J Reliable Intell Environ
DOI 10.1007/s40860-015-0016-0

ORIGINAL ARTICLE

Formalization of resilience for constraint-based dynamic systems

Nicolas Schwind[1,2] · **Morgan Magnin**[1,3] · **Katsumi Inoue**[1,4,5] · **Tenda Okimoto**[6] · **Taisuke Sato**[1,7] · **Kazuhiro Minami**[8,9] · **Hiroshi Maruyama**[8,9]

Received: 15 March 2015 / Accepted: 17 November 2015

Abstract Many researchers in different fields are interested in building resilient systems that can absorb shocks and recover from damages caused by unexpected large-scale events. Existing approaches mainly focus on the evaluation of the resilience of systems from a qualitative point of view, or pay particular attention to some domain-dependent aspects of the resilience. In this paper, we introduce a very general, abstract computational model rich enough to represent a large class of constraint-based dynamic systems. Taking our inspiration from the literature, we propose a simple parameterized property which captures the main features of resilience independently from a particular application domain, and we show how to assess the resilience of a constraint-based dynamic system through this new resilience property.

Keywords Resilience · Dynamic system · Constraint-based system · Resistance · Recoverability

1 Introduction

During the past two decades, human beings have been reminded that they can suffer, at any time, from potentially lethal events. At a macroscopic scale, major events can suddenly occur such as destructive earthquakes followed by unstoppable tsunamis (e.g., the 2004 Sumatra–Andaman earthquake and the 2011 Tohoku tsunami) and nationwide banking emergencies (e.g., the US subprime mortgage crisis [42]). Meanwhile, the recent progress in understanding

This paper is an extended and revised version of [38].

✉ Nicolas Schwind
schwind@nii.ac.jp

Morgan Magnin
morgan.magnin@irccyn.ec-nantes.fr

Katsumi Inoue
inoue@nii.ac.jp

Tenda Okimoto
tenda@maritime.kobe-u.ac.jp

Taisuke Sato
satou.taisuke@aist.go.jp

Kazuhiro Minami
kminami@ism.ac.jp

Hiroshi Maruyama
hm2@ism.ac.jp

1 Principles of Informatics Research Division, National Institute of Informatics, Tokyo, Japan

2 Transdisciplinary Research Integration Center, The Research Organization of Information and Systems, Tokyo, Japan

3 Institut de Recherche en Communications et Cybernétique de Nantes, LUNAM Université, École Centrale de Nantes, IRCCyN UMR CNRS 6597, Nantes, France

4 Department of Informatics, School of Multidisciplinary Sciences, SOKENDAI (The Graduate University for Advanced Studies), Tokyo, Japan

5 Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology, Tokyo, Japan

6 Graduate School of Maritime Science, Kobe University, Kobe, Japan

7 Artificial Intelligence Research Center (AIRC), The National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, Japan

8 Department of Statistical Modeling, The Institute of Statistical Mathematics, The Research Organization of Information and Systems, Tokyo, Japan

9 Department of Statistical Science, School of Multidisciplinary Sciences, SOKENDAI (The Graduate University for Advanced Studies), Tokyo, Japan

Published online: 09 December 2015

the microscopic world confronts us with the same kind of unpredictability that can lead to breaking changes (e.g., the mutation of the H1N1 influenza virus resulting in a global contagion). Through a better understanding of our environment, the development of modern societies (instantaneous connection throughout Internet, globalization of the economy, travellers going from one country to another in a glimpse) and the uprising of new research axes (e.g., the Industrial Revolution and the Internet), the scale of the systems we are addressing has gone much larger, and it has become crucial to design approaches that succeed in tackling such events.

Resilience is an attribute given to a system which is basically defined as the ability to cope with and recover from misfortune or external effects. The notion has first been introduced by Holling [23] in the context of social-ecological systems, but then has been adapted to a broad range of reliable systems and environments, e.g., engineering systems, computer networks, financial systems, civil infrastructures, organizations, society. Moreover, even within the same discipline, resilience is a key notion at different levels of the underlying domain; for instance, to ensure safety, security and reliability of computer networks, one should ensure resiliency at the physical level [4] (sensor networks), the microscopic level [30] (security protocols) and the macroscopic level [35] (security of the whole communicating network). Furthermore, prior to the step of improving the resilience of the system under consideration, it is essential to be able to assess how resilient it is given its current description. However, rather than being a property exhibited by the system itself, resilience is often interpreted as the emerging result of a *dynamic* process [11]. Thus, the challenges typically addressed in the literature consist in elaborating resilient strategies for a system through an analysis of its past behaviour. Such strategies are usually domain-dependent [26,27,40,41,50].

In this paper, we do not intend to address a specific real-world system, nor do we provide methods for the design of a resilient system. On the contrary, we focus on the *verification* of the behaviour of a system from the resilience viewpoint, which is a crucial stage prior to the design of a reliable system or Intelligent Environment. Our approach of computational resilience lies on a different layer of resilience from the point of view of Big Data Analysis and Machine Learning. The latter techniques could be used to understand and learn the structure of a system and its dynamics, i.e., to build a model itself; but our framework aims at taking benefits from that learned model once it is designed and to check whether it actually meets desirable specifications from the resilience point of view.

We provide a general, flexible computational framework which represents the system's specifications through a constraint optimization problem (COP) [1,24,29,37]. Doing so, the system's components are represented by variables, the possible states of these components correspond to the domain of these variables and interactions between the components are represented through constraints. This choice is motivated by the fact that one expects an Intelligent Environment to contain a number of networked devices [3], and several benefits arise from such a compact description. First, this allows us to take into consideration a large set of states possibly taken by the system. Second, each state can be simply associated with a measure (e.g., a cost or a reward) characterizing the system state's overall performance (e.g., workload or budget invested to maintain the state in the case of a cost, the transmission rate in a computer network, or the social welfare in the case of a reward). Then, through this measure and taking into consideration the common aspects of resilience discussed in the literature in different domains, we propose a new property which quantitatively assess the resilience of a system in a specific scenario, i.e., a succession of states followed by the system at different time steps. To describe the dynamics of the system, i.e., how it may evolve depending on the actions at hand and the exogenous events, we adapt the well-known formalism of discrete event dynamic systems (DEDS) [5,12,31,34] to our framework. The originality of our framework lies in the embedding of COPs (where a COP represents the system's specifications at a given time step) into a DEDS (which models the system's dynamics, i.e., how its specifications through a COP may change in response to exogenous events). Doing so, the property of resilience can be applied to the system independently of a specific scenario it may follow, accounting for all its "possible futures." Most importantly, our approach is highly domain-independent. It provides an abstract way to assess the resilience of a system given its current specifications and dynamics and is applicable to a large class of systems, from computer networks to social-ecological systems.

In the next section, we discuss some related work on constraint-based systems, dynamic systems and resilience in several disciplines and emphasize the position of this paper with respect to the literature. In Sect. 3 we formally introduce the notions of constraint-based systems and state trajectories. In Sect. 4 we review several concepts underlying resilient scenarios from the literature and introduce a restricted set of formal properties for state trajectories inspired from them, as well as a new, single parameterized resilience property capturing all concepts. In Sect. 5 we introduce our definition of dynamic systems and show how to extend the property of resilience from state trajectories to dynamic systems. We afterwards discuss in Sect. 6 some further related work and conclude with open questions that are important for future research.

2 Position of this paper with respect to related work

2.1 Resilience

Resilience can be informally defined as the ability "to maintain a system's core purpose and integrity in the face of dramatically changed circumstances" [8,23,28,46]. The term "resilience" was originally introduced by Holling [23]. He adopted a verbal, qualitative definition of ecological resilience, rather than a mathematical, quantitative one: "Resilience determines the persistence of relationships within a system and is a measure of the ability of these systems to absorb changes of state variables, driving variables, and parameters, and still persist" [23, p. 17]. More recently, Walker et al. [46] proposed the following revised formulation: "Resilience is the capacity of a system to absorb disturbance and reorganize while undergoing change so as to still retain essentially the same function, structure, identity, and feedbacks." Many variants of the notion of resilience were proposed in various fields. For instance in psychology, resilience is viewed as "an individual's ability to properly adapt to stress and adversity" [9]; in physics, resilience coincides with the notion of *buoyancy*, i.e., "the quantity of work given back by a body that is compressed to a certain limit and then allowed freely to recover its former size or shape" (New International Websters Comprehensive Dictionary 2014).

The concept of resilience has been considered as a key property of systems in various disciplines such as environmental science and risk management [20,51], computer networks [27,40,41], sociology or material sciences. In [40] resilience is considered for computer networks and is evaluated through the shape of the state trajectory followed by the network within a two-dimensional state space: after the system suffers from a degradation due to an exogenous event, the network goes from delivering an acceptable service under normal operations to degraded service; then there is a remediation step which improves the service on the one hand and a recovery step which allows the system to return to the normal state on the other hand. Such networks are resilient if the level of service during these different phases of resilience is maintained globally acceptable. In [26,27] the authors establish a "cyber resilience matrix" for cyber systems, i.e., a list of the main strategies to adopt at some stage of resilience in some given domain. On the one hand, they focused on four stages of resilience identified by the National Academy of Sciences (NAS) as the main steps of the event management cycle that a system needs to maintain to be resilient, i.e., prepare, absorb, recover and adapt. On the other hand, they considered four important domains identified by the Network-Centric Warfare (NCW): the physical, information, cognitive and social domains. Each cell of the matrix addresses the following question: "how is the system's capability to prepare to, absorb, recover from and adapt to a cyber disruption implemented in the physical, information, cognitive and social domains?" In other terms, these works provide hints on the abilities a system should demonstrate to be resilient when disruptions occur and stresses the importance of understanding and documenting the system structure and functions. In [41] Stoicescu et al. focus on computer systems and their awareness on fault tolerance. They propose a model where the system lies in a three-dimensional space (fault tolerance, system's resources and application assumptions) and where a fault tolerance mechanism should be available given any state of the system. In other terms, they consider that a set of strategies covering the whole three-dimensional state space should be prepared in advance so that to ensure the system's reactivity to adversity. In IT security, some recent works have been conducted [50] which aim at improving IT systems in terms of adaptation to both security vulnerabilities and reliability of information.

Further domain-specific aspects of resilience have been proposed, in organization science, human ecology, civil engineering and computer science (see, e.g., [11]). However, most of the existing related works are concerned with qualitative characterizations of resilience, or with the elaboration of domain-dependent, proactive and reactive strategies for the design of a resilient system. Moreover, interpretations of resilience can be numerous, even within a specific domain; for instance, Grimm and Wissel [19] reviewed 163 definitions of resilience and stability concepts from ecology. Our goal in this paper was to provide a single, unifying parametrized property which we simply name *resilience*. We assume that the state of our system can be associated with a measure characterizing its "quality." This measure can be interpreted as the quality of service (QoS) provided by the system in a given state; for instance, in the field of computer networks it can represent a combination of different parameters, e.g., bandwith, latency and reliability. When the system follows a specific scenario, it can be associated with a trajectory of rewards (or costs). Our property evaluates how resilient the system has been with regard to that trajectory; in other terms, it allows us to assess the performance of the system's *dynamics* given a series of measures. Our formal definition of resilience captures by itself the properties that have been identified as important in the literature regarding resilient behaviours [38]:

- *resistance*, which is the ability for the system to absorb by itself drastic modifications of the environment. It deals with the system's inherent response to some perturbation. This is assessed under the ability to maintain some underlying costs under a certain "threshold," such that

the system satisfies some hard constraints and does not suffer from irreversible damages;
- *recoverability*, which is the ability to reach an admissible state within a given time interval after potentially damaging modifications. This property allows the system to express temporarily the consequences of some external disturbances, but then it should be able to return to a satisfactory state. This is assessed under the ability to recover to a baseline of acceptable quality as quickly and inexpensively as possible;
- *functionality*, which is the ability to provide a guaranteed average degree of quality for the state trajectory under consideration. Through this property, the core functionality of the system is measured *in the long run* while preventing disturbances that may destabilize the system.

Indeed, we will show that these notions can be simply expressed through our property of resilience by adjusting its parameters accordingly.

