SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 0000; **00**:1–25
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe

A Survey of Self-Healing Systems Frameworks

Chris Schneider, Adam Barker, Simon Dobson
{chris.schneider, adam.barker, simon.dobson}@st-andrews.ac.uk

SUMMARY

Rising complexity within muti-tier computing architectures remains an open problem. As complexity increases so do the costs associated with operating and maintaining systems within these environments. One approach for addressing these problems is to build *self-healing* systems (*i.e.*, frameworks) that can autonomously detect and recover from faulty states. Self-healing systems often combine machine learning techniques with closed control loops to reduce the number of situations requiring human intervention. This is particularly useful in situations where human involvement is both costly to develop, and a source of potential faults. Therefore, a survey of self-healing frameworks and methodologies in multi-tier architectures is provided to the reader. Uniquely, this study combines an overview of the state of the art with a comparative analysis of the computing environment, degree of behavioural autonomy, and organisational requirements of these approaches. Highlighting these aspects provides for an understanding of the different situational benefits of these self-healing systems. We conclude with a discussion of potential and current research directions within this field.

Received ...

1. INTRODUCTION

The increasing complexity of modern computing environments is continuing to produce challenges in reliable and efficient systems management. As infrastructures share multifaceted physical and virtual requirements, the static capabilities of human administration are showing decreases in their relative effectiveness. This is increasing the costs of systems management, whilst simultaneously introducing potential problems–such as issues with change management, and simple human error. This problem is particularly evident in multi-tier architectures where services are comprised of several sets of systems with differing responsibilities.

Self-healing systems frameworks are emerging as a useful approach in addressing the rising complexity requirements of systems management. These frameworks attempt to classify and analyse sensory data to autonomously detect and mitigate faults. This in turn reduces the need for systems to interface with human administrators, lowering operational costs and, ideally, improving upon existing mitigation techniques. Self-healing methodologies are often realised through the use of machine learning techniques or other aspects in artificial intelligence. They have have been described via architectural differences [1], network behaviours [2], research areas [3], and even biological likenesses [4]. These surveys have produced a broad spectrum of knowledge and highlighted notable advances and exigencies within the field. However, the effectiveness of these solutions, and the commonalities shared between implementations, has yet to be fully explored.

Uniquely, this survey contrasts the type of environment or infrastructure in which self-healing frameworks operate, the learning methodologies these self-healing frameworks are expected to exhibit, and their manageability requirements or hierarchical needs. These

criteria were chosen as they are expected to be ubiquitous within any given self-healing systems implementation, and thus make for good markers of comparison. By providing an understanding of where specific methodologies are being leveraged and under what circumstances, this survey provides a groundwork for comparing the relative effectiveness of self-healing frameworks. Furthermore, analysing self-healing frameworks based on commonly shared properties allows for a comparative understanding of each methodology, their respective benefits, and their relative human costs. It explores the type of self-healing methodologies as related to their expected environment, and provides groundwork for exploring correlations between these factors. By contrasting behavioural properties with their expected implementation and level of autonomy, this paper provides a greater understanding of which techniques are being leveraged, and under what circumstances.

The remainder of this paper is structured as follows: The rest of section 1 briefly covers the vision and foundational research of self-healing systems. Section 2, provides an overview of existing self-healing systems methodologies – including frameworks, operating systems, and services. Section 3 outlines the results each of these systems have had, and novel approaches that have been produced. It is here that different approaches are contrasted, including management styles, architectures, and learning mechanisms. Section 4 concludes with a summary of findings and a brief discussion on future research.

1.1. Background

Many of the methodologies discussed in this paper refer to existing works in Autonomic Computing. Autonomic Computing covers a wide range of topics in self-managing systems– including *self-healing*, *self-optimisation*, *self-protection*, and *self-configuration* properties. Although a familiarity with this area of research is assumed, a summary of foundational literature is provided here for ease of reference.

This section discusses in brief the Autonomic Computing Initiative [5], and the goals and criteria of self-healing systems, as initially described by IBM and subsequent publications [6,7]. The illustration of these goals provides a way to narrow the problem space into addressable components and brings context to the methodologies presented in this survey.

1.1.1. Autonomic Computing The Autonomic Computing Initiative was proposed in 2001 to address growing complexity in systems management [5]. IBM proposed building software that could autonomously manage systems using a series of closed control loops and 'environmental knowledge'. These recursive software elements utilise a series of inferential steps to make real-time decisions that mitigate problems and automate palliative maintenance tasks. Over the last 10 years several advances have been made in realising these goals.

In 2003, IBM published two articles that built upon their initial proposal outlining the aforementioned four primary topics (or 'tenets') in Autonomic Computing, a general process for autonomic systems management [7], and a set of criteria that described behavioural 'levels' and generic goals of self-managing systems [6]. The process for automating systems management tasks, often referred to as MAPE+K, outlined a recursive approach for continuously understanding and making changes to a system's state. By utilising 'knowledge' (K) about a system's environment, a designated software agent would: Monitor, Analyse, Plan, and Execute (MAPE) instructions to meet user-specified policies. Since its introduction MAPE+K has proven to be a central component in many self-managing systems implementations.

In order to understand the effectiveness of a given MAPE+K based process, behavioural levels were used to evaluate the implementations maturity. These levels ranged from basic to fully autonomic and were evaluated based on whether they could consolidate information, recommend an action, autonomously take an action, and finally interpret a user-specified policy to do all of the aforementioned behaviours. Importantly, this article recommended an evolutionary approach in reaching each of these stages [6]. Building self-managing systems that operate at different levels permits heterogeneous infrastructures, and allows for the gradual

Prepared using speauth.cls
DOI: 10.1002/spe

adoption of Autonomic Computing technology. This includes environments where existing systems may not be compatible with all of the autonomic computing levels.

To address the challenges proposed in these two articles, agent-based approaches for managing systems were introduced [8]. Utilising aspects in artificial intelligence, this work was based on an earlier text discussing reflex, goal, and utility agents [9]. Simply stated, reflex agents use if-then rules to map actions to a specified state. In practice, this approach is used once some criteria is met to execute a pre-specified set of instructions. Goal and utility-based agents attempt to exhibit rational decision making by autonomously determining what actions to take based on expected results. The primary difference between goal and utility based agents is that the former selects behaviours to attain a given objective, whilst the latter attempts to reach and optimise behaviours such that multiple objectives can be achieved at once. This was particularly useful if two goal policies contradict each other.

Using this approach as a foundation, IBM proposed that self-managing solutions leverage Action, Goal, and Utility 'policies'. These policies incorporated high-level objectives with systems tasks whilst allowing for resolution conflicts between enacted behaviours. However, the implementation of broad level policies have produced challenges in evaluating the effectiveness of self-managing systems. In the following year, a framework was introduced for evaluating the performance of a self-healing system called DTAC [10].

This framework unified the MAPE+K control loop with industry requirements, and provided metrics for evaluating self-managing systems. It described and quantified properties such as stability, accuracy, settling times, and efficiency. By using these properties it became possible to conduct behavioural evaluations based on a system's environmental knowledge, and historical performance data. The evaluation of this information led to a more expansive approach that discussed general research challenges in self-managing systems, and a variety of scientific advances in self-managing systems [11].

Specifically, self-managing systems solutions were divided into elements, systems, and interfaces, and standards definitions and requirements for each of these components was proposed. This helped to unify the mission of Autonomic Computing with practical implementations by illustrating examples of where action, goal, and utility policy approaches had been implemented [12–17].

Notably, Kephart argued that the division of self-managing systems into autonomic elements would allow for easier adoption of legacy systems. By incorporating existing services with an autonomic interface, legacy architectures could be made to adopt self-managing strategies. Once a legacy system had an access point for autonomic communications, self-managing systems could exert some influence over the existing infrastructure. Indeed the notion of inter-element communication was arguably the central thesis of this paper: "The main new research challenge introduced by the autonomic computing initiative is to achieve effective inter-operation among autonomic elements" [11]. This challenge continues to be an open problem in self-managing systems.

The establishment of core tenets, the MAPE+K process, evaluation methodologies, the autonomic maturity model, and action, goal, and utility policies, created a foundation for stand-alone contributions in self-managing systems [18]. The ideas have also migrated into the domain of communications [2], and have found extensive use in networks and embedded systems.

1.1.2. Self-Healing Systems The definition of a self-healing system has evolved over the past 10 years. Initially, self-healing systems were described as being able to detect and recover from faults, without the need for human interaction [7]. Although no system has been able to operate with complete autonomy, several advances have been made towards the realisation of self-managing computing environments. This has come as evolutionary processes are being routinely involved with the development and maintenance of autonomic computing frameworks; a prediction made by IBM in 2001 [6].

Prepared using speauth.cls
DOI: 10.1002/spe

As large-scale computing infrastructures have become more complex, existing methods for operating and maintaining systems have become less effective. The use of skilled engineers to apply monitoring techniques that search for faults, engage in root-cause analysis, and execute appropriate recovery strategies remains the *de facto* standard of most professional organisations. Most of these monitoring techniques utilise some form of behavioural test to indicate when a fault is present. Self-healing systems seek to automate these processes. If a service fails, rather than requiring an engineer to intervene, a self-healing system would autonomously diagnose the fault and then execute a recovery strategy.

The potential exists for systems to diagnose issues more quickly than their human counterparts. If realised, the result would mean less time spent administering systems, reductions in operational costs, and decreases in lost revenue. Consequently, the definition of self-healing systems has been expanded to include behavioural aspects that are commonly evaluated in modern computing infrastructures. It is no longer acceptable for a system to simply detect and recover from faults – it must do so transparently, and within certain criteria.

The definition of criteria can vary as infrastructures have different requirements, however they often include aspects such as availability, reliability, and stability. Availability is defined as whether or not a system is accessible, whilst reliability is a percentage of time that a system operates as expected. Stability is defined as how fast a system can mitigate faults and return to its original state.

The integration of behavioural aspects has helped to unify business needs with IBM's original vision of self-healing systems. By adopting partially self-healing systems into traditional infrastructures, an evolution of techniques and new self-healing systems methodologies have emerged. However, not all self-healing methodologies are compatible with existing infrastructures and the maturity of many of these techniques has not been fully realised. As self-healing systems methodologies become more mature, less human supervision should be required.

One approach to understanding maturity in a self-healing environment is by evaluating systems state *via* behavioural properties [8, 19, 20]. By understanding when and how long a systems executes self-healing behaviours, it becomes possible to evaluate self-healing methodologies against existing implementations. Understanding the effectiveness of self-healing systems methodologies against current approaches provides a practical baseline for understanding the advancement of self-healing systems outside of the Autonomic Maturity Model. However, to achieve this goal a set of criteria must first be defined that is present in a majority of self-healing systems methodologies that are to be evaluated both now and in the future. It is for this reason that computing environment, learning methodology, and management style were selected for comparison.

