5: (i) the system's ability to learn (extract new knowledge) from the data it collects, and (ii) the ability to apply this knowledge to evolve and improve itself.

On a broader perspective, we recall that the result of the antifragility assessment is one of the elements that may concur with the assessment of an overall satisfaction function $Q_R(M)$, which we have suggested in Section 4.3 as overall dependability measure for an ICT system. We have discussed in Section 8.3 some issues concerning this broader goal, which represent further lines of research we plan to investigate.

CRediT authorship contribution statement

Vincenzo Grassi: Conceptualization, Investigation, Methodology, Visualization, Writing – original draft, Writing – review & editing. **Raffaela Mirandola:** Conceptualization, Investigation, Methodology, Visualization, Writing – original draft, Writing – review & editing. **Diego Perez-Palacin:** Conceptualization, Investigation, Methodology, Software, Visualization, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work has been partially founded from the topic Engineering Secure Systems of the Helmholtz Association (HGF) and by KASTEL Security Research Labs, and from the Swedish Knowledge Foundation with Grant No. 20200117: ALADINO – ALigning Architectures for DIgital twiN of the Organization.

Appendix A. A short discussion about resilience

As mentioned in the Introduction, *resilience* is another term, besides *antifragility*, that has gained much hype in recent years when discussing dependable systems. We have focused on antifragility, but we briefly expose here some considerations about this other term and its role in the discourses about dependability.

The role of this term was already considered by A. Avizienis in his paper about what he calls the "jungle of terminology" (Avižienis, 2017). The conclusion he arrives at in that paper is that this term, together with others he considers, does not bring with it new perspectives or concepts that were not already included in the dependability taxonomy (Avizienis et al., 2004). We share this opinion, and remain convinced that taxonomy still represents a fundamental contribution to the conceptual clarification in the dependability field, so that new concepts and ideas possibly emerging in this field would benefit from positioning themselves with respect to it. For what concerns in particular the *resilience* term, we recognize two different flavors in the use of this term within the existing literature about ICT systems that must cope with changes.

On the one side, from a rather narrow perspective, it is used to denote some specific means that can be adopted to cope with faults that can affect the system. This perspective is consistent with the definition already given in Avizienis et al. (2004), where resilience is defined as a synonym of *fault tolerance* and hence considered as one of the *means* that can be used to achieve dependability. This same perspective underlies other definitions that can be found in technical literature, see, e.g., Vettor and Smith (2023), where resilience is defined as " [...] the ability of a system to react to failures and still remains functional." and some specific design patterns to achieve it are suggested.

On the other side, the *resilience* term is sometimes used from a broader perspective, encompassing the generic ability of a system to cope with changes by whatever means. This second perspective is in line with the general definition given in Laprie (2008): *"Resilience is defined as the persistence of dependability when facing changes"*.

This double perspective recalls the problem faced by the authors of the dependability taxonomy with respect to the use of the *reliability* term, which was then used to denote both the broad concept of a desirable property and the precisely defined mathematical concept of continuous service delivery in a given time interval (Avižienis, 2017). In that case, this overload of meanings was solved by introducing the term *dependability* for the broad concept, thus leaving reliability to denote the second narrower concept.

A similar solution should probably be also devised for the use of the resilience term. To this end, our suggestion is to make resilience denote the broad concept, while more specific (and often already existing) terms should be used in the other cases.

Appendix B. Antifragility tractability: Experiments with the running example

This appendix describes a set of experiments based on the running example to study the cost of running an implementation of the *learning and exploitation* mechanisms for that system. The objective of this experiment is to obtain some preliminary insights, albeit limited to the considered example, about the complexity of running antifragile systems.

To this end, we measure how much time it takes to perform the *learning and exploitation* after the *change triggering* has revealed beneficial a system evolution. This means that this latter task has triggered a change, created the modified representation of the system and its environment, a Simulator has calculated the consequences of the change, and the result is that the system does not satisfy the requirement on *keeping, on average, at least 95% of nodes healthy*.