2.2 Constraint-based systems

The constraint satisfaction problem (CSP) framework [2,13,36] is a declarative paradigm to represent and reason about the components of a given system and the interactions between these components. CSPs have been successfully applied in many domains of Artificial Intelligence, including Operations Research (scheduling, vehicle routing, timetabling), bioinformatics (DNA sequencing) and resource allocation. A CSP is formed of a set of variables which can take their values within a given domain and constraints between the variables. The problem typically consists in finding a solution to the CSP, i.e., an assignment of the variables to values such that all constraints are satisfied. The constraints involved in CSPs can represent temporal or tangible constraints, e.g., making sure we do not go over the budget within a certain amount of time. What makes CSPs a standard representation paradigm is the structure of the system or problem to be represented. Unlike many AI problems, there is a standard structure to CSPs that allows general methods for finding a solution to a problem, using search algorithms and heuristics (with knowledge about the structure of the problem and not necessarily domain-specific knowledge) implemented for any CSP.

CSPs have been extended to COPs [1,24,29,37], where an additional objective function is considered to evaluate the "quality" of each solution. The constraints in a COP are augmented with quantitative preferences (in terms of cost or rewards), making some solutions preferable than others. The goal consists in finding a solution with the minimal cost (or the maximal reward).

In our framework, we consider any system which can be described using a COP. This choice allows us to represent the specifications of a system in a much more compact way than through the exhaustive description of its states (i.e., of its solutions). This is useful to model the possible configurations that a system can take according to a set of constraints with costs or rewards, which are meant to provide each configuration of the system with a measure, e.g., describing the QoS provided by the system within this configuration. Indeed, this measure will be a key ingredient to analyse the evolution of a system from the point of view of resilience.

2.3 Dynamic systems

Though COPs provide us with a standard and succinct language for representing a system, it still does not allow us to describe its dynamics. In the literature, CSPs have been extended to Dynamic CSPs [14]. Dynamic CSPs are useful to represent problems which are altered in some way, typically when the constraints evolve because of the environment [45]. Dynamic CSPs can be represented as a sequence of (static) CSPs, where each CSP in the sequence is a transformation of the previous one in which variables and constraints are added or removed. They also can be viewed as an initial CSP with a set of given events which are expected to occur with a certain probability [10]; in the case where an event occurs, some constraints are added to the CSP at the next time step, restricting the possible set of solutions. Most of research on Dynamic CSPs consist in developing efficient methods to compute a sequence of solutions. This is done, for instance, by reusing any previous solution computing in the sequence and producing the next one by local modifications (i.e., by changing the values of a restricted set of variables) [45]. Natural extensions from Dynamic CSPs to Dynamic COPs have been recently proposed [39], which also focus on efficient computational procedures to find a sequence of solutions which all optimize the underlying objective functions. To the best of our knowledge, the evaluation of systems from the resilience viewpoint has been unaddressed in the literature so far, when these systems are represented through COPs.

An other important drawback of existing approaches is that the sequence of COPs is usually assumed to be known in advance, which is an unrealistic assumption when considering how unpredictable is our environment and its possible drastic effects on our systems. DEDS [32] cope with such kind of unpredictability through non-deterministic "actions" performed by an agent. DEDS are systems whose states evolve over time via instantaneous, non-deterministic transitions due to instantaneous events. Numerous frameworks exist to model such systems, e.g., automata [6] which allow for compact representation of automata properties, or Petri

nets, Markov chains and Markov decision processes [33] which account for stochastic transitions to be tackled.

It has recently been suggested in [44] that combining CSPs and DEDS seems to be a promising approach for modelling a system and its dynamics. In [44] the authors have shown that the genericity and flexibility of such constraint-based dynamic systems allow them to subsume many existing frameworks, including automata, Petri nets and other classical frameworks used in planning and scheduling. Despite the recent interest of researchers into such a generic framework, the evaluation of constraint-based dynamic systems from the point of view of resilience is still an open issue that we wish to address in this paper. Moreover, no framework has been proposed so far which combines the structure of COPs (to represent a system) and DEDS (to describe the system's dynamics).

2.4 Outline of our model

We choose here to describe a system and its dynamics by combining COPs and DEDS. This allows us to serve the main purpose of this work, i.e., to propose a general and flexible model that is applicable to a broad range of reliable environments and at different scales and to assess the resilience of the represented systems from the resilience point of view.

As we are interested in resilience, we aim at capturing an effect of drastic modifications applied to the system. This is performed by introducing exogenous events that force the system's specifications to change with respect to some (non-deterministic) action, thus allowing us to model the full dynamics of the system. The way we build a unifying constraint-based framework through actions that force the system to modify its configuration can be compared to another unifying work, that is about hybrid systems. These ones were extensively explored to study models combining discrete event dynamics with nonlinear continuous dynamics [43]. The advantage of putting constraints at the very core of our modeling is that it is a very concise, yet expressive, method to define system states and its dynamics [44]. It offers a framework expressive enough to study a wide range of dynamic properties while capturing the intrinsic complexity of the system.

Proactiveness is an important desired feature of an Intelligent System, at least as important as reactiveness [3]. For the sake of dependability, it is important for an intelligent agent to take (or advise) decisions at each time step which will lead the system to exhibit a resilient behaviour. Hence, observing the system is one thing, but making it evolve in a way that preserves some key resilience properties in response to damaging events is the real deal here. In other words, this raises a control problem [34]. In our framework, the modeling considers two layers; thus control has to be considered at these two levels. This approach is close to the idea of hierarchical control [52]. The first level of control consists in tuning the system's configuration at a given time step, i.e., given the constraint-based specifications of the system. This operation is called here the *configuration of the system*. Then, since the system is allowed to be modified over time, there is a second level of control which handles the actions to perform in reply to exogenous events. Doing so, the system's specifications (i.e., its variables, constraints and thus costs or rewards associated with states) can evolve based on decisions made at this second level of control in response to outside environmental events. This operation is called here the *design of the system's specifications*. In this paper, we assume that these two levels of control are performed by the same agent (an intelligent software make or a human supervisor) which we call here the *system's controller*. Because of the possible occurrence of exogenous events (i.e., that makes the evolution of the system not fully under control), the design of the system's specifications consists in performing an action that is non-deterministic in the general case. Many *system trajectories*, which represent the possible scenarios describing the evolution of a system through time, have to be considered. Then, given a system trajectory (i.e., one specific scenario), the system's controller can build a so-called *state trajectory* by configuring each system from the system trajectory. Intuitively, a state trajectory represents a possible evolution of the constraint-based system together with its specific configuration at each time step. This is at the level of such state trajectories that our central property of resilience is introduced.

Since our property of resilience is introduced on the level of *state trajectories*, one is asked now about evaluating the resilience of the constraint-based dynamic system itself. This is based on the notion of a "strategy" given to the system's controller, a standard notion in the DEDS framework. At each time step, given the current constraint-based description of the system: (i) the system's controller configures the system, i.e., it chooses a specific configuration for the current system; (ii) the system's controller chooses a non-deterministic action among the available ones (this corresponds to the design of the system's specifications); (iii) depending the action performed in (ii), the environment selects one of the possible alternatives for the next system's specifications, thus defining the next constraint-based description of the system. Being faithful to the concept of resilience as it is described in most related works, we define a resilient dynamic system as follows: a dynamic system is resilient if one can guarantee that there will be a strategy for the system's controller such that any possible state trajectory followed by the dynamic system will be resilient.

As a first step towards the analysis of real-life constraint-based systems from a resilience viewpoint, our model induces several assumptions summarized as follows: The constraint-

based specifications of the system are fully observable at any time point; the resilience of the system reflects the worst case and relies on its dynamics which are given: all possible events are listed (they are represented implicitly in the non-determinism of actions), and the set of possible consequences of each event is completely known (though we do not assume to know which one of the consequences will actually hold, since the actions are not deterministic); every configuration of the system can be associated with a "cost" or "reward" value, which can be done with no harm through the compact representation of a system as a COP.

All along the paper, we will illustrate our notions on a simple example of management of servers in a web service company in a data center, with underlying natural challenges consisting in adjusting resources in a proactive way to meet fluctuating and possibly unpredictable business demand.

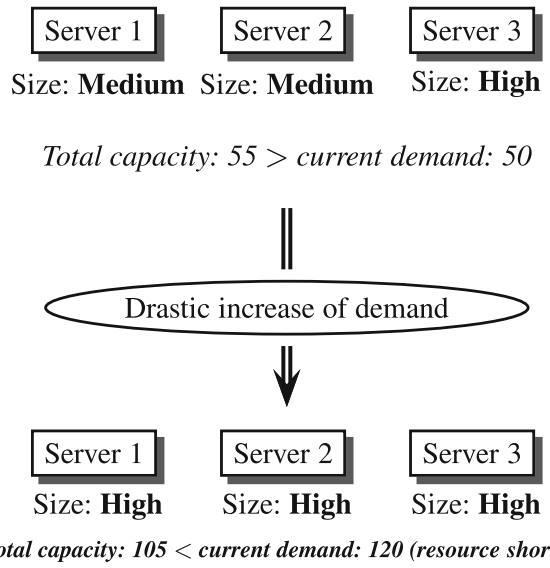


Fig. 1 Illustration of a simple scenario of resource shortage resulting in a sudden increase of demand

3 Formalisation of a constraint-based system

A system is composed of a set of variables representing the components of interest. Each one of these variables can be assigned to some value ranging over some specific *domain*. For instance, a variable can represent a server within a network, and the set of possible values of this variable would correspond to its possible states, e.g., {on, off}. An assignment of all system's variables to some specific values is called a *configuration*, and a configuration is associated with some "cost value," conforming to some constraints inherent to the system. Before introducing formal definitions, let us introduce a simple, intuitive example of a system that will serve as a support along the paper.

Example 1 (*Management of servers in a web service company*) Let us consider a web service company which provides IT resources to its customers using its own sets of servers. The company should manage dynamically the number of servers together with their size allocation in an efficient manner, depending on the demand of its customers. It can buy new servers or dispose of existing ones from the data centre, scale up the servers as the users' computing needs increase and scale down again as demands decrease. A given configuration of such servers engenders a global "cost" representing an efficient use of these servers; more precisely, this cost results from the aggregation of two factors: (i) the price charged to the company, engendered by each server and depending on its size; (ii) the (servers') resource shortage, in case the demand from the customers is not appropriately fulfilled. Here the web service company constitutes our system, and each one of the servers managed by the company induces a variable whose domain is the granted memory size (low, medium or high). The natural challenges the company is faced with consists in adjusting resources to meet fluctuating and possibly unpredictable business demand. Figure 1 below illustrates a simple scenario of resource shortage resulting in a sudden increase of demand for which the company was not prepared.

This example could be enriched in many ways depending on the actual options the company has at hand. For instance, one could consider that the company could rent additional resources from other companies in case of resource shortage. Renting new servers may nevertheless involve an implementation delay, and without appropriate preparation the company may be unable to avoid resource shortage in case of a quick, drastic increase of demand.

We would like to stress the following important point about our running example: Some works investigate the resilience of data centre networks from different criteria, e.g., reliability or robustness [7]; these works focus on the analysis of the structure of the network to assess such properties. On the contrary, in our framework we rather focus on the "observation" of the behaviour of a system and intend to analyse the resilience of a system given its possible evolutions, regardless of its structural properties. Our choice of a web service company as an example is motivated by the fact that the cost involved in the configuration of such systems is a key issue [17].

We are now ready to introduce the formal definition of a *system* which is defined through a COP [13,37].

Definition 1 (*System*) A *system* is a tuple $\mathcal{S} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ where

- $\mathcal{X} = \{X_1, \ldots, X_n\}$ is a finite set of variables;

- $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a multiset of non-empty sets; each D_i is called the *domain* of the variable X_i, i.e., it represents the set of possible values for the variable X_i;
- $\mathcal{C} = \{C_1, \ldots, C_m\}$ is a finite set of constraints; more precisely, each constraint $C_j \in C$ is a mapping from some specific set of domains $\mathcal{D}(C_j) \subseteq \mathcal{D}$ to $\mathbb{R}^+$; note that each constraint C_j involves a subset of variables from $\mathcal{X}$, called the *scope* of C_j and denoted by $scope(C_j)$; intuitively, a constraint C_j associates a "cost" with a partial assignment of the variables from its scope $scope(C_j)$.

