Dissecting Self-* Properties

Andrew Berns and Sukumar Ghosh
The Department of Computer Science
The University of Iowa
Iowa City, Iowa, USA
andrew-berns@uiowa.edu, sukumar-ghosh@uiowa.edu

***Abstract*—The scale and complexity of distributed systems have steadily grown in the recent years. Management of this complexity has drawn attention towards systems that can automatically maintain themselves throughout different scenarios. These systems have been described with many terms, such as self-healing, self-stabilizing, self-organizing, self-adaptive, self-optimizing, self-protecting, and self-managing. These attributes are collectively referred to as self-* properties. Even with the increased focus on self-* research, there exists much ambiguity in the perceptions of the different self-* properties. In this paper, we propose to resolve the ambiguity by introducing a template for defining self-* properties, and use it to offer formal definitions of existing self-* terms. We then present some observations about the relationships among the different self-* properties. Finally, we propose two new self-* properties that are meaningful in this space.**

***Keywords*-Masking and non-masking tolerance; safety and liveness properties; self-healing; self-stabilizing; self-organizing; self-adaptive systems**

I. Introduction

The recent past has been a productive time for distributed systems research. The lowering cost of components has made distributed computing more practical, and large systems have witnessed significant growth. Furthermore, the Internet's evolution into a common staple in households today has created a demand for other types of distributed systems of a tremendous scale, such as peer-to-peer systems and cloud computing.

As the scale and complexity of distributed applications have grown, their management also has become more challenging. For instance, the recent trend of cloud computing relies heavily on large clusters of servers. Amazon's EC2 [1] provides huge amounts of computing power to many customers, including imageloop.com [2] (a photo-sharing website) and SnappyFingers [3](a search engine). These applications expect reliable and fast service at low cost, even though their server instances may be running on a complex system containing hundreds of computers. Skype delivers audio and video to nearly 200 million users over peer-to-peer networks, and is clearly a success story. It is not hard to imagine that, without sophisticated techniques, managing this level of complexity to provide reliable performance for so many customers would become next to impossible.

To relieve some of the burden of complexity, the current trend is towards maintaining at least some aspect of operation automatically, without human intervention. These systems have been called "self-healing," "self-protecting," "self-stabilizing," "self-organizing," and more. Collectively, such properties are referred to as "self-*" properties. Autonomic computing, an initiative started by IBM in 2001, is a vision to create systems that implement a collection of these self-* properties [4], [5]. However, in spite of the increased attention, there exists considerable ambiguity about the perception of the precise meaning of these properties. Such ambiguities are likely to become a troublesome barrier in formalizing research in this area. This paper attempts to resolve these ambiguities.

Our contribution is threefold. First, we create a common framework to define and analyze the various self-* properties, and map the current definitions into this framework. Second, on the basis of this framework, we explore the inter-relationship among the various self-* properties, along with appropriate examples. Finally, we use the framework to introduce two new self-* properties that can exist in this space.

The paper has six sections. Section 2 introduces the model and the notations. Section 3 casts the different known self-* properties into a common framework. Section 4 introduces two new self-* properties. Examples of how various self-* properties are related to one another are discussed in section 5. Finally, Section 6 contains some concluding remarks.

II. The Model

The system under consideration consists of n processes ($n \geq 1$), and the graph $G = (V, E)$ represents its topology, with vertices V representing processes, and edges E representing its communication links. For each process $i \in V$, let $N(i)$ represent the neighbors of i: thus $(i, j) \in E \Leftrightarrow j \in N(i) \wedge i \in N(j)$. Each process i executes a program that consists of one or more guarded actions $g \rightarrow A$, where g is a predicate involving the variables of i and those of its neighbors, and A is an action that updates the local state $s(i)$ of i. The global state S (also called a *configuration*) consists of the local states of all the processes. A *computation* is a finite or infinite sequence of global states that satisfies two properties: (a) if S and S' are two consecutive states in the sequence, then there exists a process i such that i has an enabled guard in S and execution of the corresponding action results in the state S', and (b) if the sequence is finite, then in the last state of the sequence, no process has an enabled guard.

Divide the set of of all possible actions into two classes: *internal* and *external*. An internal action is a system action as described in the previous paragraph. An external action belonging to the set F, on the other hand, is an action caused

by an adversary, and has a wider scope. It can change the *environment* R: the environment consists of parameters that the system can only read, but cannot modify using the internal actions. External actions include the set of internal actions and can modify any process' internal state, but they can also crash processes, induce failures by taking ad-hoc actions, change the network topology, put the system in an arbitrary configuration, launch security attacks, or change user demands for service from the system, etc. External actions that affect the performance of a system are also called *fault actions* or *adversarial actions*.

The correctness of a system is specified by its *safety* and *liveness* properties [6]. Informally, a safety property implies that " bad things never happen to the system," and is characterized by an invariant P. A liveness property says that "good happens eventually happen," examples are progress, termination, or convergence to a desired configuration. While the violation of a safety property can be determined by examining a finite prefix of the computation, the violation of the liveness property cannot determined in a similar manner. A system is in a *legal* or *consistent* configuration w.r.t. a given environment R, when the corresponding safety predicate P holds. An external or adversarial action can affect the safety or liveness properties of a system. Based on how these properties are affected and how they are restored, one can define four different types of tolerances [7] (1) masking, (2) non-masking, (3) fail-safe, and (4) graceful degradation. A system exhibits *masking tolerance* w.r.t. an action C, when both safety and liveness properties are unaffected by C. In *non-masking tolerance*, the safety property is violated by some external action in F, but not the liveness, and eventually the safety property is restored. A system is *fail-safe* w.r.t. an action in F, if the liveness properties are compromised but not its safety properties. Finally, in a *gracefully degrading* system, the external action affects the safety properties (but not its liveness), and the system eventually recovers to a configuration that satisfies a weaker predicate P' ($P \subset P'$) that is acceptable to the application.