For the experiments, we have developed in Java the following modules:

- Model: We have developed a representation of the running example.
- Simulator: We have developed a simulator of the system dynamic behavior. It includes the probability of an attack happening in a given moment, the probability of a node healing in a given moment, and the propagation of the infection from an infected node to its neighbor nodes of the same type. At each time step, the simulator calculates the current proportion of healthy nodes and the average of healthy nodes since the beginning of the simulation.
- Learner: We have developed a learning module that explores the quality of candidates for the system, and if it finds a candidate that improves enough the system quality, it returns it.

Since this appendix explores the tractability of the design space exploration problem, we provide more details of the learner implementation. The learner module considers the system structure of nodes and connections as a connected undirected graph $G = (V, E)$ where the set of vertices V corresponds to the nodes and the set of edges E corresponds to the connections. An edge e is represented by its connecting vertices $\langle v_i, v_j \rangle$. The generation of system alternatives is based on the removal of edges from E. The search for a better candidate is developed considering that having many connections in the system is a positive characteristic because it allows faster communication between nodes. However, it also considers that removing some connections could be a good solution if the number of connections is high enough to propagate infections easily and threaten the satisfaction of the requirement.

A connection $e = \langle v_i, v_j \rangle$ must satisfy some conditions to become a candidate for removal: (a) the graph $G' = (V, E \setminus \{e\})$ must remain connected, (b) both v_i and v_j must have at least two edges in G', and (c) the installed software in nodes v_i and v_j must be the same (otherwise e is not propagating any infection). All the alternatives explored by the learner have fewer connections than the original system. To make the infection propagation slower, it is considered a good greedy heuristic strategy to start pruning the connections of the nodes with the highest number of connections, i.e., reduce the size of the nodes that are "hubs" to avoid infection super-spreaders.

After finding a possible candidate, the learner calls the Simulator module, which calculates the average percentage of healthy nodes. If the requirements are not met yet, the learner iterates and finds the next candidate G'', which will have one edge less than G'.

To observe an intensive execution of the learner, we create systems where the improvement activity needs to apply a significant number of iterations to reach a candidate system that satisfies the requirements. In this respect, we noticed that several systems with n nodes, $n/10$ software types, $n/5$ average neighbors per node, 0.005 for the probability of attack in a given moment, and 2.5 time units as average healing time tended to show an average percentage of healthy nodes ~65%. We consider that evolving a system with 65% of healthy nodes to a system that can keep more than 95% of them healthy is an improvement that intensively exercises the learner module. We experiment with this improvement for 20 different system sizes, from n=20 nodes to n=400, in increments of 20. In the experiments, we create ten random system configurations for each system size to mitigate the effects of particularly favorable or unfavorable software types and connections configurations.

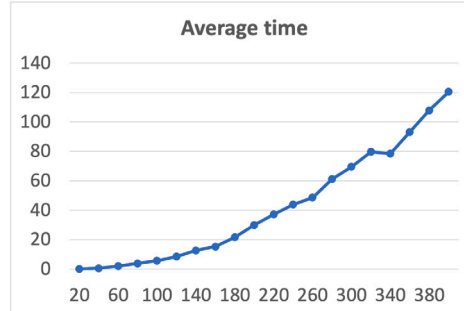

Fig. B.10. Execution times of the learning mechanism for different sizes of the running example.

Fig. B.10 shows the experiment results, depicting a relation between the system size and the execution time needed to find a suitable candidate that satisfies the requirements. This time is less than a second for systems of up to 40 nodes, and it grows up to 120 s, on average, for systems of 400 nodes. The figure shows that the execution time grows faster than linearly. More precisely, we can observe from the numeric values that the execution time for problems of size $2n$ is between 3.9 and 5.9 times larger than the execution time for size n. That gives us an approximate time complexity between n^2 and $n^{2.5}$ in the observed values. We expected that the execution time would grow at least quadratically with the system size in the current implementation since the problem size affects the time needed for the simulation of each candidate and also affects the number of candidates that the learner needs to generate (i.e., the number of iterations in the learner).

From these results, we can observe that, for our example, the cost of the *learning and exploitation* task may or may not be affordable depending on the system size. This indicates the need to carefully select the actual mechanisms used to implement this task, whose cost should be traded off with the benefits of achieving the antifragility property. Investigating this issue beyond the limited scope of our running example is part of our planned future work.