Let $\mathcal{S}$ be a system $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$. A *configuration* α of $\mathcal{S}$ is a mapping associating each variable $x_i \in X$ with a value $\alpha(x_i) \in D_i$. The set of all possible configurations of $\mathcal{S}$ is denoted by $\Omega(\mathcal{S})$. The *cost* of a configuration $\alpha \in \Omega(\mathcal{S})$, denoted by $cost(\alpha)$, is defined as

$$cost(\alpha) = \sum_{C_j \in \mathcal{C}} C_j(\alpha(x_{j_1}), \ldots, \alpha(x_{j_k})), \quad (1)$$

where for each constraint $C_j \in \mathcal{C}$, $scope(C_j) = \{x_{j_1}, \ldots, x_{j_k}\}$. This means that the computation of the cost of a configuration α is performed through the summation of all associated "local costs" specified by the constraints from $\mathcal{C}$. Last, when $\mathcal{S}$ is a system and α is a configuration of $\mathcal{S}$, then the pair $(\mathcal{S}, \alpha)$ is called a *system state*.

We want to stress the fact that such a definition of a system allows one to describe it *statically*, i.e., a system is characterized in terms of its specifications at a given time point. That is, dynamic issues remain unaddressed so far. Let us now consider again our example about the service company.

Example 2 (*Continued*) We denote by $\mathcal{S}$ the system $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ representing the set of servers. Let us consider that three servers are ready for use, i.e., $\mathcal{X} = \{x_1, x_2, x_3\}$ is the set of variables associated with the three servers. We set $D(x_1) = D(x_2) = D(x_3) = \{Low, Medium, High\}$ the domain of each variable, where each one of these values qualitatively represents the size allocation of each server. For the sake of simplicity, we represent a configuration of $\mathcal{S}$ as a chain of letters among the set $\{L, M, H\}$, e.g., $\alpha = MMH$ means that the two first servers are given a medium size allocation and the third server is given a high size allocation. Then one needs to define the set of constraints $\mathcal{C}$ which will be used when computing the cost of any configuration as given in Eq. 1 above. One can consider here two types of constraints representing (i) the price for each server depending on its size, which is, therefore, characterized by the configuration and the market, and (ii) the (servers') resource shortage depending on both the configuration and the current demand.

Table 1 lists each server's available memory and price depending on its value (low, medium or high). The memory and price are given as arbitrary units. To express the price of the servers using constraints, we use one constraint for each server, i.e., three constraints C_1, C_2, C_3 in total, where $scope(C_1) = \{x_1\}$, $scope(C_2) = \{x_2\}$ and $scope(C_3) = \{x_3\}$ and for each variable x_i, $C_i(\alpha(x_i)) = Price(\alpha(x_i))$. For instance, for the configuration $\alpha = MMH$, the total price will correspond to $C_1(\alpha(x_1)) + C_2(\alpha(x_2)) + C_3(\alpha(x_3)) = 3 + 3 + 9 = 15$.

Table 1 Available memory and price for each server in $\mathcal{S}$

$\alpha(x_i)$	$Memory(\alpha(x_i))$	$Price(\alpha(x_i))$
Low	2	1
Medium	10	3
High	35	9

Let us now define a fourth constraint C_4 which will stand for the resource shortage. One can associate for simplicity with our system $\mathcal{S}$ an additional constant $Demand$ representing the current demand from the customers (in memory units). Then the resource shortage corresponds to the (weighted) difference between the demand and the total available memory, i.e., $C_4(\alpha(x_1), \alpha(x_2), \alpha(x_3)) = \omega_1 . Demand - \max(0, \omega_2 . \sum_{x_i \in X} Memory(\alpha(x_i))$, where ω_1, ω_2 are weight factors. Note here that this constraint involves all three variables, i.e., $scope(C_4) = \{x_1, x_2, x_3\}$. For instance, let us set $\omega_1 = 1$ and $\omega_2 = 5$, i.e., satisfying the demand is considered as a more critical matter than the budget involved in the servers. Consider the case where the demand corresponds to 48 memory units, i.e., $Demand = 48$. Then we get that $cost(MMH) = C_1(\alpha(x_1)) + C_2(\alpha(x_2)) + C_3(\alpha(x_3)) + C_4(\alpha(x_1), \alpha(x_2), \alpha(x_3)) = 3 + 3 + 9 + \max(0, 5 \cdot (48 - (35 + 10 + 10))) = 3 + 3 + 9 = 15$. One can remark that this configuration involves no resource shortage, and one can easily verify that this configuration is the one with the minimal cost over all 27 possible configurations of the system. In another situation where the demand is very high and no configuration can fulfill it (here, $Demand > 105$), then inevitably any configuration of the system will induce a high cost proportional to the demand.

Due to the dynamic nature of our environment, our systems are subject to change with respect to time. Depending on the situation, the cost associated with the same configuration may be different between the specifications of two different systems . and some components (i.e., some variables) may also be added or removed from a time point to another. In this framework, we consider a uniform, discrete representation of time. That is, a "snapshot" of the system is taken at a sequence of times. These "snapshots" could occur once a day, a month or any other frame of time depending on the application. However, we assume that the discretiza-

tion of time is *regular*: the period of time between any pair of consecutive snapshots remains the same. Modifications within a system are then assessed at each time step within a specific scenario. From a time step to the next one, the nature of these modifications is twofold. On the one hand, the system's controller takes a decision (i.e., he performs an *action*) as an upstream step, in an attempt to shape the system so as to target desirable configurations. This control operation is made by the system's controller and is called here the *design of the system's specifications*. This notion of control is different from the configuration of the system which we described earlier. In parallel with the action performed by the system's controller, external disturbances such as natural disasters or some smaller perturbations, may also alter the system's structure.

Our framework could be adapted to continuous time, but we adopt here a discretized point of view of time for simplicity reasons. Doing so, the system's modifications from a time step to the next one reflects as a whole the consequences of the actions performed by the system's controller at the system's design level (i), and the consequences of some exogenous event (ii). We are ready to formalize the notion of *system trajectory*, which describes the evolution of a system within some specific scenario.

Definition 2 (*System trajectory*) A *system trajectory* ST is a (possibly infinite) sequence of systems $(\mathcal{S}_0, \dots)$.

Accordingly, a system trajectory $ST = (\mathcal{S}_0, \dots)$ represents a possible evolution of a system. Each $\mathcal{S}_i \in ST$ represents the system (according to Definition 1) at time step i, and $\mathcal{S}_0$ represents the initial system. Different systems $\mathcal{S}_i, \mathcal{S}_j$ in a system trajectory are not required to satisfy any relationship in general. Let us consider again our running example:

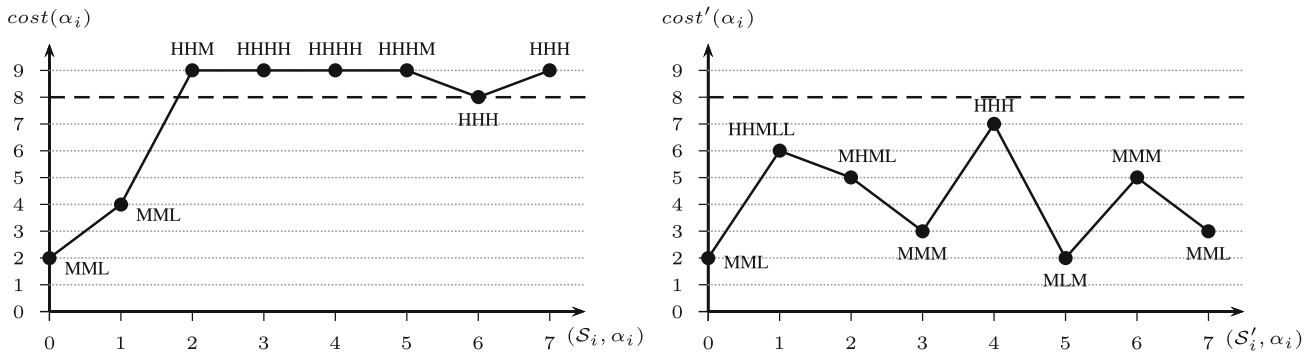
Example 3 (*Continued*) Let us assume that the time period (i.e., the time between any pair of consecutive systems within a system trajectory) is fixed to 1 h. The outcomes of the constraints $\mathcal{C}$ may differ from a time step to the next one. For instance, the price for the maintenance of each server may increase, i.e., the value $Price(\alpha(x_i))$ for each variable x_i. Moreover, the domain of the variables could be refined i.e., $D_i = \{None, Low, Medium, High, Very\ High\}$, where the value "None" would represent the fact that the server is maintained but not allowed any memory size. Further, the company could invest into more servers, i.e., the set of variables would increase.

Let us now introduce the definition of a *state trajectory*.

Definition 3 (*State trajectory*) A *state trajectory* SST is a (possibly infinite) sequence of system states $((\mathcal{S}_0, \alpha_0), \dots)$, i.e., for every $i \in \{0, \dots\}$, α_i is a configuration of $\mathcal{S}_i$. A *subtrajectory* SST' of SST is a subsequence of consecutive system states from SST. We note $SST' \sqsubseteq SST$ whenever SST' is a subtrajectory of SST.

A state trajectory associates with each system from a system trajectory a configuration which is specified by the system's controller.

Example 4 (*Continued*) We consider that the domains D_i remain unchanged within a system trajectory, i.e., $D_i = \{Low, Medium, High\}$ for each variable x_i. The sequence $SST^{(1)} = ((\mathcal{S}_0, MML), (\mathcal{S}_1, MML), (\mathcal{S}_2, HHM), \dots, (\mathcal{S}_7, HHH))$ which is depicted in Fig. 2a is an example of a state trajectory on a 7-h frame (from time 0 to time 7). This state trajectory is represented in the figure as a cost curve, i.e., each system state $(\mathcal{S}_i, \alpha_i)$ is associated with the cost of the configuration α_i in $\mathcal{S}_i$. Note that within $SST^{(1)}$, the configuration remains sometimes unchanged from a system to the next one (e.g., from $\mathcal{S}_0$ to $\mathcal{S}_1$); however, the cost associated with it may change [we have $cost(MML) = 2$ in $\mathcal{S}_0$ and $cost(MML) = 4$ in $\mathcal{S}_1$]. Moreover, the number of variables here varies within the trajectory: the systems $\mathcal{S}_3$, $\mathcal{S}_4$ and $\mathcal{S}_5$ contain four variables, whereas the other systems contain three variables.



(a) $SST^{(1)}$ is *not* 8-resistant **(b)** $SST^{(2)}$ is 8-resistant

Fig. 2 Two state trajectories $SST^{(1)}$ and $SST^{(2)}$

At this point, we are ready to introduce the resilience-related properties for state trajectories. These properties will be later extended to dynamic systems in Sect. 5.

4 Measuring the resilience of state trajectories

In this section we introduce several measures, first introduced in [38], which are crucial to assess the resilience of a state trajectory. To this purpose, we take our main inspiration from Bruneau's work [8] who reconciled a number of concepts underlying resilience within two main characteristics: the *absorption* of shocks when they occur, and the *recovery* after a shock, i.e., the capability to establish back a normal performance.

4.1 Resistance

Resistance deals with a system's inherent response to some perturbation, i.e., its ability to absorb (or to resist to) external fluctuations. This corresponds in Grimm and Calabrese's persistence [18] as the ability for a system to stay essentially unchanged despite the presence of disturbances. Bruneau [8] suggested that this is one of the two main characteristics of a resilient system, which shows "reduced consequences from failures, in terms of lives lost, damage, and negative economic and social consequences." He also referred to the term "robustness" as the capability to keep a given level of demand without directly suffering degradation. This concept is adequate to the initial interpretation of resilience from Holling [23], as a system's inherent ability to absorb the external shocks. We represent this general intuition for state trajectories as follows:

Definition 4 (*Resistance*) Given a state trajectory $SST = ((\mathcal{S}_0, \alpha_0), \dots)$ and a non-negative number l, SST is said to be *l-resistant* if for each $i \in \{0, 1, \dots\}$, $cost(\alpha_i) \leq l$.