1.1.3. Assumptions and Definitions Research in self-managing systems contains a wide variety of terminology and definitions. Although some definitions are considered to be standardised [19], many publications utilise terms in a modified fashion to meet the stricter requirements of their purposes. As vocabulary usage diverges, multiple connotative and denotative understandings are formed by readers. The following paragraph describes comparative terms utilised within this survey. This is done with the intent of helping the reader to establish a referential understanding. Likewise, many of the terms and purposes of the Autonomic Computing Initiative share properties and, to a lesser extent, charters. For example, in order for a system to self-heal, often it must elect to change its configuration in some way. This is sometimes confused with the concept of self-configuration, which deals primarily with self-provisioning–the ability for a system to continuously evaluate and integrate itself without human interaction within a given computing environment. Security aspects share similar traits under the self-protection tenet. This is a topic that is later addressed in sections 3 and 4.

For the purposes of this survey, *self-healing systems* are assumed to be real or virtualised servers that exist within a grid, cloud, or standard large-scale computing infrastructure. Although there are numerous physical components that make up large-scale computing

DOI: 10.1002/spe

environments, the scope of this survey primarily emphasises servers as central points of focus. It is important to note that exigencies can exist outside of this scope, which the server is still responsible for identifying. Examples of this include network connectivity diagnosis, and being able to determine resource availability, such as a remote API.

Although there are numerous interpretations as to what constitutes a computing environment the majority of terms applied within this survey are taken from a single source [21]. Specifically, *grids* are defined as voluntary collections of physical systems that share resources, and typically consist of multiple, heterogenous configurations . In these environments *churn* – the rate at which membership changes – is expected to be high, and systems are expected to be managed in an *ad hoc* fashion. This can translate to computing environments that do not require professional services to operate, such as those housed in a data-centre. Conversely, *clouds* are collections of either real or virtualised computing devices that are centrally managed, and controlled by a single entity. Devices that exist as part of a cloud are more likely to be configured identically, housed in a data-centre, and operated by a large professional or academic staff. Membership of devices in these environments are expected to have high-availably constraints, and be relatively static in terms of their rate of churn. A third category of *standard* (*i.e.*, "traditional") is reserved for established environments. Typically, these environments utilise multi-tiered architectures divided into front-end, middleware, and back-end sub-divisions that exist absent of virtualised components. This category is intended to represent the most common configurations for small, mid, sometimes large-size network-aware service applications.

It is assumed that computing environments may never be fully autonomous and that some problems will indefinitely require human interaction. Although this is not in keeping with the initial proposal, at some point it is perhaps unavoidable. For example, there are no known software solutions to mitigate non-redundant hardware failures. However, diagnosing and escalating such a situation to an administrator is still a desirable 'self-healing' behaviour. As such, systems that can operate to the edge of their limitations are still considered to be successfully self-healing.

Lastly, recovery is assumed to be a more difficult problem than detection. "The final stage, automated re-mediation of a problem once it has been localized, is perhaps the most difficult." [11] – however, the detection of faulty states is necessary before executing recovery strategies, *a fortiori*. This logic is the foundation upon which some aspects of framework maturity are gauged.

2. METHODOLOGIES

Self-healing frameworks leverage a diverse set of methodologies to autonomously detect and recover from faults. This section discusses and compares self-healing frameworks based on three primary aspects: Management style, computing environment, and learning methodologies. These properties are often interrelated and exist within each self-healing systems framework. As such, they provide a way to compare the relative utility of each approach and establish a consensus for comparison.

The remaining subsections are organised as follows: Section 2.1 contrasts top-down and bottom-up management styles that utilise self-healing frameworks. Section 2.2 discusses computing environments, and contrasts different self-healing behaviours commonly found within grids, clouds, and standard infrastructures. Lastly, section 2.3 provides an overview of learning methodologies used to autonomously detect and recover from faults. A distinction is made between supervised, semi-supervised, and unsupervised methodologies, and in what environments they are most commonly implemented.

Prepared using speauth.cls
DOI: 10.1002/spe

2.1. Management Styles

Managing complexity in computing environments has led to an abundance of architectural and systems management techniques. This survey focuses on two specific styles: Top-down, and bottom-up. Top-down approaches organise systems into hierarchies by leveraging authoritative nodes. These nodes control, propagate, and validate the behaviours of subordinate child-nodes within the computing environment. Conversely, bottom-up methodologies operate in an *ad hoc* fashion, leveraging neighbouring devices to make or suggest changes to configuration state.

Each approach divides computing environments into smaller, more management sub-components. The division of systems into sub-components helps to address the natural complexity that arises when managing multiple nodes. This includes aspects from change management, divisions in workflows, and enacting policies to automate systems tasks. Depending on the management style, however, the nature of the sub-components also changes to provide different advantages and disadvantages. It is often the case that management styles are selected based on computing environment specific needs – a subject discussed further in sub-section 2.2, "Computing Environments".

2.1.1. Top-Down Management Styles Top-down management styles are based on a hierarchical infrastructure for accepting and enacting policies on child systems [22]. This is often realised through the use of databases on parent-nodes to which subordinate nodes are instructed to periodically communicate with. By changing information within these databases, the collective behaviour of systems communicating with the parent can be altered. Thus, rather than requiring an administrator to access each system individually, top-down methodologies can execute instructions autonomously. Localising configuration changes to a single point has the benefit of reducing human error during implementation, and retaining a homogeneous configuration baseline within a computing environment. Top-down management styles are useful in ensuring predictable recovery behaviour, and are widely utilised [23–26]. Conversely, centralised infrastructures often require extensive pre-configuration and training before they can exhibit self-healing behaviour.

Rainbow [23, 27] is a self-healing framework that leverages a centralised, top-down management style. Utilising a set of 'system concerns', child-nodes are divided into clusters based on a similar set of expected behaviours. These properties are collectively described as system 'roles', and are maintained by a single Rainbow instance. An administrator then provides a set of constraints and recovery plans, which the service uses to evaluate systems behaviour. Evaluations occurred using a three-tiered, abstract architectural model that autonomously categorises systems behaviours. If a fault is detected, the server's configuration is then altered using recovery plans associated with the system's synthesised role, and respective constraint model.

Rainbow's approach to dynamic systems evaluation, and its centralised methodologies, are arguably foundational by many subsequent approaches. This in includes the ability to utilise centrally located recovery plans that are associated with the identification of specific faults [26], and the use of recovery plans that have been created by systems administrators at run-time. Once enacted these result are stored for later use within a centralised database - a technique sometimes referred to as *case-based reasoning* (CBR).

MARKS+ [24] leverages a comparable approach to Rainbow by using what it refers to as *healing manager* nodes to select and implement pre-defined recovery plans. The recovery plans are again evaluated based on a constraint model, but also include a *service availability mapping*. This mapping, combined with a collection of behavioural unit tests, provides context to the evaluation of the constraint model. Systems determined to be in a faulty state are removed from service until a 'good' behavioural context can be re-established via the the return of the system to a previously known working configuration or 'state'. For MARKS+, Healing managers facilitate these behaviours by acting as a centralised orchestration service. This is similar to Rainbow in that both approaches use an architectural perspective to facilitate resource discovery and recovery behaviours.

Prepared using speauth.cls
DOI: 10.1002/spe

The use of 'behavioural skeletons' is another perspective on understanding systems activities in top-down infrastructures [25]. Behavioural skeletons are similar to models and consist of an abstract collection of patterns that can be used to evaluate a system's behavioural properties. When combined with a set of constraints, or 'contract' [25], top-down methodologies can attribute context to systems behaviours without depending on pre-defined roles. This has the advantage of not requiring developers to commit to pre-approved configuration states. Similarly, skeletons and contracts can be used to provision a specific subset of information to child-nodes–such as configuration data or faults. Whilst the child-nodes retain this information locally, a reduction in the need for 'call-backs' to management services remains present. This allows the systems to work more independently and utilise external resources only when required.

The use of locally provisioned self-healing logic is similar to the two previous approaches in that it leverages rule-based action policies to decide on recovery strategies. However, it differs in how systems are allowed to interact, and provides an approach for leveraging more autonomous behaviours. The latter is an artefact that has been extended in subsequent publications [28]. Rather than using a series of contracts, SASSY handles infrastructure management through the use of dynamic model generation called *Service Activity Schemas* (SAS's). By aggregating these SAS's, an architecture can be dynamically mapped into subgraphs. This allows not only the systems to be modeled individually, but the service architecture itself to be evaluated in a dynamic fashion. Consequently, using this approach affords greater flexibility in compartmentalising faults within the environment than other top-down frameworks, and provides more distributed management of resources than stand-alone top-down service discovery methodologies.

MOSES [29] takes a similar approach to SASSY in that management of the service architecture itself is leveraged in detecting and recovering faulty systems components. Like SASSY, MOSES dynamically models the architecture in which it is operating. By using a *position manager* this framework determines if the service's detected resources can be combined until a usable model. Once completed, an *adaptation manager* addresses any faults or quality of service issues encountered by using a series of vectors abstracted from the services model. This information is then abstracted into an ordered list of service priorities that can then be used to direct or redirect service flows–even in the presence of conflicts.

The sampled centralised management styles exhibit similar self-healing logic when recovering from faults. In most instances, the use of behavioural testing is implemented with a contextual reference – such as a constraint or systems model. This is further expanded upon by user validation (in the case of supervised methodologies), or by using predictive measures to discretely synthesise recovery solutions. Furthermore, the use of these techniques in a centralised orchestration service affords many benefits–including the ability to retain control of the infrastructure from singular management points, and being able to leverage re-use of recovery strategies [23–28]. This is in contrast to systems that inherent or infer self-healing behaviours; discussed further in the following section 2.1.2. The learning methodologies for each management style is discussed further in section 2.3.

2.1.2. Bottom-up Management Styles Bottom-up management styles emphasise *ad hoc* interaction between systems. Systems within these environments typically infer self-healing behaviours based on independent sampling, either of the service infrastructure at large or neighbouring systems, and exhibit a greater degree of administrative autonomy. They represent a direct alternative to approaches that leverage centralised management, and typically demonstrate more exploratory behaviours. This type of systems management can require less initial configuration than centrally managed approaches, but at the cost of predictability, and individualised control.