References

Andersson, J., Grassi, V., Mirandola, R., Perez-Palacin, D., 2021. A conceptual framework for resilience: fundamental definitions, strategies and metrics. Computing 103 (4), 559–588. http://dx.doi.org/10.1007/s00607-020-00874-x.

Avižienis, A., 2017. A visit to the jungle of terminology. In: 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W). pp. 149–152. http://dx.doi.org/10.1109/DSN-W.2017.32.

Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C.E., 2004. Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Sec. Comput. 1 (1), 11–33.

Bakhouya, M., Gaber, J., 2014. Bio-inspired approaches for engineering adaptive systems. Procedia Computer Science 32, http://dx.doi.org/10.1016/j.procs.2014.05.503.

Baruwal Chhetri, M., Uzunov, A., Vo, Q.B., Kowalczyk, R., Docking, M., Luong, H., Rajapakse, I., Nepal, S., 2018. AWaRE - towards distributed self-management for resilient cyber systems. In: 23rd International Conference on Engineering of Complex Computer Systems. ICECCS, pp. 185–188. http://dx.doi.org/10.1109/ICECCS2018.2018.00028.

Baruwal Chhetri, M., Uzunov, A., Vo, B., Nepal, S., Kowalczyk, R., 2019. Self-improving autonomic systems for antifragile cyber defence: Challenges and opportunities. In: IEEE International Conference on Autonomic Computing. ICAC, pp. 18–23. http://dx.doi.org/10.1109/ICAC.2019.00013.

Basiri, A., Behnam, N., de Rooij, R., Hochstein, L., Kosewski, L., Reynolds, J., Rosenthal, C., 2016. Chaos engineering. IEEE Softw. 33 (3), 35–41. http://dx.doi.org/10.1109/MS.2016.60.

Berenbach, B., 2014. On technical credit. Procedia Comput. Sci. 28, 505–512. http://dx.doi.org/10.1016/j.procs.2014.03.062, URL https://www.sciencedirect.com/science/article/pii/S1877050914001252.

Bertoa, M., Burgueño, L., Moreno, N., Vallecillo, A., 2020. Incorporating measurement uncertainty into OCL/UML primitive datatypes. Soft. Syst. Model. 19, http://dx.doi.org/10.1007/s10270-019-00741-0.

Bouchenak, S., Boyer, F., Claudel, B., De Palma, N., Gruber, O., Sicard, S., 2011. From autonomic to self-self behaviors: The JADE experience. ACM Trans. Auton. Adapt. Syst. 6 (4).

Casti, J., 2011. Four faces of tomorrow. OECD Int. Futures Proj on Future Glob. Shock.

de Bruijn, H., Größler, A., Videira, N., 2020. Antifragility as a design criterion for modelling dynamic systems. Syst. Res. Behav. Sci. 37 (1), 23–37. http://dx.doi.org/10.1002/sres.2574, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sres.2574.

de Lemos, R., Camara, J., Cerqueira, R., Kaaniche, M., 2013. Dependability in self-adaptive systems: How to justify trust in the face of "unknown unknowns"?. LADC 2013 WDAS Panel, Accessible at http://www2.dc.ufscar.br/~delano/WDAS/slides/deLemos_panelWDAS.pdf.

Garlan, D., 2021. The unknown unknowns are not totally unknown. In: International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS, pp. 264–265. http://dx.doi.org/10.1109/SEAMS51251.2021.00047.

Gheibi, O., Weyns, D., 2022. Lifelong self-adaptation: Self-adaptation meets lifelong machine learning. In: International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). ACM/IEEE, pp. 1–12. http://dx.doi.org/10.1145/3524844.3528052.

Ghezzi, C., 2016. Dependability of adaptable and evolvable distributed systems. In: Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems. Springer International Publishing, Cham, pp. 36–60. http://dx.doi.org/10.1007/978-3-319-34096-8_2.

Ghosh, D., Sharman, R., Rao, H.R., Upadhyaya, S., 2007. Self-healing systems—survey and synthesis. Decis. Support Syst. 42 (4), 2164–2185.

Gorgeon, A., 2015. Anti-fragile information systems. In: International Conference on Information Systems - Exploring the Information Frontier, ICIS. Association for Information Systems, URL http://aisel.aisnet.org/icis2015/proceedings/BreakoutIdeas/6.