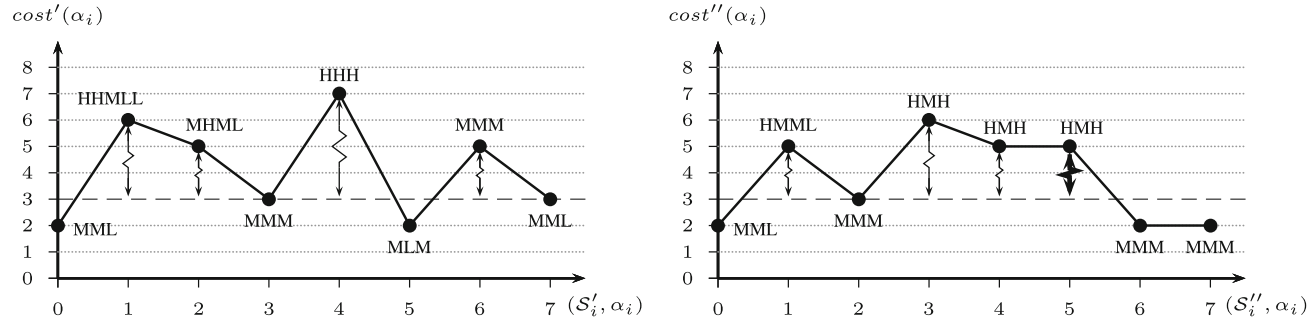
A state trajectory is l-resistant if the cost of each configuration of its system states is kept under the threshold l. Accordingly, it corresponds to the system's inherent capability to absorb the perturbations (at a certain degree). In some applications where the distinction between the set of "irrevocable" states and the set of "safe" states can be identified by a threshold l, checking the l-resistance of a state trajectory is useful to guarantee the safety of the system.

Example 5 (*Continued*) Assume that we are able to identify irrevocable system states from others, i.e., the situation becomes unacceptable when the available budget does not allow the company to afford new servers, or when the resource shortage is excessively high. Let us suppose that this characteristic resistance threshold here is $l = 8$. Figure 2a, b depicts two state trajectories $SST^{(1)} = ((\mathcal{S}_0, MML), (\mathcal{S}_1, MML), (\mathcal{S}_2, HHM), \dots)$ and $SST^{(2)} = ((\mathcal{S}_0, MML), (\mathcal{S}'_1, HHMLL), (\mathcal{S}'_2, MHML), \dots)$ which represent two different scenarios. For both of these trajectories, the initial system state is set to $(\mathcal{S}_0, MML)$ with a cost equal to $cost(MML) = 2$. In both scenarios a drastic increase of demand occurs between time 0 and time 1. On the one hand, Fig. 2a depicts the state trajectory $SST^{(1)}$, in which the two control operations performed iteratively by the systems' controller every hour, i.e., the configuration of the system on the one hand, the design of the system's specifications on the other hand, lead after 2 h to the system state $(\mathcal{S}_2, HHM)$, with $cost(HHM) = 9$ in $\mathcal{S}_2$. Therefore, $SST^{(1)}$ is not 8-resistant. It this scenario, no decision is taken early enough (for instance, the acquisition of additional servers) to prevent the system from an unacceptable situation that cannot be handled (for instance, before time 1). On the other hand, Fig. 2b depicts another state trajectory $SST^{(2)}$ where a different decision was taken between time 0 and time 1, in an attempt to absorb the impacts of this drastic workload change. At time 1, it results in the system state $(\mathcal{S}'_1, HHMLL)$. In $SST^{(2)}$, at each time i the system $\mathcal{S}'_i$ can be associated with a configuration α_i and with a cost not higher than 7. Therefore, $SST^{(2)}$ is 8-resistant.

4.2 Recoverability

From the New International Websters Comprehensive Dictionary (2014), resilience is elasticity: "the power of springing back to a former position or shape." A resilient system is allowed to be "deformed," i.e., to express temporarily the consequences of some external disturbances; but then it should be able to return to a satisfactory state. To picture the idea, consider a squeeze ball which absorbs energy when it is deformed elastically and then returns to its initial shape by unloading this energy: such an object is resilient w.r.t. this interpretation. Similar definitions involving elasticity have been considered, e.g., the capacity to cope with unanticipated dangers after they have become manifest, to bounce back [49, p. 77]. This notion was already considered by Holling [23] through the term *stability*: the ability of a system to return to an equilibrium state after some temporary disturbance: "The more rapidly it returns, with the least fluctuation, the more stable it is" [23, p. 17]. In this definition, an equilibrium state is a state providing an acceptable level of performance, and a "stable" system shows a reduced time to recovery, i.e., to restore to such an equilibrium state.

Based on this idea, Bruneau [8] proposed a quantitative formulation of the notions of elasticity/fluctuation. It is based on a measure $Q(t)$ which depends on time and evaluates the quality of the system from 0 to 100 %. For instance, when this measure assesses the degree of service of the system, 0 % means no degradation of service and 100 % means no service is available. Then, if the system is subject to some exogenous



(a) $SST^{(2)}$ is $\langle 3, 5\rangle$-recoverable **(b)** $SST^{(3)}$ is *not* $\langle 3, 5\rangle$-recoverable

Fig. 3 Two state trajectories $SST^{(2)}$ and $SST^{(3)}$

disturbance at time t_0 and is effectively affected, one can measure the size of the degradation in quality over time until the system is completely repaired (time t_1). Formally, in such a situation the system must be able to minimize the "triangular area" defined by the integral $\int_{t_0}^{t_1} Q(t)\mathrm{d}t$ (see [8] for more details). This notion can be conveniently adapted to state trajectories in our framework. As to the case of resistance (cf. Definition 4), the property we are about to introduce, called *recoverability*, exploits the cost associated with each configuration from a given state trajectory. This property is based on two parameters. Let us first introduce the notion of unstable subtrajectory:

Definition 5 (*Unstable subtrajectory*) Given a state trajectory $SST = ((\mathcal{S}_0, \alpha_0), \dots)$ and a non-negative number p, a subtrajectory $((\mathcal{S}_a, \alpha_a), \dots, (\mathcal{S}_b, \alpha_b))$ of SST is said to be *p-unstable* if for every $i \in \{a, \dots, b\}$, we have $cost(\alpha_i) > p$.

Definition 6 (*Recoverability*) Given a state trajectory $SST = ((\mathcal{S}_0, \alpha_0), \dots)$ and two non-negative numbers p and q, SST is said to be *$\langle p, q\rangle$-recoverable* if for any p-unstable subtrajectory $((\mathcal{S}_a, \alpha_a), \dots, (\mathcal{S}_b, \alpha_b))$ of SST, the following conditions are satisfied:

$$\begin{cases} \text{(i) } \sum_{i=a}^{b} (cost(\alpha_i) - p) \leq q, \\ \text{(ii) } \exists (\mathcal{S}_t, \alpha_t) \in SST \text{ s.t. } t > b \text{ and } cost(\alpha_t) \leq p. \end{cases}$$

On the one hand, q represents an upper bound for the total amount of extra cost (i.e., costs above p) that is necessary for a $\langle p, q\rangle$-recoverable state trajectory to get back to a "satisfactory" state [condition (ii)], i.e., a configuration associated with a cost below or equal to p. This cumulative extra cost corresponds to the "triangular area" of the degradation of the quality of the system over time in Bruneau's definition [8]. Since we consider a discretization of time, instead of computing an integral, extra costs are simply summed up. On the other hand, p is used here as a threshold, which is different from the parameter l in the definition of resistance (cf. Definition 4). Indeed, the parameter l in the property of resistance can be used to distinguish "acceptable" states from "unacceptable" ones. However, the parameter p represents the cost level under which configurations are "fully acceptable" (those involving no degradation of service, i.e., similar to 100 % in Bruneau's definition). The configurations associated with a cost higher than p are still considered as being acceptable, but "unsafe" or "unstable:" then $\langle p, q\rangle$-recoverability requires that one must come back to a configuration with a cost no higher than p [condition (ii)], and that should be done within a time period and a fluctuation degree specified by q [condition (i)]. Let us illustrate the property using our running example:

Example 6 (*Continued*) Assume that $p = 3$ represents the cost threshold above which the aggregation of the invested budget and resource shortage is too high to be considered as fully acceptable. On the one hand, Fig. 3a depicts the same state trajectory $SST^{(2)}$ as in Fig. 2b. This scenario involves some 3-unstable subtrajectories (i.e., whose configurations are associated with a cost higher than 3). However, it can be checked that the accumulative costs for these 3-unstable subtrajectories are never higher than 5 [condition (i) of Definition 6] and that each one of these subtrajectories are followed by a system state whose configuration has a cost not exceeding 3 [condition (ii) of Definition 6]. Therefore, $SST^{(2)}$ is $\langle 3, 5\rangle$-recoverable. On the other hand, Fig. 3a depicts another state trajectory $SST^{(3)}$ which contains a 3-unstable subtrajectory starting from step 3. At time 5, the cumulative extra cost exceeds 5, so that $SST^{(3)}$ is not $\langle 3, 5\rangle$-recoverable. Note that as a counterpart, $SST^{(3)}$ is 6-resistant whereas $SST^{(2)}$ is not.

This property allows us to identify whether a trajectory can bounce back to a satisfactory state after a shock, within a certain allowance of accumulative loss. In other words, when both parameters p and q are known, this can be easily decided whether a trajectory is $\langle p, q\rangle$-recoverable. Our definition is also consistent with the way *robustness* has been stated in the literature (e.g., [25, chapter 6]), where robustness qualifies an ability to recover from the effects of a fault as quickly as possible to minimize the impact of the fault.

4.3 Functionality

A state trajectory may be both resistant and recoverable at a certain extent (i.e., relatively to the parameters l, p, q), but may still provide an unsatisfactory level of service over an extended period of time. For instance, a state trajectory may be recoverable (for some given parameters p, q) but consist of system states associated with relatively high costs *in average*. This would be the case for a state trajectory which always recovers to a satisfactory state within the accumulative cost threshold q, but which always goes again to a configuration with a cost higher than p soon after recovery. To cope with this additional consideration, we introduce here another property, named *functionality*. Similarly to the case of resistance, functionality relates to the system's inherent capability of "absorbing" shocks. However, while resistance demands that the system maintains a certain quality level (for ecological systems) or provides a certain quality of service (for engineering systems) *at every time step*, functionality requires the system to maintain/provide this level in average. Thus, functionality analyses a state trajectory "globally" whereas resistance analyses a state trajectory at a each time step. Let us denote $|SST|$ the cardinality of a state trajectory SST (if SST is infinite, then $|SST| = +\infty$). We also denote $avg(SST)$ the average of costs associated with all configurations from SST, that is,

$$avg(SST) = \begin{cases} \sum_{i=0}^{|SST|-1} \frac{cost(\alpha_i)}{|SST|} & \text{if } SST \text{ is finite,} \\ \lim_{k\to+\infty} \sum_{i=0}^{k} \frac{cost(\alpha_i)}{k} & \text{otherwise.} \end{cases}$$

Definition 7 (*Functionality*) Given a state trajectory $SST = ((\mathcal{S}_0, \alpha_0), \dots)$ and a non-negative number f, SST is said to be f*-functional* if $avg(SST) \leq f$.

In other words, a state trajectory is f-functional if the (possibly infinite) mean of costs involved in it is kept under some threshold f. Through this property, the quality of the system is measured *in the long run* while preventing disturbances that may destabilize it. The following example shows the distinction between functionality and recoverability:

Example 7 (*Continued*) Let us consider again Fig. 3a, b that depicts the state trajectories $SST^{(2)}$ and $SST^{(3)}$. Let us recall that $SST^{(2)}$ is $\langle 3, 5\rangle$-recoverable while $SST^{(3)}$ is not. However, we have $avg(SST^{(2)}) = (2+6+5+3+7+2+5+3)/8 = 33/8 = 4.125$, and $avg(SST^{(3)}) = (2+5+3+6+5+5+2+2)/8 = 30/8 = 3.75$. Therefore, $SST^{(3)}$ is 4-functional, whereas $SST^{(2)}$ is not.