III. The self-* framework

Based on the model developed in Section 2, we analyze the different self-* properties.

A. Self-management

Self-management is a vision. It describes a system that has at least one self-* property [8]. It can be used as a generic property that encompasses the various classes of "autonomic" behavior. Self-managing, then, can be thought of as the parent or superclass of all self-* properties. The definition of self-management will serve as a framework for specifying other self-* properties.

Definition 1. *A system is self-managing with respect to a subset of external actions $C \subseteq F$, if it automatically (without human intervention) maintains, improves, or restores the safety property P following the occurrence of actions in C.*

The definition is a loose one. It does not specify if the tolerance is masking or non-masking, nor does it state anything about whether a gracefully degraded end configuration is acceptable. It is also silent about the time needed to recover (for non-masking tolerance), as long as recovery is guaranteed in a bounded number of steps. Such additional considerations will lead to a further refinement of the above definition.

Refinement occurs by modifying at least one of the three parts of the definition. The first part is the external actions C the system responds to. The second portion is whether the system maintains, improves, or restores some property P. Obviously, improving a property implies the presence of some objective function (the term *improve* is used to mean improving a function up to its maximum, or minimum, or the most desirable value). The final part of the self-management template is the property P being maintained, improved, or restored. By refining these three components, other self-* properties can be defined.

B. Self-stabilization

Self-stabilization was first introduced in 1974 by Dijkstra [9], and it has been an active research topic [10] for many years. It is a form of non-masking tolerance that refers to the ability of a system to spontaneously recover from any initial configuration.

Definition 2. *A system is self-stabilizing if, (1) starting from an arbitrary initial configuration, it recovers to a legal configuration where its safety predicate P holds, and (2) remains in that configuration thereafter.*

The arbitrary configurations can be generated by ad-hoc initialization, transient failures caused by any external actions in F. Such failures however, are ordinarily not expected to affect the program codes of the different processes. Correct solutions argue why the external actions cannot create deadlocks o livelocks, and demonstrate the convergence property (from any configuration to a legal configuration) and the closure property (of the legal configurations). Code corruption leads to byzantine behavior (that is irreversible and needs to be masked), and certain forms of security attacks are capable of modifying program codes. However, most self-stabilizing systems do not address it. That said, work has been done on the feasibility of self-stabilization where external actions include the ability to corrupt the program code [11]

External actions can also crash processes. However, to deal with crash failures, we need to adopt a broader view of a self-stabilizing system by transforming it into a *self-adaptive* system. A self-adaptive system reacts to changes in the environment by changing its safety predicate accordingly [12].

Lemma 1. *Let $\{R_1, R_2, \ldots, R_m\}$ be a set of boolean environment variables, and let P_i be the desired safety predicate when the environment R_i holds. Then the self-adaptive system is equivalent to a self-stabilizing system, for which the predicate*

$$P = \bigvee_{i=1}^{m} (R_i \wedge P_i)$$

will hold.

One can represent the crash of a process i by a boolean $crashed(i)$, which is an environment variable. If an external action sets $crashed(i)$ to $true$, then the system has to recover to another predefined legal configuration. Using Lemma 1, this can be accommodated into the extended view of a self-stabilizing system.

Dolev [10] presents a large number of self-stabilization algorithms for various applications.

C. Self-healing

Self-healing is mostly viewed as a form of *non-masking* tolerance from a limited subset of the set of possible external actions. However, self-healing does not exclude masking tolerance, either. Self-healing systems have been described in various articles like [13]–[16]. However, a *consistent* formal definition of self-healing is lacking.

Definition 3. *A system is said to be self-healing with respect to a subset of external actions $C \subseteq F$ if occurrence of actions in C causes at most a temporary violation of the system's safety property P.*

Self-healing is focused on maintaining or restoring a system's safety properties. When the safety property is violated, healing may take an arbitrary but finite amount of time. However, unlike self-stabilization, most self-healing systems guarantee healing from a limited subset of all external actions. This is in contrast with a self-stabilizing system that guarantees recovery from *all* actions that perturbs its global state. However, crash or Byzantine failures are ordinarily not factored into the definition of a self-stabilizing system, while a self-healing system may cater to such failures.

A self-healing system often exhibits a degraded level of performance when the external action causes a crash failure. An example is a peer-to-peer network that is resilient to external actions causing the removal of nodes [17]. Assuming that at most one node is removed at a time, the solution promises that the network diameter will never be more than a factor of $O(\log \Delta)$ larger than that of the original network (Δ is the maximum degree of a node in the network), and that no node will experience an increase in degree greater than *three*. Such changes will cause a limited degradation of the performance.