Although *ad hoc* systems management comes in a variety of forms, this survey focuses on three distinct approaches: System-to-system [30–32], localised healing [33–38], and those that utilise atomic interfaces to synthesise virtual resources [39]. System-to-system frameworks

Prepared using speauth.cls
DOI: 10.1002/spe

are capable of making changes by sampling from or delegating to neighbouring nodes. This is contrasted by localised healing frameworks which avoid administering other devices, and use information obtained from neighbouring systems to self-elect behavioural modifications. Atomic frameworks exist as a hybrid of these two approaches by exposing their resources in an non-holistic, read-only fashion. They can either self-elect or suggest changes to external devices, or directly access external resources as if they were locally present.

In a system-to-system infrastructure authoritative actions are delegated dynamically through the analysis of environmental knowledge. Examples include frameworks that observe both the performance and service availability of neighbouring devices [31, 33, 34]. In the case of Embryo-ware, a set of administrator supplied configurations provides each system with the ability to autonomously adapt from a 'totipotent' state into one of several pre-specified roles. This behaviour is initiated based on each systems local perception of the over-all performance and relative needs of the service infrastructure. If a service has reached a capacity threshold for its front-end web-services, for example, and the system has a totipotent configuration, it can dynamically adopt a web-role and join the front-end pool to increase capacity. Once the service has been evaluated as no longer needing additional front-end resources it then reverts back to its neutral state.

By treating systems as modular components, Embryo-ware addresses a key problem present in *ad hoc* infrastructures–drift in baseline systems configuration. As systems continue to operate they naturally encounter events that create unique systems configurations and states. This can create scenarios where systems are difficult to predict and can reduce the effectiveness of existing self-healing behaviours. By resetting the local system's state to pre-defined known-working configurations, divergence in systems operations is dramatically reduced. This allows for techniques that depend on assumptions related to the systems behaviour to continue to be effective well after initial deployment. It also allows for servers to be treated as dynamic resources within the service architecture and to transparently address the workloads associated with predefined groups of individual service components.

Transparently updating service components is an approach also used by OSIRIS-SR – an extension of OSIRIS [30] and Chord [40]. However, unlike Embryo-ware, OSIRIS-SR uses a transitive management service to create 'supervisor nodes' that facilitate self-healing behaviours. These nodes leverage a distributed hash-table to establish service parity, and to facilitate work delegation of a given resource. This allows service availability to be preserved even in infrastructures with high rates of churn, and for systems to orchestrate service flows whilst addressing faults–all without a centralised infrastructure.

Rather than shifting a system's role or instantiating a supervisory service, systems within the computing environment may also have the ability to assign work directly to each other [32]. VieCure utilises an activity management service to understand local and remote service state. Like Embryo-ware, this framework is installed locally on each system, and configured by a set of policies that guide self-healing behaviours. The policies combine 'interaction patterns' and constraints into a 'behaviour registry' – a dictionary of recognisable systems states that are used to indicate when self-healing behaviours are required. If a constraint violation occurs, the system can either choose to heal or delegate work to a neighbouring node. VieCure, OSIRIS-SR, and Embryo-ware operate holistically. The expression of their self-healing logic is based on the evaluation of their respective computing environments as a whole. However, not all *ad hoc* frameworks operate in this fashion.

Atomistic perspectives, such as the General Purpose Autonomic Computing framework (GPAC) [39], view and evaluate systems resources as individual components based on 'resource definition policies' that are supplied by an administrator. The benefit of atomistic components is that they are usable remotely by other systems. To accomplish this, GPAC first builds a model of local systems operations by utilising a four stage control loop similar to MAPE+K. The model is populated by querying either a remote or locally running service that discovers resources. Discovered resources are then integrated with the model information by a policy

DOI: 10.1002/spe

engine to create the aforementioned 'resource definition policy'. This allows resources to be directly accessed, regardless of physical location.

Atomising computing environments represents a unique approach to the delegation of work. GPAC coordinates resources in a transparent fashion to mitigate faults rather than assigning tasks directly. Sharing resources leads to a natural integration between systems, and illustrates a unique approach for mitigating faults remotely. This comes with the caveat that systems must be able to accept changes to their configurations from neighbouring nodes. In some computing environments this property is undesirable. For these cases, localised healing strategies are preferred over other approaches.

Localised healing frameworks avoid directly administering other devices. Instead, each framework instance is exclusively responsible for its local system's health, resources, and configuration state. This includes determining when issues are caused by local or external factors. Localised faults are mitigated in a similar fashion as other frameworks. A set of constraints and policies are provided by administrators which the systems use to detect and recovery from faults. However, faults determined to be external to the system are addressed much differently. External faults are either ignored, referred to another system, or, if possible, mitigated locally. These approaches are not designed to address the source of the error, but to maximise the availability and performance constraints of the computing environment–often within predefined guidelines.

For example, lowering the fidelity of content being served by front-end web-servers is one way to meet to performance constraints [41]. If a server cannot deliver content at the rate expected – *e.g.*, due to too many concurrent connections – it can elect to reduce the volume of data sent for subsequent data requests. This approach does not directly address the state of other systems, but instead focuses those issues that can be resolved locally. Frameworks that focus on localised self-healing techniques often use 'roles' to facilitate the re-use of self-healing logic and to meet constraints [36, 41]. This is particularly useful in self-healing systems that operate within a single tier of a computing environment.

WS-DIAMOND [36] is a localised healing framework specifically developed for front-end web-services. It uses two concurrent control loops to diagnose and recover from faults. The 'inner' control loop focuses on the mitigation of faults that prohibit basic systems operations. This can include resources that are critical to the system's role, and the state of services. The outer control loop addresses issues related to quality of service (QoS). If a system is not capable of performing within a set of constraints, an error is raised that the outer control loop attempts to mitigate. Other frameworks have mimicked the QoS approach, but sans use of multi-tiered control loops [42]. However, the basic approach used in these systems are essentially identical. Each failure instance is treated as a separate case from which to analyse the results of systems configuration tests. This allows faults to be categorised based on the systems role, and located using differential analysis of the systems configuration data.

Determining the source of an error is a non-trivial process. Systems configurations are complex sets of information, and often contain relationships between features and properties that are not easily classifiable. Dynamic systems modelling represents one approach for understanding correlations between faults and configuration state. In localised healing frameworks, such as Plato [37], UBL [35], and Shadows [38], these approaches have been used to categorise and compare the state of a system with historical information, such as systems configuration or performance data. This follows in the footsteps of other frameworks, such as Rainbow, that utilise architectural modeling techniques at system run-time.

Rainbow, Shadows, UBL and Plato all leverage a set of operating constraints and policies that are used to periodically evaluate service health. While the systems operate in a state that meets both their constraints and policy guidelines, they are considered to be in a 'known-good' state. In the case of Shadows, UBL and Plato, these states can be leveraged to build models of typical systems operations. If the system is not operating as expected, a comparison between previous and current behavioural models is executed. This allows for differentials to be

Prepared using speauth.cls
DOI: 10.1002/spe

discovered between systems behaviours, and recovery methodologies to be synthesised, rather than requiring them to be proscribed by administrators.

Recovery strategies for these frameworks operate with substantial difference. Plato utilises genetic algorithms to search for optimal systems configurations and enacts recovery methodology via reconfiguration. This is done by pre-computing configurations in a simulator and evaluating the results against a set of fitness criteria. The results of each configuration undergo a differential analysis that examines the health and performance of various systems models.

Shadows uses a model repository to determine a recovery strategy. The repository is populated via two mechanisms - a code extraction methodology, and a CBR-based approach similar to those described by Carzaniga [43], Shang [23], and Hassan [44]. However, rather than requiring administrators to update the repository manually, Shadows automatically builds role-based recovery solutions without human intervention. This is accomplished by using a combination of statistical and predictive modelling to synthesise configurations and evaluate potential solutions to detected faults. Once a solution has been found it can be validated and shared throughout the environment where behaviours are determined to be similar. This unique use of case-based reasoning allows the framework to leverage the advantages of *ad hoc* systems management without depending on centralised infrastructure or human administrator to approve new recovery methodologies. By removing the supervision requirement of this CBR approach anomalies can be detected that were not previously known.

UBL uses a different approach by leveraging a self-organising map (SOM) to train the system to understand failure, pre-failure, and working states. This technique allows systems to build their own recovery solutions at run-time by leveraging a vector based approach for aggregating systems configuration and performance data. Once the information has been obtained it is then classified and subsequently analysed (*i.e.*, 'mapped'). Faults are then inferred through a differential analysis of changes in both behaviour and configuration state of the system in question.

The management style of a self-healing framework is often related to its environment. In the case of *ad hoc* systems administration, the behaviours exhibited are inherently less predictable than those that leverage centralised methodologies. This comes as a caveat of allowing systems the ability to independently explore solutions outwith those having been directly supplied. Specifically, systems that leverage a bottom-up management style appear to be more prone to use semi-supervised and unsupervised learning techniques to achieve dynamic recovery solutions. While this approach is by definition more autonomous, it does not necessarily mean that it is more usable. Some environments may be required to use only proscribed recovery solutions to address specific service aspects–such as risk management or high availability requirements. In such cases solutions such as Embryo-ware may be better suited than UBL, Plato, or Shadows.

Choosing a management style for a self-healing framework is multi-faceted problem and can depend on a number of extraneous factors–such as the environment in which the system is intended to operate, acceptable levels of downtime, or expected resource utilisation. The following diagrams illustrate the frequency in which a specific management styles are applied based on computing environment, and a trend in the usage of top–down management styles. At present, these centralised approaches appear to be more commonly leveraged within environments where systems membership is stable–a topic further discussed in section 2.3, (see Figure 5). Likewise, the number of self-healing frameworks utilising this approach is continuing to increase (see Figure 3). Further to these topics, the attributes between environments and learning methodologies, and associated self-healing behaviours, are discussed further in sections 2.2, and 2.3, respectively.

DOI: 10.1002/spe

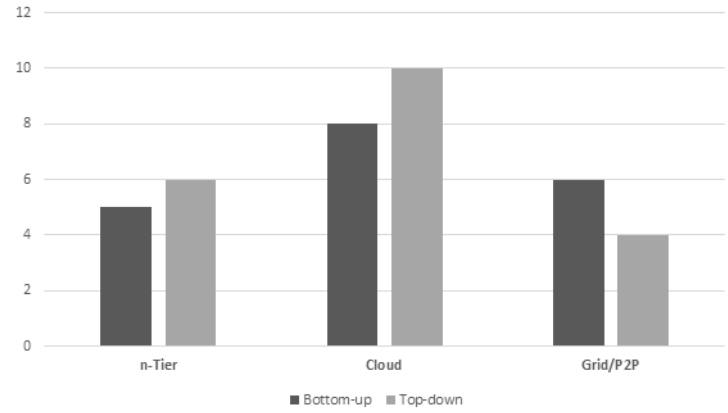

Figure 1. Management Styles versus Computing Environments

Top-down management styles are the primary choice when a computing environment is owned by a single entity. However, when membership is shared by multiple third parties, a shift can be seen in how the environment is managed. In the latter case, systems are more likely to be managed in an ad–hoc fashion.