4.4 Resilience: a unifying property

We introduced three parameterized properties, namely resistance, recoverability and functionality, that quantify different aspects of the resilience of a state trajectory. These properties are all based on the evaluation of the costs involved in a state trajectory. One may now ask oneself whether these properties share a common point that could be generalized within a single property. We give a positive answer to this question by introducing the property of *resilience*. This property is parameterized by two numbers. By adjusting the two parameters, one can express properties which are "between" resistance and functionality; for some specific parameters, one can also equivalently rephrase the properties of resistance for all state trajectories, and functionality for finite state trajectories. Additionally, by combining some parameters, the property of resilience can also be express as a strengthening of the property of recoverability when considering finite state trajectories. It allows us to capture in a simple way both notions of absorption of disturbances (captured by resistance at each time step, and by functionality in a "global" way), and fluctuation degree (captured by recoverability).

Definition 8 (*Resilience*) Given a state trajectory $SST = ((\mathcal{S}_0, \alpha_0), \dots)$, a positive integer k and a non-negative number l, SST is said to be $\langle k, l\rangle$*-resilient* if for all its subtrajectories SST' of size k, we have $avg(SST') \leq l$.

The property of $\langle k, l\rangle$-resilience requires that the average cost is kept under the threshold l when looking at *every* time interval of size k within a state trajectory. Small k reduces the time allowance for a state trajectory to recover to some satisfactory state in case of fluctuation within the cost curve. Larger k allows more time to obtain a balance of acceptable cost level. Let us illustrate the property within an example:

Example 8 (*Continued*) Let us consider the state trajectory $SST^{(2)}$ (cf. Fig. 3a). We are asked whether $SST^{(2)}$ is $\langle 4, 5\rangle$-resilient, i.e., whether each subtrajectory of four consecutive system states from $SST^{(2)}$ has a cost average below 5. This is not the case here: consider the subtrajectory $SST^{(2)}_{1\to4} = ((\mathcal{S}'_1{}^J, HHMLL), (\mathcal{S}'_2, MHML), (\mathcal{S}'_3, MMM), (\mathcal{S}'_4, HHH))$ from time 1 to time 4. We have $avg(SST^{(2)}_{1\to4}) = (6+5+3+7)/4 = 5.25$. Therefore, $SST^{(2)}$ is not $\langle 4, 5\rangle$-resilient. On the other hand, by similar computations one can check if the state trajectory $SST^{(3)}$ (cf. Fig. 3b) is $\langle 4, 5\rangle$-resilient. In simple terms, the state trajectory $SST^{(3)}$ exhibits a more "resilient" behaviour than the state trajectory $SST^{(2)}$ when considering a reference time frame of 3 h (i.e., 4 time steps).

We now point out that the properties of resistance and functionality are specific cases of resilience:

Observation 1 *For any non-negative number l, a state trajectory SST is l-resistant if and only if it is $\langle 1, l\rangle$-resilient.*

Indeed, resistance requires that no "stretching" of the cost curve is possible, i.e., no configuration within the trajectory should exceed the threshold l: external shocks should

be inherently absorbed by the system. On the other hand, when the interval size k corresponds to the size of the state trajectory, resilience amounts to functionality:

Observation 2 *For any non-negative number f, a finite state trajectory SST is f-resistant if and only if it is $\langle |SST|, f\rangle$-resilient.*

Besides capturing the concepts of resistance and functionality, the property of resilience can also be viewed as a strengthening of recoverability. Indeed, for any parameters p, q, the property of $\langle p, q\rangle$-recoverability can be characterized in a stronger way by a conjunction of $\langle k, l\rangle$-resilient conditions by varying the interval size k:

Proposition 1 *For any pair of non-negative numbers p, q, a finite state trajectory SST is $\langle p, q\rangle$-recoverable if for every $k \in \{1, \ldots, |SST|\}$, it is $\langle k, (p + q/k)\rangle$-resilient.*

Proof Let SST be a finite state trajectory and $p, q \in \mathbb{R}^+$. Assume that for every $k \in \{1, \ldots, |SST|\}$, it is $\langle k, (p + q/k)\rangle$-resilient. Toward a contradiction, assume that SST is not $\langle p, q\rangle$-recoverable. This means that there exists a p-unstable subtrajectory $SST' = ((S_a, \alpha_a), \ldots, (S_b, \alpha_b))$ of SST such that one of the two conditions is satisfied: (i) $\sum_{i=a}^{b} (c_i(\alpha_i) - p) > q$, or (ii) there is no system state (S_t, α_t) of SST, $t > b$, such that $c_t(\alpha_t) \leq p$. Assume first that condition (i) is satisfied. Then we have $\sum_{i=a}^{b} (c_i(\alpha_i) - p) > q$, or equivalently, $\sum_{i=a}^{b} (c_i(\alpha_i)) > p.(b - a + 1) + q$. By dividing both sides of this equation by $(b-a+1) = |SST'|$, we get that $avg(SST') > p+q/|SST'|$. By Definition 8, this means that SST is not $\langle k, (p + q/k)\rangle$-resilient, leading to a contradiction. In the case where condition (i) is not satisfied, assume that condition (ii) is satisfied. Then this means that (S_b, α_b) is the last system state of SST. Thus there cannot exist a system state (S_t, α_t) of SST with $t > b$. This leads to a contradiction and concludes the proof. ■

Example 9 (*Continued*) Let $p = 4$ and $q = 3$. One can easily verify that $SST^{(2)}$ (cf. Fig. 3a) is $\langle k, (p+q/k)\rangle$-resilient for every $k \in \{1, \ldots, 8\}$. Indeed, it is $\langle 1, 7\rangle$-resilient (i.e., 7-resistant), $\langle 2, 5.5\rangle$-resilient, $\langle 3, 5\rangle$-resilient, $\langle 4, 4.75\rangle$-resilient, $\langle 5, 4.6\rangle$-resilient, $\langle 6, 4.5\rangle$-resilient, $\langle 7, (31/7)\rangle$-resilient and $\langle 8, 4.375\rangle$-resilient (i.e., 4.375-functional). Therefore, $SST^{(2)}$ is $\langle p, q\rangle$-recoverable, that is, $\langle 3, 4\rangle$-recoverable.

Observations 1 and 2 show that the properties of l-resistance and f-functionality can be equivalently rephrased into the property of $\langle k, l\rangle$-resilience. Proposition 1 shows that the property of $\langle p, q\rangle$-recoverability is a weakening of a conjunction of some $\langle k, l\rangle$-resilience properties by varying the parameters k and l. These observations tell us that our property of $\langle k, l\rangle$-resilience is a good candidate as a very general concept capturing both notions of shock absorption and elasticity (for different parameters k, l) previously described. However, please note that the property of recoverability is not equivalently captured by a combination of resilience properties, since it is only a weakening of such combination: in other terms, a $\langle p, q\rangle$-recoverable state trajectory may not be $\langle k, l\rangle$-resilient for some parameters p, q, k, l which are related as given in Proposition 1. Then the property of $\langle p, q\rangle$-recoverability for state trajectories is still of interest on its own, especially when the threshold dinstinguishing "safe" states from "unsafe" ones is known.

4.5 From $\langle k, l\rangle$-resilience to k-resiliency

We now would like to stress an important point with respect to the parameterized property of $\langle k, l\rangle$-resilience. By fixing both parameters k, l, we are able to check whether a state trajectory is $\langle k, l\rangle$-resilient. However, one may not know in general some required threshold l given a specific interval size k. In this case one wants to know given some specific state trajectory and by just fixing the parameter k, *how resilient* it is without providing some specific threshold l. It is clear here that given k, if a state trajectory is $\langle k, l\rangle$-resilient, then it is also $\langle k, l'\rangle$-resilient for every threshold $l' \geq l$. Evaluating how resilient a state trajectory consists in an *optimization* problem, and this comes down to consider the smallest threshold l for which the state trajectory is $\langle k, l\rangle$-resilient. We call such minimal value the *k-resiliency* of a state trajectory with respect to some time interval k:

Definition 9 (*k-Resiliency*) Given a state trajectory $SST = ((S_0, \alpha_0), \ldots)$ and a positive integer k, the *k-resiliency* of SST is the smallest number l for which SST is $\langle k, l\rangle$-resilient, or more formally, the number $\inf\{l \in \mathbb{R}^+ | SST$ is $\langle k, l\rangle$-resilient$\}$.

One can observe from Definitions 8 and 9 that the k-resiliency of a state trajectory is simply the maximal (more precisely, the supremum) cost average found among all of its subtrajectories of size k:

Observation 3 *The k-resiliency of SST corresponds to the number* $\sup\{avg(SST') | SST' \sqsubseteq SST, |SST'| = k\}$.

Example 10 (*Continued*) Consider again the state trajectory $SST^{(2)}$ (cf. Fig. 3a), and let us compute its 4-resiliency. For each $i \in \{0, 4\}$, let us denote $SST_i^{(2)}$ the subtrajectory of $SST^{(2)}$ of size 4 starting at time i. Then we have $avg(SST_0^{(2)}) = 4$, $avg(SST_1^{(2)}) = 5.25$, $avg(SST_2^{(2)}) = 4.25$, $avg(SST_3^{(2)}) = 4.25$ and $avg(SST_4^{(2)}) = 4.25$; therefore, the 4-resiliency of $SST^{(2)}$ is equal to 5.25.

By now, one can associate with every state trajectory SST and every positive integer k the smallest number l corresponding to the k-resiliency of SST. This induces a function characterizing the k-resiliency of a given state trajectory, for all positive integers k. We call this function the *resiliency function* of SST:

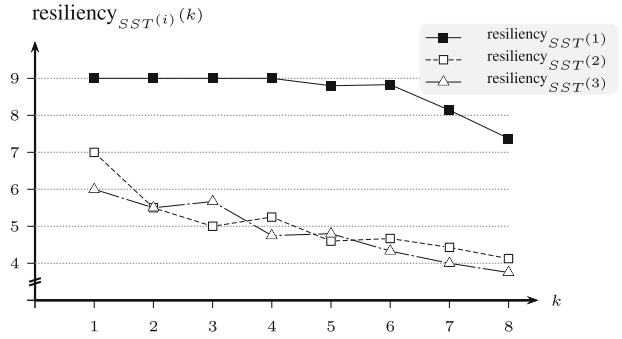


Fig. 4 The resiliency functions associated with $SST^{(1)}$, $SST^{(2)}$ and $SST^{(3)}$

Definition 10 (*Resiliency function*) Given a state trajectory $SST = ((\mathcal{S}_0, \alpha_0), \dots)$, the *resiliency function* of SST, denoted res_{SST}, is the mapping from $\mathbb{N}^* = \{1, 2, \dots\}$ to $\mathbb{R}^+$ such that for every $k \in \mathbb{N}^*$, $res_{SST}(k)$ is defined as the k-resiliency of SST.

Please note that since the property of $\langle k, l \rangle$-resilience captures both notions of l-resistance and f-functionality, one can derive the notion of "resistance degree" (respectively, "functionality degree") of a state trajectory SST by omitting the parameter l (respectively, the parameter f). On the one hand, the resistance degree of SST corresponds to its 1-resiliency, i.e., it is equal to $res_{SST}(1)$. On the other hand, the functionality degree of SST is equal to $res_{SST}(|SST|)$ in the case where SST is a finite trajectory; otherwise, it is equal to $\lim_{k \to +\infty} res_{SST}(k)$.

We dispose now of a function which is a convenient tool to analyse, visualise and compare the behaviour of different state trajectories from a resiliency point of view. This is illustrated in our running example:

Example 11 (*Continued*) We compute the resiliency function associated with each one of the state trajectories $SST^{(1)}$, $SST^{(2)}$ and $SST^{(3)}$ depicted, respectively, in Figs. 2a and 3a, b. For instance, we already computed in the previous example the 4-resiliency of $SST^{(2)}$ which is equal to 5.25. Similarly, one can compute the k-resiliency of each trajectory, for $k \in \{1, \dots, 8\}$. The resiliency functions associated with $SST^{(1)}$, $SST^{(2)}$ and $SST^{(3)}$ are depicted in Fig. 4.