The choice of the set of actions w.r.t which self-healing is desirable will depend on the application. The larger the set is, the better is the self-healing property. Recall that the Skype network, which has in-built ability to self-heal from minor failures. However, on Thursday, 16th August 2007, Skype became unstable and suffered a critical disruption. It was triggered by a massive restart of the users' computers across the globe within a very short time period (as they re-booted after receiving a routine set of patches through Windows Update). This caused a flood of log-in requests, which, combined with the lack of network resources, prompted a chain reaction that had a critical impact.[1]

Although most self-healing solutions are non-masking or reactive in nature, there are masking or proactive versions too. They are better known as *predictive self-healing*. These systems anticipate failures from symptoms of erratic behavior, but protects the system from a catastrophic service disruption by internally restarting certain modules and masking the failure from the user. The idea of internally restarting a system via reboot tree was proposed by Candea and Fox [18]. Sun Microsystems's Solaris operating system (and several other operating systems) have this facility.

Interestingly, when the external action alters the network topology G via the addition or deletion of nodes, the system is called *self-organizing* instead of self-healing.

D. Self-organization

Although self-organization is a widely used term across various scientific disciplines, in computer science, *self-organization* is a self-* property that has received increased attention largely due to the increased interest in peer-to-peer systems, ad-hoc networks, and cooperating robot systems or swarm robotics. Self-organization is a non-masking form of tolerance. Several formal definitions of self-organization exist, but with slightly differing views. One requires the system to maintain, or monotonically improve a function value involving the neighbors of the joining or the departing process in-between process joins and leaves [19]. A second definition states that a self-organizing system is a self-stabilizing system that recovers in *sub-linear time* following each join or leave, and that process joins and leaves cause only local changes to the system [20]. The time constraint imposed in [20] is not universally accepted, but its motivation is to make the algorithm scalable, so we will include it in our definition. Another issue is the number of concurrent join or leave operations supported by the algorithm. Our definition will reflect the strongest view.

Definition 4. *A system with n processes is self-organizing, if it maintains, improves, or restores one or more safety properties P following the occurrence of a subset of external actions $C_j \subset F$ involving the concurrent join of up to n processes, or the concurrent departure of up to $\frac{n}{2}$ processes, with a sublinear recovery time per join or leave.*

As an example of self-organization, consider the peer-to-peer system Chord [21]. It provides guarantees on recovery from concurrent failure happening with probability $\frac{1}{2}$, and efficient lookups even when the size of the system doubles.

E. Self-protection

Self-protection traditionally implies masking fault tolerance w.r.t external actions that have malicious intent, by applying

[1] See Villu Arak: What happened on August 16. http://heartbeat.skype.com/2007/08/what_happened_on_august_16.html

trust mechanisms or privacy policies to secure the system and its data [22], [23]. This leads to the following definition:

Definition 5. *A system is self-protecting, if it continuously maintains a safety property P in the presence of a subset of external actions with malicious intent, $C_m \subset F$.*

This definition is quite generic, and similar to the self-management template. The safety property can be any predicate related to the integrity of the system and its data. As with many definitions, the choice of the malicious external actions and the safety properties that are safeguarded, are critical to judging the merit of an implementation of self-protection.

As an example of a self-protecting system, consider the OceanStore archival storage built on a peer-to-peer network [24]. OceanStore provides a large storage system, and ensures that data is available and secure, regardless of machine failures and attempted data theft. In this sense, OceanStore is self-protecting. The predicate it maintains is data availability, and compliance to access constraints for each file, and the actions are machine failures or access control violations. Karlof and Wagner [25] described novel ways of launching attacks on wireless sensor networks, along with methods of protecting the system from such attacks.

F. Self-optimization

While the previous self-* properties are concerned with the maintenance or restoration of a safety predicate, self-optimization focuses on the value of an *objective function* as the system property, and aims at maximizing (or minimizing) this objective function. This matches the intuitive notion of optimization – striving for improvement whenever possible. The following definition captures this natural idea.

Definition 6. *A system is self-optimizing if, starting from an arbitrary initial configuration, it improves (i.e. maximizes or minimizes as appropriate) the value of a predefined objective function of its global state.*

The goal is to maintain the system in an optimal configuration. It is possible that the system was sub-optimal at the time of creation, or the system started in an optimal configuration, but a set of external actions made it sub-optimal.

Just as actions can be two types, internal and external, the objective function also can be of two types: global or local. A *global* objective function

$$cost : S \rightarrow \mathbb{N}$$

accepts the global state S as the input, and returns a value concerning the system as a whole. An example is the maintenance of a minimum weight spanning tree (MWST) by a system of processes in a network. As another example, minimizing a system's power consumption [26] is a global function: the system adjusts its processor frequency to minimize power usage, while still responding to requests in an acceptable time. Here, the global cost function is the power consumption, and the external actions are server requests – in the paper, they are HTTP requests.

The other type of utility function is based upon *local* system information, which reflects the selfish nature of processes in large systems spanning multiple administrative domains. Each process i operates with the selfish goal of maximizing its own function $cost(i)$: $S(i) \rightarrow \mathbf{N}$, regardless of the overall system metric

$$cost = \sum_{i=0}^{n-1} cost(i)$$

In many cases, such selfish moves by individual processes lead the system to an equilibrium configuration, called a *Nash Equilibrium* [27], where no process is able to unilaterally improve its cost function any more. In [28] Fabrikant et. al present an interesting example of *network creation game* to illustrate the impact of selfish moves:

> A set of nodes wants to form a network, so that each node can communicate with others at a minimum "cost." Each node chooses a (possibly empty) subset of the other nodes, and lays down undirected edges to them (Fig. 1(a)). The edges, once installed, can be used in both directions, independently of which node paid for the installation. The union of these sets of edges is the resulting network topology. The cost to each node has two components: the total cost of the edges laid down by this node (the number of edges times a constant $\alpha > 0$), plus the sum of the distances from the node to all others. The game tries to capture if the system of nodes will reach a Nash equilibrium, and also computes the *Price of Anarchy*, which is a measure of the deterioration of cost due to selfish actions when compared with *social optimum*, the optimal choice over all equilibrium configurations, for different values of α.