Grassi, V., Mirandola, R., Perez-Palacin, D., 2023. Towards a conceptual characterization of antifragile systems. In: 20th International Conference on Software Architecture, ICSA - Companion. IEEE, pp. 121–125.

Guelfi, N., 2011. A formal framework for dependability and resilience from a software engineering perspective. Central Eur. J. Comput. Sci. 1 (3), 294–328. http://dx.doi.org/10.2478/s13537-011-0025-x.

Hashmi, S.S., Dam, H.K., Smet, P., Chhetri, M.B., 2022. Towards antifragility in contested environments: Using adversarial search to learn, predict, and counter open-ended threats. In: IEEE International Conference on Autonomic Computing and Self-Organizing Systems. ACSOS.

Hole, K., 2016. Anti-fragile ICT systems. Simula SpringerBriefs on Computing, Springer International Publishing.

ISO, 2021. Automation systems and integration — Digital twin framework for manufacturing (23247 Series). URL https://www.iso.org/standard/75066.html.

Kephart, J.O., Chess, D.M., 2003. The vision of autonomic computing. IEEE Comput. 36 (1), 41–50.

Klös, V., Göthel, T., Glesner, S., 2018a. Comprehensible and dependable self-learning self-adaptive systems. J. Syst. Archit. 85–86, 28–42, URL https://www.sciencedirect.com/science/article/pii/S1383762117304472.

Klös, V., Göthel, T., Glesner, S., 2018b. Runtime management and quantitative evaluation of changing system goals in complex autonomous systems. J. Syst. Softw. 144, 314–327. http://dx.doi.org/10.1016/j.jss.2018.06.076.

Kramer, J., Magee, J., 2007. Self-managed systems: an architectural challenge. In: International Conference on Software Engineering (ICSE), Workshop on the Future of Software Engineering (FOSE). IEEE Computer Society, pp. 259–268. http://dx.doi.org/10.1109/FOSE.2007.19.

Lacerda, B., Faruq, F., Parker, D., Hawes, N., 2019. Probabilistic planning with formal performance guarantees for mobile service robots. Int. J. Robot. Res. 38 (9), 1098–1123. http://dx.doi.org/10.1177/0278364919856695.

Laprie, J.-C., 2008. From dependability to resilience. In: Dependable Systems and Networks.

Perez-Palacin, D., Mirandola, R., 2014. Uncertainties in the modeling of self-adaptive systems: a taxonomy and an example of availability evaluation. In: 5th ACM/SPEC International Conference on Performance Engineering (ICPE). ACM, pp. 3–14. http://dx.doi.org/10.1145/2568088.2568095.

Perez-Palacin, D., Mirandola, R., Merseguer, J., 2014. On the relationships between QoS and software adaptability at the architectural level. J. Syst. Softw. 87, 1–17.

Perrouin, G., Morin, B., Chauvel, F., Fleurey, F., Klein, J., Le Traon, Y., Barais, O., Jézéquel, J.-M., 2012. Towards flexible evolution of dynamically adaptive systems. In: 34th International Conference on Software Engineering. ICSE, pp. 1353–1356. http://dx.doi.org/10.1109/ICSE.2012.6227081.

Ramezani, J., Camarinha-Matos, L.M., 2020. Approaches for resilience and antifragility in collaborative business ecosystems. Technol. Forecast. Soc. Change 151, 119846. http://dx.doi.org/10.1016/j.techfore.2019.119846.

Richardson, C., 2018. Microservices patterns: With examples in Java. Manning.

Roth, F.M., Krupitzer, C., Becker, C., 2015. Runtime evolution of the adaptation logic in self-adaptive systems. In: 2015 IEEE International Conference on Autonomic Computing. pp. 141–142. http://dx.doi.org/10.1109/ICAC.2015.20.

Russo, D., Ciancarini, P., 2016. A proposal for an antifragile software manifesto. In: Shakshuki, E.M. (Ed.), In: Procedia Computer Science, vol. 83, Elsevier, pp. 982–987. http://dx.doi.org/10.1016/j.procs.2016.04.196.

Russo, D., Ciancarini, P., 2017. Towards antifragile software architectures. Procedia Comput. Sci. 109, 929–934. http://dx.doi.org/10.1016/j.procs.2017.05.426.