The k-resiliency degree of $SST^{(1)}$ is higher than those of $SST^{(2)}$ and $SST^{(3)}$, for every $k \in \{1, \dots, 8\}$. This shows that $SST^{(2)}$ and $SST^{(3)}$ are globally more "resilient" than $SST^{(1)}$. The trajectory $SST^{(3)}$ is more resistant than $SST^{(2)}$, since we have $res_{SST^{(3)}}(1) < res_{SST^{(2)}}(1)$. The trajectory $SST^{(3)}$ is also more functional than $SST^{(2)}$, since $res_{SST^{(3)}}(8) < res_{SST^{(2)}}(8)$. For relatively high k, i.e., $k = 6$ and $k = 7$, the k-resiliency of $SST^{(3)}$ is also better than the k-resiliency of $SST^{(2)}$. This reflects the fact that when we allow some relatively high amount of time for a state trajectory to "recover" from damages, then $SST^{(3)}$ behaves better than $SST^{(2)}$. The fact that $SST^{(3)}$ has a good 4-resiliency can be intuitively seen in Fig. 3b, when looking at the curve related to $SST^{(3)}$ from time 3 to time 6: indeed, within the state trajectory $SST^{(3)}$, the system state at time 3 is associated with a relatively high cost, but it recovers adequately to a satisfactory system state at time 6, i.e., a system state associated with a cost of 2. However, for some intermediate interval size k, $SST^{(2)}$ exhibits a better behaviour than $SST^{(3)}$. For instance, for $k = 3$, we have $res_{SST^{(2)}}(3) < res_{SST^{(3)}}(3)$, and indeed, $SST^{(2)}$ gets back to a more satisfactory state than $SST^{(3)}$ in less than 3 time steps. This can be intuitively seen by comparing the two state trajectories in Fig. 3a, b, focusing on their respective subtrajectories between time 3 and time 5.

The resiliency function of a state trajectory could be further analysed in a number of ways. For instance, in our example (cf. Fig. 4) one can see that there is a gap between the 1-resiliency and the 2-resiliency for $SST^{(2)}$ (indeed, $res_{SST^{(2)}}(1) = 7$ and $res_{SST^{(2)}}(2) = 5.5$). This reflects the fact that $SST^{(2)}$ may fall in a state with a relatively high cost, but in such case it gets back to a satisfactory state directly on the next time step. This is clear when looking at Fig. 3a, where the cost of the configuration at time 4 is equal to 7 and falls down to 2 at the next hour.

4.6 Stabilizability

We introduce here an additional property, namely *stabilizability*, which finds its relevance in situations where certain (drastic) transitions between a configuration to another one cannot be performed. In this property, the cost of these configurations does not play a role; on the contrary, the configurations themselves are the key elements to assess the stabilizability of a state trajectory. Such notion of transitions between system states can be adequately modelled in our framework. Indeed, we can assume that the set of all possible system states is together with a *distance*, denoted d. For instance, the well-known Hamming distance [21] can be used. The Hamming distance between two system states $(\mathcal{S}, \alpha)$, $(\mathcal{S}', \alpha')$, denoted $d_H((\mathcal{S}, \alpha), (\mathcal{S}', \alpha'))$, corresponds to the number of "differences" between them. Formally, if $\mathcal{S} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ and $\mathcal{S}' = \langle \mathcal{X}', \mathcal{D}', \mathcal{C}' \rangle$, we have $d_H((\mathcal{S}, \alpha), (\mathcal{S}', \alpha')) = |\{\alpha(x_i) | x_i \in \mathcal{X} \cap \mathcal{X}', \alpha(x_i) \neq \alpha'(x_i)\}| + |\mathcal{X} \setminus \mathcal{X}'| + |\mathcal{X}' \setminus \mathcal{X}|$.[1] Other distances can be considered depending on the application as in the following example:

Example 12 (*Continued*) We consider a specific distance d between system states. d is defined for all system states

[1] Typically, Hamming distance defines a distance between two assignments based on the same set of variables. This definition is a straighforward extension of it for system states where different sets of variables can be considered.

$(\mathcal{S}, \alpha), (\mathcal{S}', \alpha')$ as $d((\mathcal{S}, \alpha), (\mathcal{S}', \alpha')) = \sum_{i=1}^{3} l_d(\alpha(x_i), \alpha'(x_i))$, where for every $x_i \in \mathcal{X}$, $l_d(\alpha(x_i), \alpha'(x_i)) = 0$ if $\alpha(x_i) = \alpha'(x_i)$, $l_d(\alpha(x_i), \alpha'(x_i)) = 3$ if $(\alpha(x_i), \alpha'(x_i)) \in \{(L, H), (H, L)\}$, and $l_d(\alpha(x_i), \alpha'(x_i)) = 1$ in the remaining cases. This distance represents here the fact that modifying the size of each server from "Low" to "High" reflects a higher "change" than a modification from "Low" to "Medium." For instance, for all systems $\mathcal{S}$, $\mathcal{S}'$, we get that $d((\mathcal{S}, HMH), (\mathcal{S}, HLL)) = 0 + 1 + 3 = 4$.

Definition 11 (*Stabilizability*) Given a state trajectory $SST = ((\mathcal{S}_0, \alpha_0), \dots)$ and a non-negative number s, SST is said to be *s-stabilizable* if for each $i \in \{1, \dots\}$, $d(\alpha_{i-1}, \alpha_i) \leq s$.

This additional property shows that a further analysis can be conducted on resilient trajectories. Stabilizability is distinguishable from the concept of resilience itself, as a function of the system's controller (or the social component) to manage the system. However, it can be used in combination with the resilience property, as to evaluate the ability for a state trajectory to avoid undergoing modifications while maintaining a resilient behaviour. Stabilizability is closely related to the notion of *adaptability* introduced in [16,46], as the capacity of actors in a system to maintain resilience.

5 Dynamic system

We are now ready to introduce the notion of a *dynamic system*. This structure allows one to represent a large set of possible evolutions for a system, i.e., of system trajectories: at every time step, a set of possible actions operating at the system's design level are available for the system's controller, and each one of these actions may have a non-deterministic effect which depends on the environment's perturbations. Our notion of dynamic system is similar to the one of discrete event dynamic system (DEDS for short) which are standard structures to represent a dynamic world supervised by an agent [5,12,31,34].

Definition 12 (*Dynamic system*) Let $\mathcal{S}_{all}$ be the set of all possible systems. A *dynamic system* DS is a tuple $\langle \mathcal{S}_0, \mathcal{A}, poss, \Phi_A \rangle$, where

- $\mathcal{S}_0 \in \mathcal{S}_{all}$ and is the "initial" system, which represents the specifications of the current system;
- $\mathcal{A}$ is a set of actions (or moves, or decisions); the set $\mathcal{A}$ contains a specific action denoted *nop* which represents the fact that nothing is effectively done by the system's controller;
- $poss$ is a mapping from $\mathcal{S}_{all}$ to $2^{\mathcal{A}}$ which gives a set of possible actions for each system, such that for every system $\mathcal{S} \in \mathcal{S}_{all}$, $nop \in poss(\mathcal{S})$;
- Φ_A is a mapping from $\mathcal{S}_{all} \times \mathcal{A}$ to $2^{\mathcal{S}_{all}}$ such that for every system $\mathcal{S} \in \mathcal{S}_{all}$, for every action $a \in poss(\mathcal{S})$, $\Phi_A(\mathcal{S}, a)$ relates to some non-empty subset of systems from $2^{\mathcal{S}_{all}}$. Φ_A specifies how a given system may be modified in response to some actions, depending on the occurrence of some exogenous events.

As far as the functions $poss$ and Φ_A are effectively computable (as it is typically assumed), we do not add any restriction as to the choice of the representation language in which they are specified, that is a concern beyond the scope of this paper. The tacit assumptions that an action is always available in every system (by default, the *nop* action) and that possible actions always lead to some successor systems [$\Phi_A(\mathcal{S}, a)$ is always a non-empty set] are standard. A dynamic system can be represented as a graph where each vertex represents a system and each edge represents the (potential) consequence of some action which would transform the current system into the next system (e.g., adding/removing some variables, modifying the constraints). Please note again that Φ_A is a non-deterministic function, in the sense that from a specific action performed by the system's controller, one may fall into several possible successor systems. This non-determinism is due to the presence of exogenous events which may alter the result of an action. More precisely, an exogenous event is implicitly modelled as an operation of choosing one system specification from an output of the function Φ_A. Doing so, the exogenous events are not explicitly represented, but their possible consequences lie in the non-determinism of Φ_A in which they take a fundamental part.

Please note that although a dynamic system here may look like a non-deterministic finite automaton (NFA), it is not: in our definition, there is no final or accepting states, and we may have an infinite number of reachable systems from the initial one S_0.

Example 13 (*Continued*) We denote DS as a dynamic system which represents all possible scenarios about the evolution of the servers maintained by our service provider. Assume that the system $\mathcal{S}$ described at first in this example represents the current system, i.e., $\mathcal{S}_0 = \mathcal{S}$. The function Φ_A is derived from a study of the potential changes of service demand at the next time step depending on the situation at the previous step. For instance, one can expect that the change of demand is "smooth enough" from a time step to the next one, i.e., the demand can be very high at the next time step if it is already high at the current time step, but not if it is currently low. Moreover, these actions (represented in $\mathcal{A}$) typically depend on the given states of the dynamic system (i.e., on each system); one can assume that one cannot add more than a certain fixed number of servers from a time

step to the next one, due to some limitations in the adaptation of the budget, or simply because of the market.

Figure 5 depicts the (partial) graph representing how the dynamic system DS could look like. We have $\mathcal{A} = \{nop, a_1, \ldots, a_5\}$, and for instance, $poss(\mathcal{S}_0) = \{nop, a_4\}$, that is, the actions nop and a_4 can be taken within the initial system S_0 and $\phi_A(\mathcal{S}_0, nop) = \{\mathcal{S}_0, A\}$, that is, by performing the action nop within the system $\mathcal{S}_0$, we may fall into the system $\mathcal{S}_0$ or A depending on the environment.

The following definition is a natural adaptation of the definition of a state trajectory (cf. Definition 3) which has a "realization" in a given dynamic system:

Definition 13 (*Realizable state trajectory*) Given a dynamic system $DS = \langle \mathcal{S}_0, \mathcal{A}, poss, \Phi_A \rangle$, a state trajectory $ST = ((\mathcal{S}_0, \alpha_0), \ldots)$ is said to be *realizable in* DS if for every $\mathcal{S}_i \in ST$, $i > 0$, there is a move $a \in poss(\mathcal{S}_{i-1})$ such that $\mathcal{S}_i \in \Phi_A(\mathcal{S}_{i-1}, a)$.

Example 14 (*Continued*) Consider again our dynamic system DS represented in Fig. 5, and consider some system states $(\mathcal{S}_i, \alpha_i)$, $i \in \{1, \ldots, 6\}$ where $\mathcal{S}_1 = \mathcal{S}_6 = A$, $\mathcal{S}_2 = \mathcal{S}_5 = B$, $\mathcal{S}_3 = C$ and $\mathcal{S}_4 = D$. Then the state trajectory $((\mathcal{S}_0, \alpha_0), \ldots, (\mathcal{S}_6, \alpha_6))$ is realizable in DS.