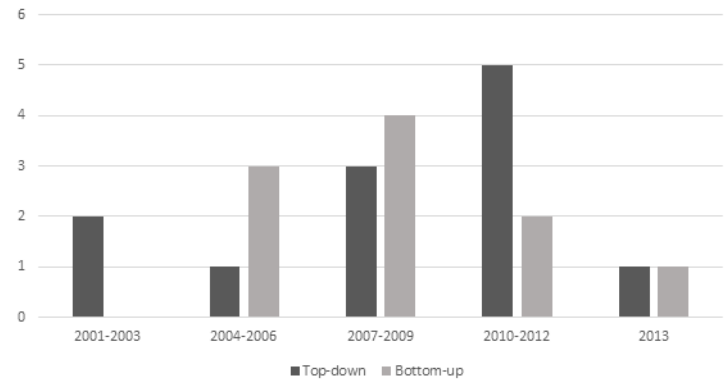

Figure 2. Management Styles in Self-Healing Frameworks by Year of Introduction

The number of self-healing frameworks that leverage top-down methodologies is continuing to increase. This comes despite the emergence of hybrid computing environments, and advances in predicting systems membership.

2.2. Computing Environments

Computing environments are a collection of resources used to manage and facilitate a given set of systems. Depending on the needs of the systems, computing environments can have different infrastructures and assets. This survey focuses on three types of infrastructures: Standard, virtualised, and *ad hoc*. Each infrastructure type represents differences in how self-healing frameworks access, categorise, and utilise resources. These differences can have profound impacts on the approaches used by self-healing frameworks and their respective goals.

Standard infrastructures are typically comprised of three categories (*i.e.*, 'roles') when discussing systems responsibilities: Front-end, middleware, and back-end. Front-end systems are responsible for establishing and maintaining connections to clients, middleware provides facilitating services such as encapsulation, transport, or orchestration, and back-end systems

Prepared using speauth.cls
DOI: 10.1002/spe

are responsible for the provisioning, storage, and parsing of information. This division of responsibility is the basis for establishing reusable code in many self-healing frameworks – regardless of computing environment – and promotes scalability by organising systems into reusable, interchangeable components. This allows for extensibility in behaviours and interchangeability of failed devices.

Virtualised infrastructures emulate physical assets by using multi-system resource management techniques. Instead of building a physical machine with a specific role, resources are dynamically allocated from a collection of physical machines to build virtualised 'instances'. These instances operate in the same fashion as physical systems. However, as the hardware itself is a software manifestation, 'physical' changes can occur more rapidly and in a more autonomous fashion than standard infrastructures. In addition to rapid reconfiguration, virtualised infrastructures handle change control exceptionally well. This is primarily due to the use of systems clones (*i.e.*, *images*) when instantiating new instances. Images allow for quick replacement, re-provisioning of faulty systems, and fast comparisons between systems' configurations. These properties make virtualised infrastructures heavily leveraged in cloud computing environments.

Standard and virtualised infrastructures share several key properties. They are often owned or operated by a single entity, have low rates of churn, and typically leverage centralised management styles (see section 2.1.1.). These aspects are vital in meeting established minimum operational requirements such as availability, reliability, and performance expectations – sometimes referred to as *service-level agreements* (SLAs). However, there are computing environments that do not share or require these properties. In these cases self-healing frameworks leverage *ad hoc* infrastructures.

Ad hoc infrastructures are unique from other approaches in that systems membership is voluntary. This property is related to *ad hoc* management styles, which enable systems to self-elect behaviours (see section 2.1.2), but is different in that it refers to the association a system has to a specific environment. The ability for systems to join and leave an infrastructure has advantages in that they are better suited for some distributed computing uses, and can potentially operate at lower costs. The transient nature of *ad hoc* infrastructures pose unique challenges for self-healing frameworks. Notable examples include higher rates of churn [21], issues with reputation [45], security [46, 47], multi-party administration [48, 49], and a lack of baseline configurations between systems [50], amongst others. In general, each framework instance must act as an authoritative point and evaluate its infrastructure independently. This is sometimes referred to as *self-elected behaviour*.

Computing environments are sometimes comprised of multiple infrastructure types. Some environments, for example, may have systems that are capable of interacting with each other in an *ad hoc* fashion, but may also depend on a centralised service model [33, 38]. In most cases self-healing frameworks have been developed to meet specific needs within a single tier of an infrastructure – such as a front-end web-service [26, 36, 41, 42]. Nearly all self-healing frameworks that are designed to operate within a single tier are capable of being implemented in a virtual infrastructure. However, not all self-healing frameworks are restricted to one area of responsibility [23,33]. The most common tier-specific self-healing frameworks are those that focus on front-end systems [26,36,41,43].

Systems that approach front-end web-services utilise a variety of approaches, including multi-tiered control loops [36], fidelity reduction [41], and behavioural modeling [26], amongst others. These self-healing frameworks are easier to develop, and can promote an intermediate stage for adapting existing infrastructures towards stronger administrative automation. Each system in a standard infrastructure must be maintained individually. This has several notable consequences including increased provisioning times, the potential for inconsistencies in configuration and implementation, and a natural deviation in configuration baselines overtime. These problems have been partially addressed by self-healing frameworks through the use of CBR and centralised management styles [24–26, 32, 38]. Centralised approaches leverage an often human supplied correlation between root causes of faults and their respective

Prepared using speauth.cls
DOI: 10.1002/spe

recovery strategies. As the expected outcome is based upon assumptions of previous state, these approaches can become less effective as configurations diverge. As changes occur within separate infrastructures outside of the control of the framework, this problem becomes more complex.

Virtual infrastructures help to address baseline configuration deviations, dynamically provision new resources, minimise the impact of external infrastructure changes, and improve deployment and recovery times. The majority of these advantages stem from the use of images which, as previously mentioned, help to maintain standard configurations between systems. Virtual infrastructures also come with several major disadvantages, the largest being cost to operate, proprietary standards for larger implementations, and challenges for physical expansion. However, virtual infrastructures provide useful properties to frameworks that use tier-based and search-space approaches to resolving faults [37, 38].

Frameworks that leverage search space methodologies require one of two conditions to occur before executing self-healing behaviours: Either an acceptable solution must be converged upon, or all available resources are exhausted. In the latter case, the framework picks the best solution found [35, 37, 51]. Standard environments limit the availability of resources to the physical capabilities of the system upon which the framework is instantiated. Virtual environments provide an advantage by allocating resources beyond the immediate instance. This promotes the self-healing behaviours from break-fix objectives to optimisation strategies (e.g. [29, 37, 52]).

In addition to optimisation, the dynamic allocation of resources is useful for promoting stabilisation in computing environments. There are several self-healing approaches that explore stabilisation in standard environments including dynamic role-adoption [31, 33], resource discovery [23, 27], resource policies, atomisation [39], and reduction in content fidelity [41]. In some cases, virtual infrastructures demonstrate comparable advantages by using instancing. Embryo-ware's ability to use 'totipotent' systems to shift to and from needed roles is comparable to virtual infrastructures ability to dynamically spawn new server instances– assuming an image exists for the needed role and a feedback mechanism is actively monitoring service state. Both approaches represent a way to preserve QoS in an environment, and minimise the need to reduce content fidelity.

Virtualisation universally addresses a major advantage of Embryo-ware: The ability to use a single subset of resources to address multiple roles within a service or computing environment. This concept is difficult to implement in standard, multi-tier infrastructures. Systems that are organised into tiers have external considerations when communicating with other devices. This includes networking configurations, security measures, and other exigencies of a practical nature that are outwith the control of the framework. With standard and virtualised infrastructures, the barriers between tiers are often preserved. One approach for avoiding these issues is to treat the computing environment as a ring [30, 31, 40]. However, it is worth noting this effectively converts the standard tier-based environment into an *ad hoc* infrastructure.

Ad hoc infrastructures avoid many of the organisational requirements of standard and virtual infrastructures. In ring-based approaches, systems are often required to accept a centralised point of management, and be operated within a confined set of conditions, such as a specific configuration or role. In *ad hoc* infrastructures systems are defined by their ability to carry divergent configurations and self-elect behavioural changes and states. These properties help to mitigate security issues, high rates of churn, diversity in systems configuration, and multi-party administration. Although this survey contains no frameworks that have been explicitly designed for entirely *ad hoc* infrastructures, several approaches expect and utilise *ad hoc* self-healing behaviours [31, 33, 35, 37].

These behaviours range from self-electing systems roles [31, 33], to aggregating resources between systems [39]. In the former case, each system evaluates the state of the service independently by querying neighbouring devices. If a system chooses to adopt a new role or configuration, it is ultimately centrally managed as the pre-specified roles must be provided

DOI: 10.1002/spe

to each individual system before they can operate. However, the collective behaviour of each individual system evaluating the service demonstrates an emergent approach to managing the infrastructure health. Experiments with biologically-inspired paradigms [53] further suggests that gradients, fields and other "spatial" structures [54] can offer robust adaptation to local challenges and failures, and can act as a programming platform on which to construct complex applications.

Plato [37] and UBL [35] demonstrate this perspective by leveraging systems that can holistically self-evaluate service state using biologically inspired computational approaches. These approaches have distinct advantages in that systems need not be provided with pre-specified recovery strategies, and are specifically designed to exhibit self-adaptive processes through environmental analysis. This affords systems using these frameworks a better suitability towards environments where *ad hoc* management styles and infrastructures are in place–such as the ability for systems to integrate their own self-healing logic by using search-space methodologies. These approaches include, chiefly, genetic algorithms and artificial neural networks. However, search and probabilistic methodologies lack the stable, predictable nature of approaches that leverage periodic human intervention.

Computing environments and the services they are house are interrelated. Systems that have the ability to operate holistically require different supporting resources than those that operate in an atomistic fashion, or centralised fashion [20]. The self-adaptive behaviours of systems leveraging *ad hoc* methodologies appear to be more advanced with respect to self-autonomy than other approaches. This is evident in how systems are being implemented, and their ability to learn new solutions to recover from problems without human intervention. This claim is further supported by the abilities systems have in their learning capabilities as the holistic logic of evolutionary approaches show further advancement and costs compared to the dictionary style approaches of CBR and other centralised learning methodologies.

The following diagrams illustrate the relationship between computing environment and learning methodology. Although supervised approaches are still the most common of those surveyed, there is a higher propensity of unsupervised learning in ad hoc computing environments (see Figures 1 and 5). Additionally, self-healing systems research appears to be shifting towards cloud and virtualised technologies.