For some self-optimizing systems involving selfish processes, no equilibrium configuration is reached [29]. Fig. 1(b) shows four process trying to form a shortest path rooted tree in a selfish manner, where the edge costs are functions of the process colors. The computation, however, never reaches an equilibrium configuration.

G. Self-configuration

Self-configuration is a non-masking form of tolerance. It refers to a variety of responses of a system, such as changing network topologies [30], [31], changing the geometry of formation in mobile robots, or changing and setting up various software and hardware components [4]. Determining a consistent definition for self-configuration, then, requires a properly-framed definition of a "configuration". For the purposes of our template, define a configuration as a set of connections amongst modules of the system. Notice that what constitutes a "module" is purposely left open-ended - a module may be a particular software package, a server, or a process in the system. Also "connections" may be physical, or virtual, or any other form of neighborhood relationship.

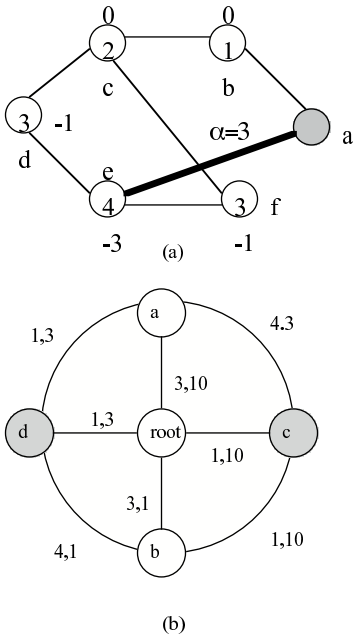

Fig. 1. (a) Selfish nodes creating a network. Node a adds the link to node e by paying $\alpha = 3$, since its overall cost goes down. (b) Four selfish processes are trying to set up a shortest path tree towards the root. The black processes and the white processes have different edge costs (which are labeled as `white`, `black`). This system will never reach an equilibrium configuration.

Definition 7. *A system is self-configuring with respect to a set of actions $C \subset F$, if it is able to change its configuration to restore, or improve some safety property defined over the configuration space.*

In a way, this is quite similar to self-optimization, except that the common perception, so far, does not indulge selfish views of the predicate. Self-configuration may be a reaction to any set of external actions, as long as the reaction changes the configuration.

In [32], the idea of a self-configuring system is used to maximize performance of a website during a specified benchmark. The system consists of an application server, an image server, and a database server, all three of which share a limited number of memory and processor cycles. As the workload changes, the system changes the allocation of memory and processor cycles to each component, which results in better performance (such an idea is increasingly important considering the surge in cloud computing and virtualization). For this system, the set of actions is the workload contained in the benchmark, and the system property being improved is system response time (measured in the benchmark). The different configurations correspond to different allocations of memory and processor - each of the three components can be thought of as "connected" to a certain number of memory and processor "modules" (notice that if a single physical machine is used, these "modules" are actually software pieces). This is an excellent example of a self-configuring system.

H. Self-scaling

Self-scaling refers to a system's ability to perform well under many different system sizes [33]–[36]. The definition of self-scaling must capture this notion of a change in size, but at the same time must not lose the generality to be applied to the variety of existing systems that are self-scaling. One must carefully distinguish between the notions of self-organization and self-scaling, since both deal with changes in system scale. The difference is in the definition of the predicates: self-organization primarily caters to "topological properties," (like setting the neighborhood or routing tables correctly) whereas self-scaling caters to "functional properties" as perceived by the users or the application. Self-scaling systems however, may need to modify topology to handle a change in size, when it is unable to handle changes via internal modifications.

Definition 8. *A system is self-scaling with respect to a set of external actions influencing its scale $C_s \subset F$, if it maintains or improves a system property P during the occurrence of those actions.*

One cannot ignore the resemblance between self-scaling and self-optimization, although self-optimization does not exclusively deal with actions affecting the system scale.

An example of a self-scaling system is load balancing in structured P2P systems [37]. Many P2P systems are made up of heterogeneous nodes, which may have different demands and capabilities. Therefore, it is helpful if a mechanism exists to maximize the available resources from all nodes. The nodes that are operating beyond their desired capacity are considered "heavy" nodes, while those under their capacity are "light". The property being improved, then, is the function $(n - h)$, where n is the number of nodes in the system, and h is the number of heavy nodes (the goal is to have no heavy nodes). This involves the moving of objects within the P2P system.

I. Excluded Terms

The above list does not include a definition of every self-* term used in previous research. Many of these exclusions, however, are purposeful, and were done for one of several reasons. One reason for excluding a term is that a synonym for the term has already been defined. For instance, one could think of self-repairing [38] to be a synonym for self-healing. Also, self-adjusting [39] can be considered a synonym of self-management.