Schmeck, H., Müller-Schloer, C., Çakar, E., Mnif, M., Richter, U., 2010. Adaptivity and self-organization in organic computing systems. ACM Trans. Auton. Adapt. Syst. 5 (3), 10:1–10:32.

Taleb, N.N., 2012. Antifragile: Things that gain from disorder. Random House, New York.

Troya, J., Moreno, N., Bertoa, M.F., Vallecillo, A., 2021. Uncertainty representation in software models: A survey. Softw. Syst. Model. 20 (4), 1183–1213.

Uzunov, A.V., Brennan, M., Chhetri, M.B., Vo, Q.B., Kowalczyk, R., Wondoh, J., 2021. AWaRE2-MM: A meta-model for goal-driven, contract-mediated, team-centric autonomous middleware frameworks for antifragility. In: 2021 28th Asia-Pacific Software Engineering Conference. APSEC, pp. 547–552. http://dx.doi.org/10.1109/APSEC53868.2021.00066.

Vettor, R., Smith, S., 2023. Architecting cloud-native .NET apps for azure. https://dotnet.microsoft.com/en-us/download/e-book/cloud-native-azure/pdf.

Walker, W., Harremoes, P., Romans, J., van der Sluus, J., van Asselt, M., Janssen, P., Krauss, M., 2003. Defining uncertainty. A conceptual basis for uncertainty management in model-based decision support. Integr. Assess. 4 (1), 5–17.

Weyns, D., et al., 2013. Perpetual assurances for self-adaptive systems. In: Software Engineering for Self-Adaptive Systems III. Assurances. In: Lecture Notes in Computer Science, vol. 9640, Springer, pp. 31–63.

Woods, D.D., 2015. Four concepts for resilience and the implications for the future of resilience engineering. Reliab. Eng. Syst. Saf. 141, 5–9, Special Issue on Resilience Engineering.

Zavala, E., Franch, X., Marco, J., Berger, C., 2020. HAFLoop: An architecture for supporting highly adaptive feedback loops in self-adaptive systems. Future Gener. Comput. Syst. 105 (C), 607–630. http://dx.doi.org/10.1016/j.future.2019.12.026.

Vincenzo Grassi is full professor of Computer Science at the University of Roma Tor Vergata. He received the Laurea degree in Computer Science with the highest honors from the University of Pisa, Italy, in 1984. His general research interests are in the field of methodologies and tools for the analysis and validation of non-functional quality indices (in particular, performance and dependability indices) for computing and communication systems. Within this general framework, he has recently focused his research activity on mobile computing and distributed service-oriented systems, and self-adaptive systems. He has more than 80 publications on these topics in international conferences and journals.

Raffaela Mirandola is Full Professor at Karlsruhe Institute of Technology (KIT), Germany. Her main research interests are in (i) Software quality requirements modeling, analysis, and verification, (ii) Formal methods for (self-)adaptive dependable IT systems, (iii) Model-driven Software engineering, and the application of the theories, approaches, and techniques specific to the above research areas to service-oriented and component-based systems, adaptive systems, mobile systems, and cloud computing. Her research has been supported by several National and European programs. She has been and is involved in the Program committee and in the organization of several conferences. Recently, she has been program co-chair for the International Symposium on Software Engineering for Adaptive and SelfManaging Systems-SEAMS 2021 and SEAMS 2023 and general co-chair for the European Conference on Software Architecture, ECSA 2021. She has been part of the Editorial Board of IEEE Transactions on Software Engineering and the Journal of System and Software, Elsevier. She is at present Special Issue co-editor for the Journal of System and Software, Elsevier.

Diego Perez-Palacin is currently a Senior Lecturer in the Computer Science and Media Technology Department at Linnaeus University, Sweden. He received the Ph.D. degree in Computer Science from the University of Zaragoza, Spain. Before joining Linnaeus University, he was a postdoctoral researcher at Politecnico di Milano, Italy, and Senior Researcher at the University of Zaragoza. His research interests are in the areas of self-adaptive Software, smart industry via the use of Digital Twins, and evaluation of Software quality properties such as performance, resilience, and antifragility; with a special interest in autonomous decisions under uncertainty, the use of formal methods and model-driven engineering.