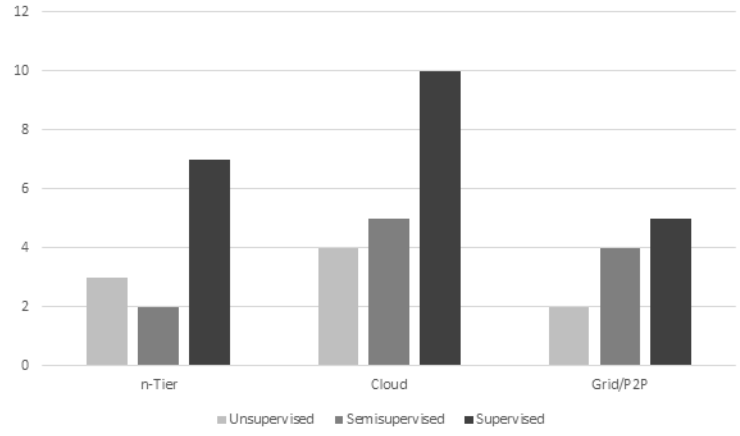

Figure 3. Computing Environments versus Learning Methodologies

The use of supervised learning remains the most common approach to self-healing systems frameworks for labelling training data. This is ubiquitous regardless the computing environment the framework is expected to operate in. However, these approaches seem to exhibit less autonomy in both detecting and recovering from faults (Section 2.3).

DOI: 10.1002/spe

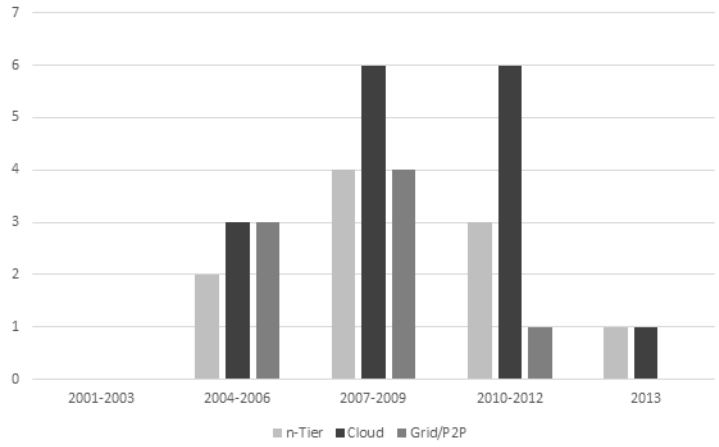

Figure 4. Computing Environments with Self-Healing Frameworks by Year of Introduction

Self–healing frameworks are being developed primarily for use in traditional, n–tier, or cloud computing environments. However, some hybrid solutions are also emerging that combine aspects that account for dynamic computing environments. This may represent a new research direction.

2.3. Learning Methodologies

Self-healing systems frameworks rely on heuristic algorithms to correct or change behaviour without human intervention. In order to maximise their effectiveness, learning methodologies have been developed that optimise when and how instructions are executed. These methodologies often utilise recursive, evolutionary, or close-control loop programming techniques to improve and evaluate behaviours. In each of these cases a feedback mechanism is used to determine both the validity and efficiency of a specific solution. The degree of required human interaction within a feedback mechanism is referred to as *supervision*.

Self-healing frameworks can be categorised as being fully supervised, semi-supervised, or unsupervised. Traditional definitions of these terms usually emphasise when or how a system classifies its learned behaviour – either manually, or dynamically–and whether or not data utilised by a specific algorithm has been labelled. As self-healing frameworks can contain multiple learning methodologies–each with varying degrees of supervision–cataloguing a framework's learning taxonomy into a single category is challenging. The primary intent of IBM's Autonomic Computing Initiative is to reduce the amount of required human interaction for a set of systems [7, 11, 18, 55]. Therefore, this survey focuses the most evident factors in evaluating and simplifying their classification: The frequency of required human interaction, and whether or not the framework can autonomously extend its self-healing behaviours. The latter component is to address changes in the state of a system, and its ability to respond dynamically to its environment. It is accepted that computing environments, in general, are not static entities. As needs and circumstances change so must systems be ready and able to adapt.

The most common approach to self-healing systems is to use a fully supervised methodology [23–26, 28–31, 42]. Supervised methodologies can require frequent interaction, and extend their self-healing behaviours only upon human intervention. This allows for validated, controlled configuration updates and provides the least amount of uncertainty in systems behaviours [56–58].

The most frequent implementations of supervised learning are *case-based reasoning* (CBR) methodologies. CBR typically utilises a database of prescribed recovery plans that are correlated to specific faults or events. When a system encounters an error it queries the database for recovery instructions. In those circumstances where recovery instructions have not been previously included, the framework will ask an administrator for a solution, or refer

Prepared using speauth.cls
DOI: 10.1002/spe

to a default set of actions. CBR approaches extend their behaviour by storing these additional solutions in their databases. Typically, the requirement of human supervision as a required part of the self-healing logic produces the natural caveat of only partial automation.

Rainbow takes supervised methodologies a step further by leveraging dynamic resource discovery with prescribed, role-based recovery logic. Using this approach computing environments are divided into recognisable components that can be used to dynamically build an architectural model of the service infrastructure. Using this model systems and services are categorised within a specified role or type, whilst the architectural model continues to choreograph service interaction and defines expected behaviours. These components are provided by developers before deployment. Once errors are detected, they can be mitigated using the architectural model to restore the service to a known working state or, if unsuccessful, an administrator can update the model at run-time.

WS-DIAMOND and GPAC take similar approaches to Rainbow in that a model is specified to which a system's performance is evaluated. However, rather than monitoring an entire service, each system is managed independently. As previously mentioned, WS-DIAMOND does this by instantiating two concurrent control loops to monitor and correct systems behaviours. Dividing the recovery logic into separate components allows the framework to prioritise and isolate recovery strategies. This is naturally conducive to goal and utility policy implementation within the specified model. A number of extensions to this framework have seen improvements to its detection and recovery logic* including the ability to monitor workflows, orchestrations, and choreographies.

GPAC contrasts this approach by utilising 'resource-definition policies' to autonomously discover and atomise systems components into network accessible objects. This non-holistic approach allows the framework to access resources on remote systems as if they were locally present. When combined with a model of the service, systems can act transparently to heal, and optimise the service architecture in a semi-supervised or potentially unsupervised fashion. These policies can also be used to tier service performance based on priority of behaviours or resources.

Performance tiering is a self-healing methodology used to divide systems and service health into levels [38, 41]. These levels in turn are used to understand QoS changes and instantiate behaviours that maximise the usage available resources. Arguably, the most direct approach to defining service levels is to use statically assigned resource constraints. Each level corresponds to a set of QoS metrics or fitness criteria that tells the system when to dynamically reduce content fidelity [41]. Primary developed for front-end web-services, static service tiering requires a human-supplied policy to determine when content fidelity can either be reduced or increased. In contrast, allowing policies to dynamically set thresholds for self-healing behaviours can have more autonomous results [29, 38, 42, 50, 59]. The Shadows framework, for example, uses a set of SLA's and utility policies to automatically generate behavioural expectations of a system. This allows the system to perform more in line with human-readable goals, such as cost, average service time, and other criteria instead of discrete metrics. It then combines this information with historical performance data to provide internal revalidation of recovery solutions. By using a time windowed mean expectations in behaviour can allow for elasticity versus pre-defined QoS metrics.

In a supervised framework the revalidation of a new set of expectations are normally completed by a member of technical staff. This occurs in a similar fashion as that being leveraged by Shadows: SLA's are compared against a system's overall performance and combined with historical data–such as application logs, configuration files, etc. The addition of correlating events with systems faults provides an advantage in contextually evaluating anomalies [32, 38]. By sampling the system at key intervals, faults can be associated with

*http://wsdiamond.di.unito.it/status.html

DOI: 10.1002/spe

specific changes and, ideally, their respective sources. This is useful for establishing a root-cause analysis and to map similar events with recovery solutions–sometimes referred t as Event driven monitoring. [35].

Event driven monitoring combines a complex set of sensor classification algorithms with run-time analysis techniques for isolating anomalies from normal or established patterns of behaviour. These approaches can range from the reactive use of simple exponential smoothing algorithms in a time series prediction [52], to pro-active prediction of states [60]. VieCure [32] is a CBR-style framework that leverages event detection in addition to direct analysis of metrics. Instead of directly mapping faults to recovery plans, VieCure looks for deviations in expected systems behaviours that can indicate when self-healing is needed. Events can constitute a series of incidents within a log, or a set of incidents that exhibit either a certain order or rate of occurrence. If an event is determined to coincide with a fault, then a recovery strategy is selected from a known set of working solutions. As expected, unknown events and faults require supervision in the same manner as other CBR frameworks.

Periodic interaction by administrators remains a caveat of supervised and semi-supervised self-healing frameworks. However, some frameworks have demonstrated an ability to dynamically elect self-healing behaviours without this requirement [33–35, 37, 59, 61]. Chiefly, these methodologies leverage biologically inspired approaches including genetic algorithms [37, 61], artificial neural networks [35], and totipotent behaviours [33, 34, 62, 63]. Each of these techniques have different properties that related to their suitability at solving particular tasks – from producing candidate solutions within a given search space [51] to the autonomous classification of sensory information [64]. These approaches range in degree of suitability based and how much risk and resource commitment a specific computing environment or service infrastructure is willing to accept.

Using a genetic algorithm, Plato can search for and mitigate faults based on correlations between behavioural properties and configuration data. This is a a framework that dynamically produces self-healing solutions based on a stochastic search methodology that comprises of multiple candidate solutions [37, 51, 61]. By comparing the operational SLAs and policies with the performance of the candidates individually, a degree of fitness can be ascertained from the candidate. Once the candidates have been evaluated, their individual features are analysed and correlated to produce new candidates. This occurs until either preset resource constraints are met, or an optimal solution is found per the associated fitness functions. In this instance, the utility functions in previous frameworks are analogous to the properties that are emphasised by the fitness functions in genetic algorithms. Each respective function provides the same base purpose: To translate and enact human-readable goals into systems behaviours. Examples of these goals include cost minimisation, application priorities, or performance traits.

This approach allows Plato to stochastically search for and build recovery strategies providing a critical advantage over other methodologies. Rather than requiring prescribed recovery solutions, either during development or run-time, Plato can autonomously produce viable self-healing solutions. However, there is no assurance that an acceptable systems configuration will always be found using this methodology, nor that it will be optimal. This is as expected [37], and inherent to the nature of existing search-space methodologies [51]. It is also computationally costly, and can produce behaviours that would not be anticipated. Thus, a high degree of risk can be associated with this approach.