Another reason some self-* terms were excluded from this discussion is that some self-* terms do not represent the behavior of the system. Self-awareness, a term that has a previous definition [40], is a good example of this. A system may be self-aware, but its behavior may be identical to a system which is not. Self-awareness, then, is a property the system may have and use, but it implies nothing about the system's behavior. Self-monitoring [41] is similar to self-awareness, and therefore was excluded for the same reason. This paper focused instead of those self-* terms that fall into the self-management category.

IV. New Self-* Properties

In this section, we revisit the space of self-* properties, and introduce two new self-* properties.

A. Self-immunity

Self-immunity is still a relatively unexplored self-* term. The primary goal of self-immunity is to introduce a *learning component* in a system. This will enable the processes to invoke the tolerance mechanism at run-time on an as-needed basis, instead of statically overloading a non-immune system with such tolerance mechanism regardless of whether the faults occur. As a result, the system incurs less overhead, when failures or perturbation cease to occur, but switches to the masking mode (which is trivially self-immune) when failures of any particular type start occurring. Several previous work focused on artificial immune systems, including those for distributed systems [42]–[44]. These artificial immune systems can be considered to be self-immune, although a system need not imitate the biological immune system to be self-immune, as the definition of self-immunity is based on behavior, not on design.

Definition 9. *A system is self-immune with respect to a subset of external actions $C \subseteq F$, when (1) it restores the safety predicates following an occurrence of an action in C, and (2) eventually, the safety predicates are no more compromised in spite of the occurrence of that type of action.*

The underlying slogan is that "the system learns to get better with time," which strikes a balance between long-term survivability and short-term efficiency. A typical scenario is as follows:

> A mobile ad-hoc network of wireless nodes can become partitioned, if one or more nodes move too far away from the radio range of other nodes. This will cause service disruption. To maintain connectivity, a few other nodes have to increase their transmission power. If the mobility follows some pattern or is predictable, then some nodes will proactively cover for the anticipated period of disruption, and the system becomes immune to service disruptions caused by the drifting node.

Self-immunity is meant to be added to an underlying self-stabilizing, self-organizing, or self-healing system. A system that is self-immune w.r.t. an external action belonging to C, can lose self-immunity when an action not belonging to C occurs. An interesting question is: will it be restored? The following lemma answers this question:

Lemma 2. *A system that is self-immune with respect to a set of actions C, will restore the self-immunity property destroyed by an action not belonging to the set C, only if the underlying system offers non-masking tolerance w.r.t that action.*

This mirrors the intuition on a biological immune system – eventually, the immune system builds up a defense against the actions perturbing the system.

B. Self-containment

Although not explicitly stated, currently all forms of self-protection appear to be masking in nature. However, there is room for a non-masking form of self-protection too. This leads to the introduction of a *self-containing* system:

Definition 10. *A system is self-containing if external malicious actions $C_m \subset F$ compromise a fraction of the processes of a distributed system, but eventually normal operation is restored with the non-compromised processes.*

This definition shadows the definition of *fault-containment* [45] that has been examined in the context of self-stabilizing systems – the objective is damage control and eventual recovery via limiting the impact of the malicious action. Consider a wireless sensor network, and assume that an external node has duped an existing node into routing data to it (i.e. the external node is stealing data from the system), or a node has been physically replaced by a rogue proxy node. If the neighbors of this node detect this event, then they can disconnect itself from the compromised node (or the proxy node), protecting the remainder of the nodes, while restoring connectivity via other paths. Such behavior is useful for places where self-protection is too strict, but self-healing is too relaxed.

Lessons from biology and nature teach us to use heterogeneity in system composition for resilience. For example, the computers in a robust system should not all run the same operating system, but instead use an assortment of operating systems for better survivability. The rationale is that only a small fraction of these will fall prey to an attacker, but eventually all of them will restore their functionality via self-healing.

V. Relationships among self-* properties

The properties defined above are obviously not mutually exclusive. Many of the properties overlap, some are subsets of others, and a system may exhibit multiple self-* behaviors. These relationships are easier to see when using the self-management template, as the three components and their relationships can be derived quickly. Table I helps to show these relationships.

Using the table, some of the relationships become more apparent. For instance, all properties are subsets of self-managing. Self-protection implies self-containment, since the fraction of the system compromised is of size zero. Some properties are not strict subsets, but rather overlap other properties. For instance, self-protection and self-immunity have some overlap for systems that try to maintain a safety predicate P in the presence of repeated malicious actions. In the following section, we examine a selected set of self-* properties in greater detail.

A. Self-stabilization vs. self-healing

Self-stabilization requires the eventual restoration of a safety propertry following the occurrence of *any* transient fault. Self-healing has a broader definition: it can restore a system to a safe configuration only after the occurrence of *certain* faults,

TABLE I
SUMMARY OF SELF-* PROPERTIES. F = SET OF EXTERNAL ACTIONS, P = PREDICATE OVER GLOBAL STATE.

Self-* Property Name	**Adversarial actions**	**Behavior**	**Predicate maintained**
Self-stabilization	All transient failures	Restore	P
Self-adapting	Change of environment R	Restore	**if** R_i **then** P_i
Self-healing	Some $C \subseteq F$	Restore	P
Self-organizing	Process join or leave	Maintain, improve, or restore	P
Self-protecting	Some mailicious actions	Maintain	Predicate over trust
Self-optimization	Some $C \subseteq F$	Improve or maximize/minimize as appropriate	An objective function of the global or local state
Self-configuration	Some $C \subset F$, often includes user demands for service	Maintain, improve, or restore using configuration changes	Predicate over system configuration to optimize performance. Sometimes addresses geometric invariant
Self-scaling	Changes of system scale or demand	Restore, or improve	Predicate over service quality
Self-immunity	Some $C \subseteq F$	Eventually maintain	P
Self-containment	Some mailicious actions	Maintain (for a subset of processes)	Predicate over trust

while the choice can be made from a larger set of failure beyond transient failures. It can also mask such failures. which is beyond the scope of self-stabilizing systems. This leads to lemma 3.