In [22], the authors provide a general discussion about the criteria to take into account for building embedded systems: critical systems engineering considers the worst case analysis while best-effort systems engineering is based on an average-case. The design issues for building resilient systems as defined in this paper are quite similar to the challenges of *ensuring* a given level of QoS. In embedded systems, QoS corresponds to the guarantee that some combination of parameter values stay above a given of minimum thresholds (this can be a function of these parameters). Then naturally comes the question of controlling the system [34] such that its behaviour meets the expected specifications. To address this issue, we now introduce the notion of a *strategy* which plays a fundamental role in evaluating the resilience-related properties satisfied by a dynamic system. Indeed, a strategy marks out the level of control from the system's controller, which lies in two aspects of a dynamic system: the configuration which can be specified within a specific system at some point of time in some scenario (called the *configuration of the system*), and the decision which can be taken within a specific system to restrain the successor possible systems (called the *design of the system's specifications*). Accordingly, what is not under the control of the system's controller lies in the non-determinism of such actions, possibly leading to a multiple set of "next" possible systems triggered by an action. We define the notion of strategy for dynamic systems in a generic way, i.e., with respect to *any possible* underlying property $\mathcal{P}$ on state trajectories:

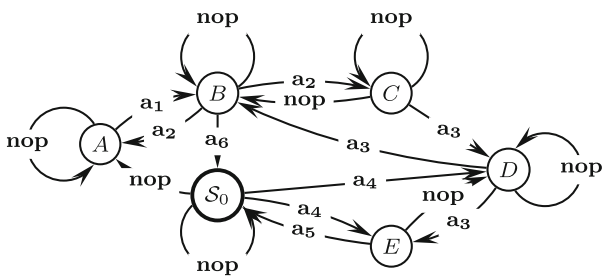


Fig. 5 The graphical representation of DS

Definition 14 (*Strategy*) Let $DS = \langle \mathcal{S}_0, \mathcal{A}, poss, \Phi_A \rangle$ be a dynamic system, $((\mathcal{S}_0, \alpha_0), \ldots, (\mathcal{S}_{b-1}, \alpha_{b-1}))$ be a state trajectory realizable in DS, $\mathcal{S}_b$ be a system state from $\mathcal{S}_{all}$ such that $\mathcal{S}_b \in \Phi_A(\mathcal{S}_{b-1})$, and $\mathcal{P}$ be a property on state trajectories. A 0-*horizon strategy* for DS w.r.t the property $\mathcal{P}$, preceded by $((\mathcal{S}_0, \alpha_0), \ldots, (\mathcal{S}_{b-1}, \alpha_{b-1}))$ and anchored in $\mathcal{S}_b$ is a configuration α_b of $\mathcal{S}_b$ such that $((\mathcal{S}_0, \alpha_0), \ldots, (\mathcal{S}_b, \alpha_b))$ satisfies $\mathcal{P}$.

Now, let t be a non-negative integer. A t-*horizon strategy* for DS w.r.t the property $\mathcal{P}$, preceded by $((\mathcal{S}_0, \alpha_0), \ldots, (\mathcal{S}_{b-1}, \alpha_{b-1}))$ and anchored in $\mathcal{S}_b$ is a pair $(\alpha_b, \mathcal{S}_{b+1})$, where α_b is a configuration of $\mathcal{S}_b$, $\mathcal{S}_{b+1} \in \Phi_A(\mathcal{S}_b)$, and such that there exists a $(t\text{-}1)$-horizon strategy for DS w.r.t the property $\mathcal{P}$, preceded by $((\mathcal{S}_0, \alpha_0), \ldots, (\mathcal{S}_b, \alpha_b))$ and anchored in $\mathcal{S}_{b+1}$.

An (*infinite horizon*) *strategy* for DS w.r.t the property $\mathcal{P}$, preceded by $((\mathcal{S}_0, \alpha_0), \ldots, (\mathcal{S}_{b-1}, \alpha_{b-1}))$ and anchored in $\mathcal{S}_b$ is a pair $(\alpha_b, \mathcal{S}_{b+1})$, where α_b is a configuration of $\mathcal{S}_b$, $\mathcal{S}_{b+1} \in \Phi_A(\mathcal{S}_b)$, and such that for each $t > 0$, there exists a t'-horizon strategy for DS w.r.t the property $\mathcal{P}$, preceded by $((\mathcal{S}_0, \alpha_0), \ldots, (\mathcal{S}_b, \alpha_b))$ and anchored in $\mathcal{S}_{b+1}$, with $t' \geq t$.

Last, a strategy for DS w.r.t the property $\mathcal{P}$ is an infinite horizon strategy for DS w.r.t the property $\mathcal{P}$, anchored in $\mathcal{S}_0$.

Noteworthy, a strategy for a dynamic system w.r.t. some property $\mathcal{P}$ consists in finding a configuration to the initial system S_0 (i.e., configuring the system S_0) and specifying a move a which is among the possible ones in S_0, in such a way that whichever events occur in the future, we are able to specify a state trajectory satisfying the property $\mathcal{P}$. A specific scenario will proceed as follows, for each time step i:

- the dynamic system is in the system state S_i;
- the system's controller chooses an action $a \in poss(S_i)$;
- the environment selects a system S_{i+1} from $\Phi_A(S_i, a)$;
- the system's controller specifies a configuration α_{i+1} of S_{i+1};
- the dynamic system moves accordingly to the system state (S_{i+1}, α_{i+1}).

The set of all expected scenarios can be developed as a tree structure rooted in S_0, as shown in the following example:

Example 15 (*Continued*) Figure 6 gives the tree representation of a strategy within the dynamic system graphically represented in Fig. 5. The system's controller initially defines the root of the tree by choosing a configuration α_0 of $\mathcal{S}_0$, for instance, setting the size of all available servers at $Medium$. Then she selects a specific move from $poss(\mathcal{S}_0)$, i.e., the move *nop* in this example, representing the fact that no server is added or removed from the system. At the next time step one of the two systems $\mathcal{S}_0$, A may be obtained as the result of performing the move *nop* in $\mathcal{S}_0$, depending on the situation: one may fall into (i) the system A, different from S_0 in the sense that the set of variables has changed (one (or more) server(s) fall(s) down), or that there is a change in the constraints (there is an increasing of the resources demand). In any of these systems, a strategy must exist, i.e., a configuration must be chosen (i.e., a memory size given to each server, α_1 for the system A, α'_1 for the system $\mathcal{S}_0$), as well as a decision (a_1 for the system A, *nop* for the system $\mathcal{S}_0$). The process is iterated at each level of the tree. If we assume that all state trajectories represented in the tree satisfy some underlying property $\mathcal{P}$, then each node of the tree represents a strategy w.r.t. $\mathcal{P}$ preceded by the path from the root to this node and anchored in the system specified in the node: here the strategy employed for DS w.r.t. $\mathcal{P}$ is the pair (α_0, nop).

It is interesting to note that considering the restricted case where each system is formed of one single unary variable whose cost may have only two values (e.g., 0 for "stable" and 1 for "unstable"), a dynamic system in our sense coincides with a DEDS, and our notion of strategy coincides with the one in the field of DEDS [5]. However, in our definition there are two levels of control in a strategy, i.e., an additional choice of a configuration in a given system. As a consequence, different decisions may have to be taken from the same system depending on the current configuration (for instance, the strategy depicted in Fig. 6 assigns the decision a_1 in the system state (A, α_1) and the decision *nop* in the system state (A, α'_2)).

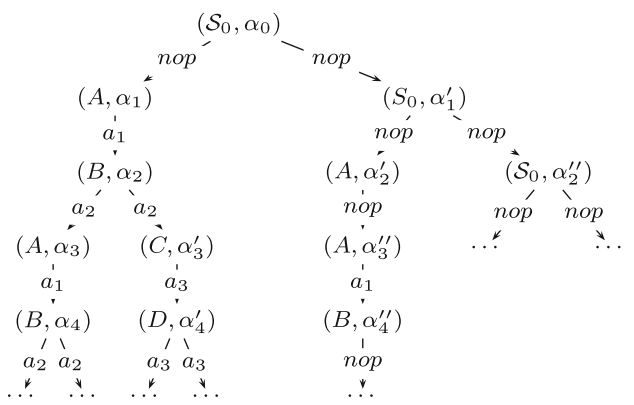


Fig. 6 The representation of all expected scenarios as a tree structure

We are ready to extend the properties defined in the previous section for state trajectories into dynamic systems:

Definition 15 Let $\mathcal{P}$ a set of properties among those presented in Sect. 4. A dynamic system DS satisfies the set $\mathcal{P}$ of properties if there is a strategy for DS w.r.t. $\mathcal{P}$.

Example 16 (*Continued*) For instance, let k be a positive integer and l be a non-negative number. Then one can assess that our dynamic system DS is $\langle k, l\rangle$-resilient if one can find a strategy depicted through a tree as in Fig. 6, such that all paths rooted in the initial system state (S_0, α_0) (i.e., all systems state trajectories which can be induced from this strategy) are $\langle k, l\rangle$-resilient. That is to say, one can *guarantee* that whichever state trajectory will be followed, it will be $\langle k, l\rangle$-resilient.

One can now consider again the notion of k-resiliency introduced in the previous section for state trajectories (cf. Definition 9) and adapt the notion to dynamic systems:

Definition 16 (*k-Resiliency of a dynamic system*) Given a dynamic system DS and a positive integer k, the *k-resiliency* of DS is the smallest number l for which DS is $\langle k, l\rangle$-resilient.

If l is the k-resiliency of a dynamic system DS, it means that there exists a configuration α_0 and a move $a \in poss(\mathcal{S}_0)$ from which one has the guarantee to build a $\langle k, l\rangle$-resilient state trajectory in any possible future scenario.

Definition 17 (*Resiliency function of a dynamic system*) The *resiliency function* of a dynamic system DS, denoted res_{DS}, is the mapping from $\mathbb{N}_*$ to $\mathbb{R}^+$ such that for every $k \in \mathbb{N}_*$, $res_{DS}(k)$ is defined as the k-resiliency of DS.

At the level of state trajectories, it can be easily seen that the resiliency function can be computed in time quadratic to the size of the underlying state trajectory (for each k varying from 1 to the size of the state trajectory, the k-resiliency can be computed in time linear to its size). However, it may be computationally hard to determine the resiliency function of a dynamic system, as the number of state trajectories grows exponentially with the time horizon. This complexity issue is important for the design of resilient dynamic systems when considering practical applications, and this is out of the scope of this paper and will be investigated in a future work.

6 Discussion

In this section, we further discuss our contribution with regard to the existing literature on dynamic systems and their properties. We summarize some previous work tackling some aspects of resilience in DEDS. Then we relate our model to some notions of robustness and stability in dynamic constraint satisfaction problems.

6.1 Stabili(zabili)ty in discrete event dynamic systems

DEDS are standard structures to represent a dynamic world supervised by an agent [5,12,31,34]. Our notion of dynamic system is similar to the one of DEDS. However, the dynamics underlying a DEDS does not allow an explicit occurrence of decision/action and exogenous events at the same time. In our model, the system evolves with respect to time, and the occurrence of decisions and exogenous events are combined within a time step through the non-determinism of the decisions.

As to properties of dynamic systems in terms of resilience, closely related to our framework are [5,31]. In the beginning of the 1990s, Özveren et al. [31] dealt with some properties of resilience in a class of discrete-event systems modelled as non-deterministic finite state automata where only some of the transitional events are directly observed. The authors addressed the problem of *maintenance goals*, where for an agent situated in an dynamic environment, the goal is to reach one out of the available "stable" states under certain restrictions. More precisely, they proposed two properties as part of resiliency or error-recovery, namely *stability* and *stabilizability*, which underlie the capability of maintaining such stable states in a DEDS. Given a set of "good" states, stability is defined in a sense where the system will finally visit "good" states infinitely often, from any set (or subset) of system states; a DEDS is stable if all its reachable states satisfy the property of stability. Stabilizability focuses on an agent and its ability to make the system stable, which intuitively means that there is a "control policy" which the agent can use to design a stable system. This definition is consistent with the one given by Djisktra [15] about self-stabilizing systems, which concerns systems in which there is a guarantee of reaching a safe state from any state within a finite number of transitions. This property differs from our notion of stabilizability (cf. Sect. 4.6): indeed, in our framework stabilizability underlies the evaluation of a transition degree from a system state to its successor, while in [31] it is based on the reachability of some stable states. In fact, the notion of stabilizability in [31] can be interpreted as a much weaker form of our notion of recoverability.

Additionally in [5], the authors have introduced the notion of *k-maintainability*. An example in support of the intuition behind the rationale of k-maintainability is a person who is asked to "maintain" a room clean. By assuming that the job can be done only when this room remains unoccupied for a certain time; one cannot blame the cleaning person that the room is not clean if he or she is continually sent away because the room is continuously being used. In this example, if we are given a window of non-interference of "size" k during which maintenance is performed by the agent, and if a stable state can be reached within k steps under these conditions, then the dynamic system is said to be k-maintainable. Though this property is closely related to stabilizability in the sense of Özveren et al. [31], it is not directly relevant with respect to the resilience of dynamic systems in our framework where the exogenous events should be dealt with in any circumstances.