Complimentary to using genetic algorithms, UBL operates by using historical configuration data to autonomously train a specific type of unsupervised artificial neural network called a self-organising map (SOM) [64]. Features in the historical configuration are converted into vectors which are then used as input for predicting behaviour, and feature state. This information helps to analyse the validity and impact on a system's behaviours when configuration changes occur. Once the SOM is trained, the system can then synthesise new, valid systems configurations by predicting which features are causal to specific faults. This approach leverages a smaller search space than the genetic algorithms used in Plato, and consequently presents less risk and potentially divergent systems behaviours. However, UBL displays some limitations in

Prepared using speauth.cls
DOI: 10.1002/spe

exploring new configurations and seems to produce a stronger likelihood of local minima in configuration synthesis. This is represented in the purposes of these two approaches being somewhat divergent: The ability to synthesise new systems configurations upon fault, and the prediction of failures within distributed infrastructures. As yet there exists no research comparing the effectiveness of combining these two approaches, however 'feature locality' continues to display positive results [50, 65] – a topic discussed further in section 3.

Separate from either of these approaches is the transparent management of resources within a service infrastructure via dynamic role or service adoption [31,33]. In each of these approaches systems use information about the general state of the service infrastructure to dynamically elect a localised reconfiguration. However, these approaches differ by allowing systems to dynamically adopt roles through self reconfiguration, in the case of Embryo-ware [33], and the self-instantiation of localised management services [31].

As previously mentioned, these systems are initially instantiated with a representation of the service, a set of roles, and an ability to query service state on remote systems. Using these three components the framework is then able to dynamically adopt new configurations or return to an original, neutral configuration based on service performance. Any device found to be without a base set of configuration data is automatically provisioned with the latest 'genome' via a replication agent. This provides a measure of self-configuration and provisioning; a process typically referred to as a separate challenge in Autonomic Computing [7, 11]. The adoption of new roles is facilitated via a differentiation agent that tracks and contextualises roles and expected functions. The differentiation agent must then self-elect a role-based on its independent understanding of the state of service.

This approach is contrasted by OSIRIS-SR, a framework that leverages Chord [40] to produce a *Safety Ring* to manage service infrastructures [31]. OSIRIS-SR operates by using supervisory systems roles to monitor and recover from failures in resource availability. These systems leverage meta-data to build an understanding of neighbouring systems behaviours, and then aggregate that information across multiple supervisory nodes. This is similar to Embryo-ware where only neighbouring nodes are monitored and influence the ability of those systems to adopt roles. What makes this approach unique is that any system can elect to become a supervisory node. This is useful for ensuring availability and reliable service management in infrastructures where systems membership can change without notice [66].

Both Embryo-ware and OSIRIS-SR can autonomously change the behaviours of its component systems at run-time, but only using information supplied at design time. There is no logic within either approach that will autonomously generate new recovery strategies or roles. However, this affords both frameworks the benefit of minimising risk for unexpected systems behaviours, and maximising available resource utilisation. In the case of OSIRIS-SR, this can be particularly useful if implementation occurs in mobile networks or other environments where systems membership is expected to be transient.

The following diagrams illustrate trends in the management styles associated with self-healing frameworks and their respective learning methodologies are outlined along with a diagram of their introduction by year. There are two key properties immediately evident within these figures: 1.) Self-healing frameworks research is driving towards solutions that utilise supervision, and 2.) The learning methodology leveraged by the self-healing framework appears to be linked to its management style. Additionally, if we extrapolate this information with that contained in Figure 1, it seems evident that the progression towards less supervision is being driven chiefly in ad hoc computing environments. However, due to the sample size of grid and P2P approaches being relatively low, this may not be immediately evident (see Figure 3). Instead developments in this area appear to be occurring in cloud computing and other environments that leverage virtualisation.

Prepared using speauth.cls
DOI: 10.1002/spe

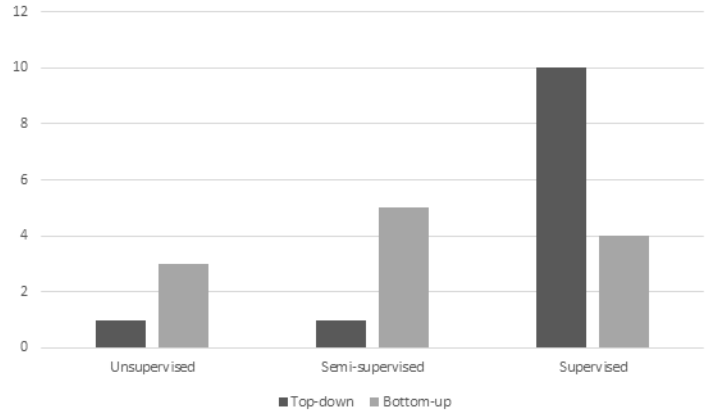

Figure 5. Learning Methodologies versus Management Styles

The learning methodologies leveraged in self–healing systems appear to be strongly correlated with their respective management styles. In the case of top-down, (*i.e.*, centrally managed) styles, self–healing frameworks overwhelmingly support the use of supervised learning. Conversely, systems that operate in an ad–hoc fashion (*i.e.*, 'bottom-up') fashion are substantially more likely to leverage semi–supervised or unsupervised approaches.

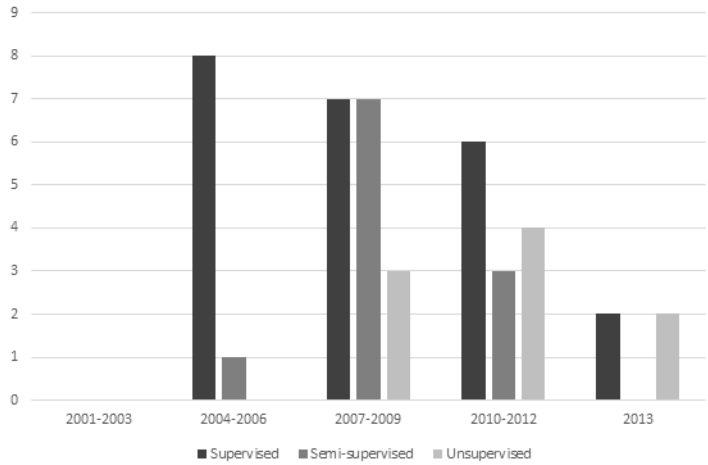

Figure 6. Learning Methodologies in Self-Healing Frameworks by Year of Introduction

Self-healing systems frameworks are showing greater autonomy in their learning behaviours. In the last 10 years we've seen the number of self-healing frameworks that leverage supervised approaches decline, whilst those with either semi–supervised or unsupervised techniques are increasing.

Prepared using speauth.cls
DOI: 10.1002/spe

Author	Year	Title	Learning Methodology			Management Style		Computing Environment		
			Supervised	Semi-supervised	Unsupervised	Bottom-up	Top-down	n-Tier, Traditional	Cloud, Virtualised	Grid/P2P
Dean	2013	UBL			•	•		•	•	
Li	2013		•				•	•	•	
Gomaa	2012	SASSY-ext			•		•	•	•	
Stojnic	2012	OSIRIS-SR	•			•	•		•	•
Menasce	2011	SASSY	•				•	•	•	
Psaier	2010	Viecure		•			•		•	
Simmonds	2010		•				•	•	•	
Miorandi	2010	Embroware		•	•	•			•	•
Pernici	2009	WS-DIAMOND	•			•		•	•	
Ahmed	2009		•				•			•
Ramirez	2009	Plato		•	•	•		•	•	•
Cardellini	2009	MOSES	•				•	•	•	
Aldinucci	2008		•				•		•	•
Calinescu	2007	GPAC		•		•			•	•
Shehory	2007	SHADOWS		•		•		•	•	
Naccache	2006		•			•		•	•	
Rillin	2006	Vigne	•			•	•			•
Messig	2005	WSRF		•		•				•
Shang	2004	Rainbow	•				•	•	•	
Schuler	2004	OSIRIS	•				•		•	•

Figure 7. Summary of Findings: Self-Healing Systems Frameworks

Self-healing systems frameworks as categorised by supervisory requirement, computing environment, and expected management style. In some cases frameworks exhibit abilities to operate under multiple assumptions–these incidents are represented by concurrent bullets within the graph.

3. DISCUSSION

Self-healing systems methodologies are becoming more autonomous, but remain dependent upon either required periodic human interaction or the acceptance of uncertainty in systems behaviours. This finding comes as self-healing methodologies being to specialise based on external factors such as their intended computing environment and respective management styles. Notably, a framework's specialisation has been shown to provide distinct advantages in autonomously identifying and resolving faults. These advantages play pivotal roles in understand how self-healing frameworks are evolving. Furthermore, many approaches display behaviours that are not universally desirable-self-healing approaches are diverging based on their specialisations. This is a concept that until now has not been explicitly addressed within the field. By contrasting where self-healing frameworks are being implemented, an understanding is gained of where self-healing systems are making progress and towards which specific problems.

The intended computing environment of a given framework is a foundational factor in evaluating the success of its self-healing behaviours and has produced a divergence in the types of self-healing systems that are being developed. Environments that require a greater degree of control of its systems often exhibit centralised management techniques [23, 24, 26, 28–30, 32, 42, 59]. These approaches are evaluated based on how predictable their behaviours are, and often intentionally build in a requirement for human intervention. Conversely, frameworks that operate in *ad hoc* infrastructures [31, 33, 35–37, 39] are often

DOI: 10.1002/spe

expected to exhibit behaviours that do not require human intervention, and in some cases to synthesise new self-healing strategies. This result is an artefact of computing environments having inherently different properties, exigencies, and requirements. The result has been that self-healing frameworks have begun to develop specialised strategies that address each of these factors explicitly.

Evidence of specialisation in self-healing strategies is becoming increasingly more common as frameworks exhibit hybrid approaches for mobile [31] and centralised computing environments [33]. These approaches place a specific emphasis on leveraging different self-healing strategies based on the environmental suitability of the approach at run-time, and by anticipating resource availability. Notably, resource prediction is being leveraged more often where assumptions cannot consistently be made about the state of computing environment–particularly where resources are transient [31, 39, 40, 66], or virtualised [35, 67]. In these situations self-healing frameworks leverage multiple concurrent strategies to address greater degrees of systems volatility. Likewise, frameworks have leveraged various approaches for identifying and mitigating faults based on local and remote observations within their respective environments [26, 38, 68].