Lemma 3. *Every self-stabilizing systems is trivially self-healing with respect to any action that corrupts its global state, but the reverse is not true.*

B. Self-stabilization vs. self-organization

While both self-stabilization and self-organization are non-masking forms of tolerance, self-organization has a relatively limited agenda, since it only addresses join and leave operations, while self-stabilization caters to all transient failures that can corrupt the global state. Consider the Chord P2P system [21], which is self-organizing. However, Chord is not self-stabilizing – if the system is split into two rings (which could be caused by transient faults), then there is no way the rings can join back together. On the other hand, a system that is self-stabilizing may have a large recovery time (greater than the sublinear bound prescribed in Section 3), so it will fail to meet the recovery time criteria of self-organizing systems. This leads to the following lemma:

Lemma 4. *The intersection between the set of self-stabilizing systems and self-organizing systems is non-empty, although no one completely belongs to the other set.*

C. Self-organization vs. self-configuration

Self-configuration requires a system to change its configuration (often perceived as physical or logical connections amongst modules) to restore, maintain or improve a system property following an arbitrary adversarial action. These connections may be amongst different hardware or software modules, or it may even reflect a geometric invariant. Self-organization, on the other hand, deals exclusively with joins and leaves. This leads to the following lemma.

Lemma 5. *Every self-organizing system is self-configuring, but the reverse is not true.*

For an example of a system that is self-configuration is not self-organization, consider the self-configuring web server in [32]. It changes connections between the server components and processor cycles and memory capacity to provide a stable response, and is obviously self-configuring. However, if another server component is added, then the system will not automatically detect this component and integrate it into the system.

D. Self-immunity vs. self-healing

Self-immunity offers the promise of eventual masking of the effect of an external action, although it may not mask the effect at the beginning. After repeated occurrences of a certain fault, the system is guaranteed to maintain a system property. Self-healing requires a property is either maintained or restored for a certain set of faults. These observations lead to the following lemma:

Lemma 6. *Every self-immune system is self-healing, but the reverse is not true.*

To show that self-healing systems are not always self-immune, consider a system for discovering services. In some systems, such as mobile networks, it is difficult for each member of the system to know fully what services of the system are available. Service discovery protocols help solve this problem – these allow systems to either recover from the loss of a service, and locate another service that may be used [46]. One specific set of examples examined service-

discovery protocols in the presence of communication failures [47]. Such a service discovery protocol makes a system self-healing from communication failures, but does not make it self-immune. This system maintains or restores a safety property (a consistent system view) following the a set of fault actions (communication failure). However, it does not mask these communication faults over time - if a component fails many times after the system has recovered, the system will continue to be perturbed.

E. Self-immunity vs. self-protection

Self-immunity safeguards the safety property by *eventually masking* a certain set of external actions, whereas self-protection *always masks* the effects of external actions deemed to be malicious in nature.

Lemma 7. *Every self-protecting system is self-immune, but the reverse is not true.*

However, self-immunity differs in two key ways from self-protection. First, self-immunity allows a system to develop masking tolerance over time. The system learns about what failures are occuring, and prepares itself to handle the actual faults it may face. Secondly, self-immunity does not apply to malicious actions only - it may provide eventual masking tolerance for other actions as well, although it may not be a good choice in the presence of security threats, since the system becomes vulnerable during the learning phase.

VI. Conclusion

This paper is an attempt to reconcile the various viewpoints about self-* properties by providing a formal foundation. These properties are all closely related in that each is just a special case of self-management. This similarity may be helpful in expanding the field. For instance, self-stabilization has a large existing body of research, and this could used to understand things about other new or less studied self-* properties. The benefit of using a self-management template is to explore meaningful self-* properties that may exist in this space.

There are many grey areas in such a classification. Note that we tried to classify adversarial actions into process crashes, join or leave operations, malicious actions etc. However such a classification is highly subjective. For example, an adversary may crash a sensor node to reduce coverage and cause a security breach, so the crash is essentially a malicious action.

A system can have multiple types of self-* properties w.r.t different sets of adversarial actions. For example, a system may be self-stabilizing w.r.t. any number of transient failures, but self-immune to at most one failure. A self-organizing system can also be self-optimizing

So far, we characterized the system property of interest to be a safety property. A common question is: what if an external action destroys the liveness property, or induces a deadlock? Indeed, some types of fail-safe systems prefer to induce a deadlock when the consequences of the adversarial action are damaging to the application: the goal is to solicit human intervention. For non-masking tolerance, however, recovery is hampered due to the lack of progress. One solution is to modify the basic design so that progress is not affected by the adversarial action of interest. Most self-stabilizing systems use this approach. As an alternative, after a timeout period (whose value should be larger that the maximum possible response time), the system has to be internally reset or restarted. Unless the system is fully asynchronous, the use of timeout is a common tool for detecting crash and initiating reset.