6.2 Robustness and stability of solutions in dynamic constraint satisfaction problems

The notions of *stability* and *robustness* are recurrent in Dynamic CSPs (see Sects. 2.2 and 2.3 for an introduction of CSPs and Dynamic CSPs) [10,47,48]. These notions do not apply on the Dynamic CSPs themselves, but on their solutions. On the one hand, robust solutions refer to solutions that are more likely to remain valid after some change within the constraints. On the other hand, the property of stability is based on a distance between assignments, typically the Hamming distance which we considered in Sect. 4.6; then, stable solutions are guaranteed to produce a new valid solution with only few assignment modifications. The property of stability in this sense is similar to that of stabilizability which we proposed. But there exist several differences between stability and robustness on the one hand, and all the properties we proposed in this paper (including stabilizability) on the other hand. Indeed, our properties are defined on *state trajectories* rather than on configurations (solutions), i.e., those of the initial system. Moreover, in our framework the configurations are associated with a cost, allowing us to analyse a state trajectory in terms of resistance, recoverability, functionality and resilience. This is not the case when considering robust and stable solutions DCSPs where only the differences in assignments is considered, and not the "quality" of such each assignment within each CSP. Furthermore, to the best of our knowledge, no research has been conducted on how to assess resilience-related properties of Dynamic COPs.

7 Conclusion and perspectives

In this paper, we introduced the topic of systems resilience and defined a new framework which can be used to represent resilient dynamic constraint-based systems. Our work aims at formalizing resilience in a model that makes a bridge between constraint-based systems and dynamic systems. We defined the notion of resilience on the level of state trajectories, i.e., on specific scenarios which may be adopted by the dynamic system. For this purpose, we reviewed a range of qualitative definitions of resilience from the literature and compiled them into several factors, mainly resistance, recoverability and functionality. Then we captured these notions through a unifying parametrized property, simply named *resilience*. We extended these properties from state trajectories to dynamic systems by considering the notion of a "strategy" operated by the system's controller through time.

Two successive moves of different kinds are performed by the system's controller, at each time step. The first move consists in configuring the current system. The second move consists in performing an action in attempt to modify the constraints of the system at the next time step, given that these actions are non-deterministic because of the possible occurrence of exogenous events, a central consideration in this paper. Then, a dynamic system is resilient if there exists a strategy for it that provides a guarantee that all state trajectories possibly adopted by the system are resilient. Overall, we provided a generic framework which allows us to assess the resilience of constraint-based dynamic systems, which is a crucial step prior the *design* of a reliable system or Intelligent Environment.

Noteworthy, our framework does not address the resilience of Intelligent Environments in all layers. For instance, issues on improving the system 's *perception* of its environment are challenging and not addressed in our approach. Indeed, we assumed here that the given dynamic system is "correct" and that the system's controller (intelligent software or human supervisor) is capable of knowing in which exact system state one lies at each time step, to perform a given strategy; however, partial observability would be a more realistic parameter to take into account.

Computational complexity will also be an important research direction for further work, e.g., investigating the inherent complexity of problems such as checking whether a given dynamic system is $\langle k, l \rangle$-resilient or not, or computing the best trade-off strategies with respect to resilience and stabilizability. Moreover, in the current version of our framework, we assumed that our systems were completely observable. But in reality, we may only have uncertain information about them. Therefore, models for the probabilistic reasoning on dynamic systems, such as partially observable Markov decision processes (POMDPs) and dynamic Bayesian networks (DBNs), may need to be incorporated into our framework. Instead of knowing exactly in which state the system is, we may just have a belief on those possible states based on the observations made and actions taken, specified by a probability distribution. The computation of optimal strategies based on these beliefs will be considered as a future work.

References

1. Ali S, Koenig S, Tambe M (2005) Preprocessing techniques for accelerating the DCOP algorithm ADOPT. In: Proceedings of the 4th international conference on autonomous agents and multi-agent systems (AAMAS'05), pp 1041–1048
2. Apt K (2003) Principles of constraint programming. Cambridge University Press, New York
3. Augusto JC, Callaghan V, Cook D, Kameas A, Satoh I (2013) Intelligent environments: a manifesto. Hum-Centric Comput Inf Sci 3(2):1–18
4. Ballarini P, Miller A (2006) Model checking medium access control for sensor networks. In: Proceedings of the 2nd international symposium on leveraging applications of formal methods (ISoLA'06), pp 256–262
5. Baral C, Eiter T, Bjäreland M, Nakamura M (2008) Maintenance goals of agents in a dynamic environment: formulation and policy construction. Artif Intell 172(12–13):1429–1469
6. Benveniste A, Caspi P, Edwards SA, Halbwachs N, Guernic PL, de Simone R (2003) The synchronous languages 12 years later. Proc IEEE 91(1):64–83
7. Bilal K, Manzano M, Khan SU, Calle E, Li K, Zomaya AY (2013) On the characterization of the structural robustness of data center networks. IEEE Trans Cloud Comput 1(1):1–1
8. Bruneau M (2003) A framework to quantitatively assess and enhance the seismic resilience of communities. In: Earthquake spectra, vol 19
9. Cicchetti D (2010) Resilience under conditions of extreme stress: a multilevel perspective. World Psychiatry 9(3):145–154
10. Climent L, Wallace RJ, Salido MA, Barber F (2014) Robustness and stability in constraint programming under dynamism and uncertainty. J Artif Intell Res 49:49–78
11. De Florio V (2015) On resilient behaviors in computational systems and environments. J Reliab Intell Environ 1(1):33–46
12. De Schutter B (2001) Stability analysis of discrete event systems. Automatica 37(5):799–801
13. Dechter R (2003) Constraint processing. Morgan Kaufmann Publishers Inc., San Francisco
14. Dechter R, Dechter A (1988) Belief maintenance in dynamic constraint networks. In: Proceedings of the 7th national conference on artificial intelligence (AAAI'88), pp 37–42
15. Dijkstra EW (1974) Self-stabilizing systems in spite of distributed control. Commun ACM 17(11):643–644
16. Folke C, Carpenter SR, Walker B, Scheffer M, Chapin T, Rockström J (2010) Resilience thinking: integrating resilience, adaptability and transformability. Ecol Soc 15(4):20
17. Greenberg A, Hamilton J, Maltz DA, Patel P (2008) The cost of a cloud: research problems in data center networks. ACM SIGCOMM. Comput Commun Rev 39(1):68–73
18. Grimm V, Calabrese JM (2011) What is resilience? A short introduction. In: Viability and resilience of complex systems, understanding complex systems. Springer, Berlin, pp 3–13
19. Grimm V, Wissel C (1997) Babel, or the ecological stability discussions: an inventory and analysis of terminology and a guide for avoiding confusion. Oecologia 109(3):323–334
20. Haimes YY, Crowther KG, Horowitz BM (2008) Homeland security preparedness: balancing protection with resilience in emergent systems. Syst Eng 11(4):287–308
21. Hamming RW (1950) Error detecting and error correcting codes. Bell Syst Tech J 29:147–160
22. Henzinger TA, Sifakis J (2006) The embedded systems design challenge. In: Formal methods (FM'06), pp 1–15
23. Holling C (1973) Resilience and stability of ecological systems. Annu Rev Ecol Syst 4:1–23
24. Junges R, Bazzan A (2008) Evaluating the performance of DCOP algorithms in a real world, dynamic problem. In: Proceedings of the 7th international conference on autonomous agents and multi-agent systems (AAMAS'08), pp 599–606
25. Kopetz H (2011) Real-time systems: design principles for distributed embedded applications. In: Real-time systems series. Springer, US
26. Linkov I, Eisenberg DA, Bates ME, Chang D, Convertino M, Allen JH, Flynn SE, Seager TP (2013) Measurable resilience for actionable policy. Environ Sci Technol 47(18):10108–10110

27. Linkov I, Eisenberg DA, Plourde K, Seager TP, Allen J, Kott A (2013) Resilience metrics for cyber systems. Environ Syst Decis 33(4):471–476
28. Longstaff PH, Armstrong NJ, Perrin K, Parker WM, Hidek MA (2010) Building resilient communities: a preliminary framework for assessment. Homel Secur Aff 6(3)
29. Maheswaran R, Tambe M, Bowring E, Pearce J, Varakantham P (2004) Taking DCOP to the real world: efficient complete solutions for distributed multi-event scheduling. In: Proceedings of the 3rd international conference on autonomous agents and multi-agent systems, pp 310–317
30. Marrero W, Clarke E, Jha S (1997) Model checking for security protocols. Technical report, Carnegie Mellon University, Pittsburgh
31. Özveren CM, Willsky AS, Antsaklis PJ (1991) Stability and stabilizability of discrete event dynamic systems. J ACM 38(3):7300–7752
32. Özveren CM, Willsky AS, Antsaklis PJ (1991) Stability and stabilizability of discrete event dynamic systems. J ACM 38(3):729–751
33. Puterman ML (1994) Markov decision processes: discrete stochastic dynamic programming, 1st edn. Wiley, New York
34. Ramadge PJ, Wonham WM (1987) Supervisory control of a class of discrete event processes. SIAM J Control Optim 25(1):206–230
35. Ritchey RW, Ammann P (2000) Using model checking to analyze network vulnerabilities. In: IEEE symposium on security and privacy, pp 156–165
36. Russell SJ, Norvig P (2003) Artificial intelligence: a modern approach, 2 edn. Pearson Education, UK
37. Schiex T, Fargier H, Verfaillie G (1995) Valued constraint satisfaction problems: hard and easy problems. In: Proceedings of the 14th international joint conference on artificial intelligence (IJCAI'95), pp 631–639
38. Schwind N, Okimoto T, Inoue K, Chan H, Ribeiro T, Minami K, Maruyama H (2013) Systems resilience: a challenge problem for dynamic constraint-based agent systems. In: Proceedings of the 12th international conference on autonomous agents and multi-agent systems (AAMAS'13), pp 785–788
39. Sharma A, Sharma D (2012) Solving dynamic constraint optimization problems using ichea. In: Neural information processing, vol 7665. Springer, Berlin, pp 434–444
40. Smith P, Hutchison D, Sterbenz JPG, Schller M, Fessi A, Karaliopoulos M, Lac C, Plattner B (2011) Network resilience: a systematic approach. IEEE Commun Mag 49(7):88–97
41. Stoicescu M, Fabre J-C, Roy M (2011) Architecting resilient computing systems: overall approach and open issues. In: Troubitsyna E (ed) Software engineering for resilient systems. Lecture notes in computer science, vol 6968. Springer, New York, pp 48–62
42. Taleb NN (2008) The black swan: the impact of the highly improbable. Random House Inc., New York
43. Tomlin CJ, Lygeros J, Sastry SS (2000) A game theoretic approach to controller design for hybrid systems. Proc IEEE 88(7):949–970
44. Verfaillie G, Pralet C, Lemaître M (2010) Constraint-based modeling of discrete event dynamic systems. J Intell Manuf 21(1):31–47
45. Verfaillie G, Schiex T (1994) Solution reuse in dynamic constraint satisfaction problems. In: Proceedings of the 12th national conference on artificial intelligence (AAAI'94), pp 307–312
46. Walker BH, Holling CS, Carpenter SC, Kinzig AP (2004) Resilience, adaptability and transformability. Ecol Soc 9(2):5
47. Wallace RJ, Freuder EC (1998) Stable solutions for dynamic constraint satisfaction problems. In: Proceedings of 4th international conference on principles and practice of constraint programming (CP'98), pp 447–461
48. Wallace RJ, Grimes D, Freuder EC (2009) Solving dynamic constraint satisfaction problems by identifying stable features. In: Proceedings of the 21st international joint conference on artificial intelligence (IJCAI'09), pp 621–627
49. Wildavsky A (1991) Searching for safety. Transaction Publishers, New Brunswick
50. Wohlgemuth S (2014) Adaptive user-centered security. In: Availability, reliability, and security in information systems. Lecture Notes in Computer Science, vol 8708, pp 94–109
51. Wohlgemuth S (2014) Is privacy supportive for adaptive ICT systems? In: Proceedings of the 16th international conference on information integration and web-based applications and services (iiWAS'14), pp 559–570
52. Wong KC, Wonham WM (1996) Hierarchical control of discrete-event systems. Discret Event Dyn Syst 6(3):241–273