Although the approaches used by self-healing systems are varied, there are trends as to which methodologies are being leveraged and under what circumstances. Systems within environments that exhibit a high degree of churn are more likely to leverage *ad hoc* management styles [30, 31, 40, 66], and learning methodologies that require less supervision [33, 35, 37, 59]. Conversely, frameworks that do not have stable systems membership are more likely to utilise a centralised form of systems management [23, 25, 26, 29, 42, 59], and exhibit supervised or semi-supervised learning methodologies [26–29, 31, 32, 36, 38, 41, 42]. The predictability of a self-healing framework's actions are crucial in identifying operational requirements (*i.e.*, SLAs), and are a defining factor in what behaviours are allowed or desirable in its respective computing environment. As behaviours are nearly solely defined by learning methodologies, is it clear that the relationship between management style and environment is linked with the degree of supervision required for its continual operation.

Using an *ad hoc* management style allows self-healing frameworks to leverage more autonomous strategies and learning methodologies. However, systems that engage in self-elected behaviours–particularly those that have not been previously vetted–have been shown to be inherently more risky when attempting to meet operational goals and less likely to produce reusable solutions [56–58]. It is for this reason that the use of centralised management techniques remain the preferred approach when environments are expected to exhibit a low rate of churn – the most notable examples being CBR and CBR-like learning methodologies [23–26, 36, 42].

The advantages of self-healing approaches are directly related to their supervisory requirements. Although supervised learning methodologies have shown advances towards reducing human overhead, when compared to unsupervised methodologies, they have ultimately produced palliative results–particularly when executing recovery strategies. This is primarily due to the fact that supervised techniques can only reactively detect faults [35], and that the solutions they generate often must be vetted via human intervention before being implemented. These solutions can become increasingly more complex to manage as the interdependency of features must be accounted for in subsequent self-healing strategies [50]. Such solutions are difficult to vet as often relationships between features are not immediately accessible either algorithmically or intuitively.

Semi-supervised and unsupervised approaches have shown stronger capabilities in ascertaining the root cause of a given fault, and producing non-palliative recovery solutions. In particular, the use of evolutionary programming techniques have demonstrated the unique ability to autonomously generate new systems configurations at run-time to mitigate faults [37], and the use of artificial neural networks have been show to correlate specific systems configurations with operational fitness levels to produce predictive fault detection [35]. These approaches show greater capabilities for autonomously self-healing of faults, but,

DOI: 10.1002/spe

like supervised methodologies, also come with certain restrictions. Notably, the resources needed by unsupervised approaches can be much greater than supervised approaches, and frameworks leveraging these methods are not assured of finding a solution [35, 37]. These are properties inherent to the nature of search-space methodologies–either a predefined constraint is exhausted (*e.g.*, time), or an acceptable solution is converged upon [51]. Exploration into these issues remain a separate field of study and outwith the scope of this survey–however it is clear they are deeply related to the viability of self-healing solutions.

The future of self-healing systems research is multi-faceted, and remains open to further exploration. Recent advances in self-healing systems have seen a wider range of issues being addressed and in more complex environments. However, as environments continue to develop, and the relationships between self-healing solutions continues to co-evolve with their implementation requirements, still greater questions are being asked.

Self-healing systems frameworks are continuing to explore new methodologies for detecting and mitigating faults. However, which approach is most efficient–and under what circumstances–is an area of research that needs further exploration. At present, very little information is available on this topic–and, there are no publicly verifiably results [10, 69]. This may be in part due to the diverse set of situations in which self-healing systems have been implemented–making direct comparisons difficult–or due to restrictions on releasing this information publicly. A comparison of self-healing systems frameworks using non-simulated data would be greatly beneficial to the field.

The nomenclature used to describe certain aspects of self-healing systems research is becoming increasingly vague. At present, most recovery strategies utilise subsets of other self–*, self-adaptation, or self-management properties. These properties often include aspects in optimisation and configuration–sometimes concurrently. This has led to a diverse understanding of what criteria are acceptable for evaluating the success of a self-healing framework [19, 58, 69, 70]. This multiplicity of terms can be confusing–particularly between systems that autonomously provision themselves within an environment, versus those that can elect new subsets of features or configurations to correct faults. Although some research has been produced in this area, it could do with a more explicit set of definitions.

Self-healing methodologies are continuing to specialise based on their computing environment(s), and showing strides towards greater autonomy. However, these approaches can be more resource intensive and less predictable when leveraging unsupervised learning methodologies. In order to adopt unsupervised learning methodologies into production computing environments, these issues will need to be addressed. It may be possible to reduce resource requirements– and narrow the expected set of behaviours of existing approaches– by guiding the evolutionary programming techniques that are currently being leveraged. The advent of the self-organising map has shown that vector analysis can help pinpoint the location of faults by examining feature data from an historical perspective. If this information were to be leveraged via a genetic algorithm, it could substantially reduce the search-space needed for generating a new, working systems configuration.

Lastly, there are still further methods to explore in building self-healing frameworks. The use a hidden markov-model may be able to provide similar results to that of the self-organising map, but with potentially less required training time. To date, there are no known self-healing frameworks that utilise this particular methodology. The latter two areas of research are topics of research we intend to explore.

4. CONCLUSION

IBM originally predicted that building an autonomic system would be an evolutionary process. As self-healing frameworks exhibit greater autonomy this prediction is being realised in a more literal than figurative manner. The use of machine learning and evolutionary programming techniques have shown how systems can predictively mitigate faults without human interaction, and can complement the use of policies and other human-driven approaches. Ideas from

DOI: 10.1002/spe

biological and physical systems offer further inspiration and another axis in which to evaluate the construction of decentralised adaptive systems. The ability to correlate stochastic, search-based techniques with performance metrics and configuration data shows a promising venue for autonomously contextualising systems behaviours.

However, much work remains to be done in building predictive methodologies if the original self-healing tenet is to be fully realised. Understanding the relative effectiveness of self-healing frameworks with respect to their preferred methodologies, and realising self-adaptive systems that can autonomously detect and recover from faults without human intervention remain core challenges. As of yet there are few if any resources that discuss direct comparisons between the effectiveness of self-healing systems frameworks. One of the largest problems appears to be access to live information and resources that can accommodate realistic testing of such services. A single study in this area would likely prove to be immensely beneficial to the field.

Furthermore, as self-healing approaches continue to advance the division between self-configuration, protection, and optimisation is being blurred. As discussed in section 1.1.3, the definitions of self-configuration, protection, and optimisation share properties and technological boundaries with self-healing. As such, self-healing strategies often leverage techniques that fall into these previously separately defined boundaries. Stronger definitions or revisiting the terminology used in the field may produce beneficial results in the form of more direct contributions–particularly to studies in autonomous provisioning and security. The systematic integration of these properties, in a way that allows them to be evaluated and traded-off to maximise a system's pursuit of its self-healing mission, remains a core challenge, as much now as it did a decade ago.

REFERENCES

1. Kramer J, Magee J. Self-managed systems: an architectural challenge. *Future of Software Engineering (FOSE '07)*, 2007.
2. Dobson S, Denazis S, Fernández A, Gaïti D, Gelenbe E, Massacci F, Nixon P, Saffre F, Schmidt N, Zambonelli F. A survey of autonomic communications. *ACM Transactions on Autonomous and Adaptive Systems* 2006; **1**:223–259, doi:10.1145/1186778.1186782.
3. Psaier H, Dustdar S. A survey on self-healing systems: approaches and systems. *Computing* 2010; **91, Issue: 1**:43–73.
4. Ghosh D, Sharman R, Raghav Rao H, Upadhyaya S. Self-healing systems - survey and synthesis. *Decision Support Systems* January 2007; **42**(4):2164–2185.
5. Horn P. Autonomic computing: IBM's perspective on the state of information technology. 2001; .
6. Ganek AG, Corbi TA. The dawning of the autonomic computing era. *IBM Systems Journal* 2003; **42 , Issue: 1**:5–18.
7. Kephart JO, Chess DM. The vision of autonomic computing. *Computer* 2003; **36, Issue: 1**:41–50.
8. Kephart JO, Walsh WE. An artificial intelligence perspective on autonomic computing policies. *Fifth IEEE International Workshop on Policies for Distributed Systems and Networks* June 2004; :3–12.
9. Russell S, Norvig P. *Artificial Intelligence: A Modern Approach. 2nd Edition*. Prentice Hall, 2003.
10. Diao Y, Hellerstein JL, Parekh S, Griffith R, Kaiser G, Phung D. Self-managing systems: A control theory foundation. *Engineering of Computer-Based Systems* 2005; .
11. Kephart JO. Research challenges of autonomic computing. *Proceedings of the 27th international conference on software engineering* 2005; :15–22.
12. Brodie M, Ma S, Lohman G, Syeda T, Mahmood L, Mignet N, Modani, Wilding M, Champlin J, Sohn P. Quickly finding known software problems via automated symptom matching. *Proceedings of the Second International Conference on Autonomic Computing.* 2005; .
13. Candea G, Kawamoto S, Fujiki Y, Friedman G, Fox A. Microreboot-a technique for cheap recovery. *In Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)* 2004; .
14. Boutilier C, Das R, Kephart JO, Tesauro G, Walsh WE. Cooperative negotiation in autonomic systems using incremental utility elicitation. *Nineteenth Conference on Uncertainty in Artificial Intelligence* 2003; :89–97.
15. Braynard R, Kostic D, Rodriguez A, Chase J, Vahdat A. Opus: an overlay peer utility service. *Proceedings of the 5th International Conference on Open Architectures and Network Programming (OPENARCH)* 2002; .
16. Irwin DE, Grit LE, Chase J. Balancing risk and reward in market-based task scheduling. *Proceedings of the Thirteenth International Symposium on High Performance Distributed Computing (HPDC-13)* 2004; .
17. Walsh WE, Tesauro G, Kephart JO, Das R. Utility functions in autonomic systems. *Proceedings of the First International Conference on Autonomic Computing*, 2004; 70–77.