Although not explicitly mentioned, the restoration of a system property often accommodates graceful degradation, particularly when the system has to deal with the crash of processes, or loss of resources, or excessive service demands. To what extent a safety predicate P can be allowed to degrade and still it becomes acceptable to the application, is entirely the designer's choice. For example if P corresponds to servicing a number of requests in the FIFO order, and a weakened predicate P' implies servicing the requests in any order, then some applications may accept the substitution of P by P'. An increase in the space or time complexity by a polylog amount is widely accepted by the community. However, it is also expected that following a repair (or replacement) of a crashed process, the original safety predicate will be restored.

Most definitions still carry with it a certain level of ambiguity by allowing different aspects of the property to depend upon the application. This is evidenced by a common phrase from most definitions - "with respect to." This ambiguity, however, is unavoidable without creating an exponentially-large set of self-* properties. Different systems each have different safe configurations, different environments, and different potential faults. Furthermore, there are an infinite number of contingencies possible with real-world systems. Therefore, by necessity, *any* distributed system operates "with respect to" a set of limitations. In a way, the goal of self-* computing research is to widen this "with respect to" set to include more and more actual possibilities.

One last observation from these definitions is that some system attributes are required for certain self-* properties, but not for others. For instance, a system must have a form of self-awareness to compute a utility function, and therefore a system must be self-aware to be self-optimizing. Conversely, a system need not be self-aware to be self-healing - it may run a healing protocol at periodic intervals regardless of the system state, which would not require self-awareness. This observation matches with a previous work on the need for self-awareness in grid systems [40]. In general, examining these definitions can provide insight into necessary and unnecessary attributes of a system design.

References

[1] Amazon, "Amazon elastic compute cloud (ec2)," 2009, `http://aws.amazon.com/ec2/`.

[2] imageloop.com, "Slideshows, photos, pictures at imageloop.com," 2009, `http://www.imageloop.com`.

[3] SnappyFingers, "Snappyfingers," 2009, `http://www.snappyfingers.com`.

[4] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[5] A. G. Ganek and T. A. Corbi, "The dawning of the autonomic computing era," *IBM Syst. J.*, vol. 42, no. 1, pp. 5–18, 2003.

[6] B. Alpern and F. B. Schneider, "Recognizing safety and liveness," *Distributed Computing*, vol. 2, no. 3, pp. 117–126, 1987.

[7] A. Arora and M. Gouda, "Closure and convergence: a foundation of fault-tolerant computing," *IEEE Transactions on Software Engineering*, vol. 19, pp. 1015–1027, 1993.

[8] R. Sterritt and D. Bustard, "Towards an autonomic computing environment," *Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on*, pp. 694–698, 2003.

[9] E. W. Dijkstra, "Self-stabilizing systems in spite of distributed control," *Commun. ACM*, vol. 17, no. 11, pp. 643–644, 1974.

[10] S. Dolev, *Self-Stabilization*. MIT Press, 2000.

[11] F. C. Freiling, F. C. Freiling, S. Ghosh, and S. Ghosh, "Code stabilization," in *Proc. of the 7th Symposium on Self-Stabilizing Systems (SSS05 Barcelona*, 2005, pp. 128–139.

[12] M. Gouda and T. Herman, "Adaptive programming," *Software Engineering, IEEE Transactions on*, vol. 17, no. 9, pp. 911–921, Sep 1991.

[13] G. S. Blair, G. Coulson, L. Blair, H. Duran-Limon, P. Grace, R. Moreira, and N. Parlavantzas, "Reflection, self-awareness and self-healing in openorb," in *WOSS '02: Proceedings of the first workshop on Self-healing systems*. New York, NY, USA: ACM, 2002, pp. 9–14.

[14] M. Smith and B. Freisleben, "Self-healing wireless ad hoc networks based on adaptive node mobility," in *in Proc. IFIP Conf. on Wireless and Optical Communications Networks*, 2004.

[15] M. Shaw, ""self-healing": softening precision to avoid brittleness: position paper for woss '02: workshop on self-healing systems," in *WOSS '02: Proceedings of the first workshop on Self-healing systems*. New York, NY, USA: ACM, 2002, pp. 111–114.

[16] D. Ghosh, R. Sharman, H. R. Rao, and S. Upadhyaya, "Self-healing systems - survey and synthesis," *Decis. Support Syst.*, vol. 42, no. 4, pp. 2164–2185, 2007.

[17] T. Hayes, N. Rustagi, J. Saia, and A. Trehan, "The forgiving tree: a self-healing distributed data structure," in *PODC '08: Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*. New York, NY, USA: ACM, 2008, pp. 203–212.

[18] G. Candea and A. Fox, "Recursive restartability: Turning the reboot sledgehammer into a scalpel," in *HotOS*, 2001, pp. 125–130.

[19] E. Anceaume, X. Defago, M. Gradinariu, and M. Roy, "Towards a theory of self-organization," in *9th International Conference, OPODIS 2005, Pisa, Italy, December 12-14, 2005, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 3974. Springer, 2006, pp. 191–205.

[20] S. Dolev and N. Tzachar, "Empire of colonies: Self-stabilizing and self-organizing distributed algorithm," *Theoretical Computer Science*, vol. 410, no. 6-7, pp. 514 – 532, 2009, principles of Distributed Systems. [Online]. Available: http://www.sciencedirect.com/science/article/B6V1G-4TN82MN-B/2/98d29445fb55ced02c6e198d1d0f17ca

[21] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2001, pp. 149–160.

[22] C. English, S. Terzis, and P. Nixon, "Towards self-protecting ubiquitous systems: monitoring trust-based interactions," *Personal Ubiquitous Comput.*, vol. 10, no. 1, pp. 50–54, 2005.

[23] R. Ananthanarayanan, M. Mohania, and A. Gupta, "Management of conflicting obligations in self-protecting policy-based systems," *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pp. 274–285, June 2005.

[24] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao, "Oceanstore: an architecture for global-scale persistent storage," *SIGARCH Comput. Archit. News*, vol. 28, no. 5, pp. 190–201, 2000.

[25] C. Karlof and D. Wagner, "Secure routing in wireless sensor networks: attacks and countermeasures," *Ad Hoc Networks*, vol. 1, no. 2-3, pp. 293–315, 2003.

[26] N. Kandasamy, S. Abdelwahed, and J. P. Hayes, "Self-optimization in computer systems via on-line control: Application to power management," *Autonomic Computing, International Conference on*, vol. 0, pp. 54–61, 2004.

[27] S. Albers, S. Eilts, E. Even-Dar, Y. Mansour, and L. Roditty, "On nash equilibria for a network creation game," in *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*. New York, NY, USA: ACM, 2006, pp. 89–98.

[28] A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and S. Shenker, "On a network creation game," in *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*. New York, NY, USA: ACM, 2003, pp. 347–351.

[29] J. Cohen, A. Dasgupta, S. Ghosh, and S. Tixeuil, "An exercise in selfish stabilization," *TAAS*, vol. 3, no. 4, 2008.

[30] S. Wicker, "On the complexity of distributed self-configuration in wireless networks," *Journal of Telecommunication Systems*, pp. 33–59, 2003.

[31] H. Zhang and A. Arora, "Gs3: scalable self-configuration and self-healing in wireless networks," in *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*. New York, NY, USA: ACM, 2002, pp. 58–67.

[32] J. Wildstrom, P. Stone, E. Witchel, R. Mooney, and M. Dahlin, "Towards self-configuring hardware for distributed computer systems," *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pp. 241–249, June 2005.

[33] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like p2p systems scalable," in *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2003, pp. 407–418.

[34] A. J. McMurry, C. A. Gilbert, B. Y. Reis, H. C. Chueh, I. S. Kohane, and K. D. Mandl, "A self-scaling, distributed information architecture for public health, research, and clinical care," *Journal of the American Medical Informatics Association*, vol. 14, no. 4, pp. 527 – 533, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/B7CPS-4P1PR24-P/2/ec57af5b1e5e85edd7fe5007629b7276

[35] P. Felber and E. Biersack, "Self-scaling networks for content distribution," in *In Proc. of Self-**, 2004.

[36] W. Zhao and H. Schulzrinne, "Dotslash: A self-configuring and scalable rescue system for handling web hotspots effectively," in *in International Workshop on Web Caching and Content Distribution (WCW*, 2004, pp. 1–18.

[37] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in structured p2p systems," in *In Proc. of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[38] F. Kuhn, S. Schmid, and R. Wattenhofer, "A self-repairing peer-to-peer system resilient to dynamic adversarial churn," in *In Proc. of the 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2005.

[39] U. A. Acar, "Self-adjusting computation," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 2005, co-Chair-Guy Blelloch and Co-Chair-Robert Harper.

[40] A. Mowbray and R. Bronstein, "What kind of self-aware systems does the grid need," in *HP Laboratories*, 2005.

[41] M. Seltzer and C. Small, "Self-monitoring and self-adapting operating systems," in *HOTOS '97: Proceedings of the 6th Workshop on Hot Topics in Operating Systems (HotOS-VI)*. Washington, DC, USA: IEEE Computer Society, 1997, p. 124.

[42] S. A. Hofmeyr and S. Forrest, "Architecture for an artificial immune system," *Evolutionary Computation*, vol. 8, no. 4, pp. 443–473, 2000, pMID: 11130924. [Online]. Available: http://www.mitpressjournals.org/doi/abs/10.1162/106365600568257

[43] J. Kim and P. Bentley, "Towards artificial immune systems for network intrusion detection: An investigation of dynamic clonal selection," in *In Proceedings of the Congress on Evolutionary Computation*, 2002.

[44] D. Dasgupta and N. Attoh-okine, "Immunity-based systems: A survey," in *Proceeding of the IEEE International Conference on Systems, Man and Cybernetics*, 1997, pp. 369–374.

[45] S. Ghosh and A. Gupta, "An exercise in fault-containment: Self-stabilizing leader election," *Information Processing Letters*, vol. 59, no. 5, pp. 281 – 288, 1996. [Online]. Available: http://www.sciencedirect.com/science/article/B6V0F-3VVCM98-P/2/261b7403536c3fc209adde503db2c475

[46] C. Dabrowski and K. Mills, "Understanding self-healing in service-discovery systems," in *WOSS '02: Proceedings of the first workshop on Self-healing systems*. New York, NY, USA: ACM, 2002, pp. 15–20.

[47] C. Dabrowski, K. Mills, and J. Elder, "Understanding consistency maintenance in service discovery architectures during communication failure," in *WOSP '02: Proceedings of the 3rd international workshop*

on Software and performance. New York, NY, USA: ACM, 2002, pp. 168–178.