Prepared using speauth.cls
DOI: 10.1002/spe

18. Kephart JO. Autonomic computing: The first decade. *International Conference on Autonomic Computing* 2011; .
19. Rodosek GD, Geihs K, Schmeck H, Burkhard S. Self-healing systems: Foundations and challenges. *Self-Healing and Self-Adaptive Systems*, Dagstuhl Seminar Proceedings Series, Schloß Dagstuhl, 2009.
20. Salehie M, Tahvildari L. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.* May 2009; **4**(2):14:1–14:42.
21. Kirby G, Dearle A, Macdonald A, Fernandes A. An approach to ad hoc cloud computing. *ArXiv.org* 2010; Http://arxiv.org/pdf/1002.4738.pdf.
22. Sloman M. Policy driven management for distributed systems. *Journal of Network and Systems Management* 1994; **2**:333–360.
23. Cheng SW, Huang AC, Garlan D, Schmerl BR, Steenkiste P. Rainbow: Architecture-based self-adaptation with reusable. *ICAC*, 2004; 276–277.
24. Ahmed S, Ahamed SI, Sharmin M, Hasan CS. Self-healing for autonomic pervasive computing. *Autonomic Communication*. Springer US, 2009; 285–307.
25. Aldinucci M, Danelutto M, Zoppi G, Kilpatrick P. Advances in autonomic components and services. *From Grids to Service and Pervasive Computing*, Priol T, Vanneschi M (eds.). Springer US, 2008; 3–17.
26. Simmonds J, Ben-David S, Chechik M. Monitoring and recovery of web service applications. *The Smart Internet*, *Lecture Notes in Computer Science*, vol. 6400. Springer-Verlag, 2010; 250–288.
27. Cheng SW, Garlan D, Schmerl B. Architecture-based self-adaptation in the presence of multiple objectives. *ICSE 2006 Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2006.
28. Menasce D, Gomaa H, Malek S, Sousa J. Sassy: A framework for self-architecting service-oriented systems. *Software, IEEE* 2011; **28**(6):78–85, doi:10.1109/MS.2011.22.
29. Cardellini V, Casalicchio E, Grassi V, Iannucci S, Lo Presti F, Mirandola R. Moses: A framework for qos driven runtime adaptation of service-oriented systems. *IEEE Transactions on Software Engineering* 2011; **PP**(99):1–23. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5963694`.
30. Schuler C, Weber R, Schuldt H, j Schek H. Scalable peer-to-peer process management - the osiris approach. *In: Proceedings of the 2 nd International Conference on Web Services (ICWS'2004)*, IEEE Computer Society, 2004; 26–34.
31. Stojnic N, Schuldt H. Osiris-sr: A safety ring for self-healing distributed composite service execution. *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on*, 2012; 21–26, doi:10.1109/SEAMS.2012.6224387.
32. Psaier H, Skopik F, Schall D, Dustdar S. Behavior monitoring in self-healing service-oriented systems. *Socially Enhanced Services Computing*. Springer Vienna, 2011; 95–116.
33. Miorandi D, Lowe D, Yamamoto L. Embryonic models for self–healing distributed services. *Bioinspired Models of Network, Information, and Computing Systems*, *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, vol. 39. Springer Berlin Heidelberg, 2010; 152–166.
34. Miorandi D, Carreras I, Altman E, Yamamoto L, Chlamtac I. Bio-inspired approaches for autonomic pervasive computing systems. *Bio-Inspired Computing and Communication*, vol. 5151. Springer Berlin, 2008; 217–228.
35. Dean DJ, Nguyen H, Gu X. Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems. *Proceedings of the 9th international conference on Autonomic computing*, ICAC '12, ACM: New York, NY, USA, 2012; 181–190, doi:10.1145/2371536.2371571. URL `http://doi.acm.org/10.1145/2371536.2371571`.
36. Pernici B. Self-healing systems and web services: The ws-diamond approach. *Business Process Management Workshops*, *Lecture Notes in Business Information Processing*, vol. 17. Springer Berlin Heidelberg, 2009; 440–442.
37. Ramirez AJ, Knoester DB, Cheng BH, Mckinley PK. Plato: a genetic algorithm approach to run-time reconfiguration in autonomic computing systems. *Cluster Computing* Sep 2011; **14**(3):229–244.
38. Shehory O. *A Self-healing Approach to Designing and Deploying Complex, Distributed and Concurrent Software Systems*, *Lecture Notes in Computer Science*, vol. 4411. Springer-Verlag, 2007; 3–13.
39. Calinescu R. General-purpose autonomic computing. *Autonomic Computing and Networking*. Springer US, 2009; 3–30.
40. Stoica I, Morris R, Karger D, Kaashoek MF. Chord: A scalable peer-to-peer lookup service for internet. *Proceedings of the ACM SIGCOMM '01 Conference*, 2001.
41. Naccache H, Gannod G, Gary K. A self-healing web server using differentiated services. *Service-Oriented Computing – ICSOC 2006*, *Lecture Notes in Computer Science*, vol. 4294. Springer Berlin / Heidelberg, 2006; 203–214.
42. Li G, Liao L, Song D, Wang J, Sun F, Liang G. A self-healing framework for qos-aware web service composition via case-based reasoning. *Web Technologies and Applications*, *Lecture Notes in Computer Science*, vol. 7808. Springer Berlin Heidelberg, 2013; 654–661.
43. Carzaniga A, Gorla A, Pezzè M. Healing web applications through automatic workarounds. *International Journal on Software Tools for Technology Transfer (STTT)* 2008; **10**:493–502. URL `http://dx.doi.org/10.1007/s10009-008-0088-8`, 10.1007/s10009-008-0088-8.
44. Hassan S, McSherry D, Bustard D. Autonomic self healing and recovery informed by environment knowledge. *Artificial Intelligence Review* 2006; **26**:89–101. 10.1007/s10462-007-9033-6.
45. Kamvar S, Schlosser M, Garcia-Molina H. The eigentrust algorithm for reputation management in p2p networks. *Proceedings of the 12th international conference on World Wide Web*, WWW '03, ACM: New York, NY, USA, 2003; 640–651.
46. Chess DM. Security in autonomic computing 2005; **33**, doi:10.1145/1055626.1055628.

DOI: 10.1002/spe

47. Gustavsson R, Ståhl B. Self-healing and resilient critical infrastructures. *Critical Information Infrastructure Security*, *Lecture Notes in Computer Science*, vol. 5508. Springer Berlin / Heidelberg, 2009; 84–94.
48. Rilling L. Vigne: Towards a self-healing grid operating system. *Euro-Par 2006 Parallel Processing*, *Lecture Notes in Computer Science*, vol. 4128. Springer Berlin / Heidelberg, 2006; 437–447.
49. Baduel L, Matsuoka S. A decentralized, scalable, and autonomous grid monitoring system. *Principles of Distributed Systems*, *Lecture Notes in Computer Science*, vol. 4878. Springer Berlin / Heidelberg, 2007; 1–15.
50. Garvin B, Cohen M, Dwyer M. Failure avoidance in configurable systems through feature locality 2013; **7740**:266–296. URL `http://dx.doi.org/10.1007/978-3-642-36249-1_10`.
51. Holland JH. Adaptation in natural and artificial systems. *MIT Press, Cambridge, MA. US.* 1992; .
52. Metzger A, Sammodi O, Pohl K. Accurate proactive adaptation of service-oriented systems. *Assurances for Self-Adaptive Systems*, *Lecture Notes in Computer Science*, vol. 7740, Cmara J, Lemos R, Ghezzi C, Lopes A (eds.). Springer Berlin Heidelberg, 2013; 240–265, doi:10.1007/978-3-642-36249-1_9.
53. Fernandez-Marquez J, Di Marzo Serugendo G, Montagna S. Bio-core: Bio-inspired self-organising mechanisms core. *Bio-Inspired Models*, vol. 103. Social informatics and telecommunications engineering edn., Lecture Notes of the, 2012; 59–72.
54. Montagna S, Pianini D, Virolio M. Gradient-based self-organisation. *6th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2012)*, 2012; 10–14.
55. Chess DM, Kumar V, Segal A, Whalley I. Work in progress: Availability-aware self-configuration in autonomic systems. *Utility Computing*, *Lecture Notes in Computer Science*, vol. 3278. Springer Berlin / Heidelberg, 2004; 257–258.
56. de Lemos R. The conflict between self-* capabilities and predictability. *Self-star Properties in Complex Information Systems: Conceptual and Practical Foundations*. Springer, 2005; 219–229.
57. McCann J, de Lemos R, Heubscher M, Rana FO, Wombacher A. Can self-managed systems be trusted? some views and trends. *Knowledge Engineering Review* September 2006; **21**(3):239–248.
58. McCann J, Huebscher M. Evaluation issues in autonomic computing. *Grid and Cooperatve Computing - GCC 2004 Workshops*, vol. 3252. Springer Berlin, 2004; 597–608.
59. Gomaa H, Hashimoto K. Dynamic self-adaptation for distributed service-oriented transactions. *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on*, 2012; 11–20, doi:10.1109/SEAMS.2012.6224386.
60. Engel Y, Etzion O. Towards proactive event-driven computing. *Proceedings of the 5th ACM international conference on Distributed event-based system*, DEBS '11, ACM: New York, NY, USA, 2011; 125–136.
61. Ramirez AJ, Knoester DB, Cheng BH, McKinley PK. Applying genetic algorithms to decision making in autonomic computing systems. *Proceedings of the 6th international conference on Autonomic computing*, ICAC '09, ACM: New York, NY, USA, 2009; 97–106.
62. Ortega-Sanchez C, Mange M, Smith S, Tyrrell A. Embryonics: a bio-inspired cellular architecture with fault-tolerant properties. *Genetic Programming and Evolvable Machines*. 2000; 187–215.
63. Prodan L, Tempesti G, Mange D, Stauffer A. Embryonics: artificial stem cells. *In: Proc. of ALife VIII*. 2002; 101–105.
64. Kohonen T. The self-organizing map. *Proceedings of the IEEE* 1990; **78**(9):1464–1480.
65. Zheng Z, Yu L, Lan Z, Jones T. 3-dimensional root cause diagnosis via co-analysis. *Proceedings of the 9th international conference on Autonomic computing*, ICAC '12, ACM: New York, NY, USA, 2012; 181–190, doi:10.1145/2371536.2371571. URL `http://doi.acm.org/10.1145/2371536.2371571`.
66. Tauber M, Kirby G, Dearle A. Autonomic management of maintenance scheduling in chord. *CoRR* 2010; **abs/1006.1578**.
67. Dai Y, Xiang Y, Zhang G. Self-healing and hybrid diagnosis in cloud computing. *Cloud Computing*, *Lecture Notes in Computer Science*, vol. 5931. Springer Berlin / Heidelberg, 2009; 45–56.
68. Snyder P, Valetto G, Fernandez-Marquez J, di Marzo Serugendo G. Augmenting the repertoire of design patterns for self-organized software by reverse engineering a bio-inspired p2p system. *Proceedings of the 6th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2012)*, 2012.
69. Brown A, Redline C. Measuring the effectiveness of self-healing autonomic systems. *Proceedings of the Second International Conference on Autonomic Computing 2005* 2005; .
70. Funika W, Pęgiel P. A role-based approach to self-healing in autonomous monitoring systems. *Parallel Processing and Applied Mathematics*, *Lecture Notes in Computer Science*, vol. 6068. Springer Berlin / Heidelberg, 2010; 125–134.

DOI: 10.1002/spe