State of runtime adaptation in service-oriented systems: what, where, when, how and right

Leah Mutanu1, Gerald Kotonya2

1School of Science and Technology, United States International University, P.O. Box 14634-00800, Nairobi, Kenya
2School of Computing and Communication, Lancaster University, Lancaster LA1 4WA, UK

Abstract: Software as a Service reflects a ‘service-oriented’ approach to software development that is based on the notion of composing applications by discovering and invoking network-available services to accomplish some task. However, as more business organisations adopt service-oriented solutions and the demands on them grow, the problem of ensuring that the software systems can adapt fast and effectively to changing business needs, changes in their runtime environment and failures in provided services has become an increasingly important research problem. Dynamic adaptation has been proposed as a way to address the problem. However, for adaptation to be effective several other factors need to be considered. This study identifies the key factors that influence runtime adaptation in service-oriented systems (SOSs) and examines how well they are addressed in 29 adaptation approaches intended to support SOSs.

1 Introduction

Service-Oriented Architecture (SOA) provides the conceptual framework for realising service-oriented systems (SOSs) by supporting dynamic composition and reconfiguration of software systems from networked software services [1]. Rosen et al. [2] identify the key motivations for SOA as agility, flexibility, reuse, integration and reduced cost. However, the need to ensure that the systems can adapt quickly and effectively to changing business needs, changes in system quality and changes in their runtime environment is an increasingly important research problem [3]. Effective adaptation ensures the system remains relevant in a changing environment and is an accurate reflection of user expectations. Taylor et al. [4] define dynamic adaptation as the ability of a software system’s functionality to be changed at runtime without requiring a system reload or restart.

Taylor et al. point out that there is an increasing demand for non-stop systems, as well as a desire to avoid annoying users. However, current approaches for supporting runtime adaptation in SOSs differ widely with respect to the nature of systems they support, the types of system changes they support and their underlying model of adaptation [5, 6]. In addition, it is also unclear how these approaches address the important issue of ensuring the adaptation is effective. A growing consensus amongst researchers is that runtime adaptation in SOA should incorporate a validation element [7, 8].

In their research roadmap for self-adaptive systems, Lemos et al. [9] emphasise the need for feedback control in the life cycle of self-adaptive systems, and the need to perform traditional design-time verification and validation at runtime. In another survey, Salehic and Tahvildari [10] note that testing and assurance are probably the least focused phases in the engineering of self-adaptive software. Papazoglou et al. [8] echo this view. They note that the bulk of research in adaptive SOAs has focused largely on dynamic compositions. Adaptation validation goes beyond verifying that the adaptation conforms to its operational specification. Validation is concerned with verifying the acceptability of an adaptation, often from the point of view of the system user – i.e. is it the right adaptation for the problem as opposed to whether it is specified right? Validation assesses the effectiveness of an adaptation. Because user requirements are constantly changing, a self-validation process would enable the adaptation system to self-assess and self-evolve in order to remain relevant.

This paper identifies the key research challenges and the factors that influence runtime adaptation in SOSs. The influencing factors are used as a basis for reviewing 29 approaches intended to support runtime adaptation in SOSs. The survey compliments existing surveys and extends our earlier survey [11] to include an in-depth review of runtime validation in SOSs.

Notable additions in this survey include a detailed review of the different models and techniques used to support runtime validation in SOSs, when they are applied, their primary focus and the different strategies employed. Eighteen service-oriented approaches that support runtime validation are reviewed in this context. This survey has three objectives: (i) to provide an overview of the key challenges in runtime adaptation for SOSs; (ii) to propose a simple, but effective scheme for assessing runtime adaptation approaches in SOSs; (ii) to provide an overview of the state of runtime adaptation approaches in SOSs.

The remaining of this paper is organised as follows. Section 2 outlines the key research challenges for runtime adaptation in SOSs. Section 3 identifies the key factors that influence runtime adaptation in SOSs and review how well they are supported in 29 service-oriented approaches. Section 4 provides some concluding thoughts and a look ahead.

2 Research challenges

A number of research initiatives are investigating effective ways to improve on runtime adaptation in SOSs. These initiatives are, however, inadequate for addressing the issues identified in Section 1 for the following reasons:

• Static adaptation rules. Current approaches for supporting runtime adaptation in SOSs are based on rules that reconfigure systems based on fixed decision points which do not take into account the dynamic nature of the factors that influence adaptation [9, 10]. Indeed, Di Nitto et al. [12] attribute the dynamic nature of software to the fact that requirements cannot be fully gathered upfront and cannot be ‘frozen’. Thus, while various studies have been conducted to address the challenge of adapting software to address the ever changing requirements, currently, no single solution to this problem exists. Existing research revolves around the ‘local’ adaptation of specific cases.
Di Nitto et al. highlight the need for research to devise technologies and methods to enable crosscutting adaptations.

- **Poor support for validation.** Current approaches for supporting runtime adaptation in SOSs offer poor support for validation [7, 9]. Like most autonomic systems, runtime adaptation in SOSs is based on IBM’s Monitoring, Analysis, Planning, and Execution model (MAPE) [13]. However, MAPE does not support validation. The lack of mechanisms for validating adaptation makes it difficult to gauge the appropriateness and effectiveness of adaptation decisions and limit our understanding of the nature of problems for which they are suited. Validation provides an avenue for adaptation rules to evolve and remain relevant because the factors that influence adaptation are constantly changing [11]. Validating adaptation goes beyond verifying that the adaptation conforms to its operational specification. Validation is concerned with verifying the acceptability of an adaptation [14], often from the point of view of the system user, i.e. ‘is it the right adaptation for the problem?’ as opposed to ‘is it specified right?’ Validation assesses the effectiveness of an adaptation.

- **Poor support for diversity.** Current approaches for runtime adaptation are built around predefined changes requests and adaptations and are often embedded within the applications they support. This limits their extensibility, portability and the quality of adaptation they offer. For example, Cubo et al. [6] and Tanaka and Ishida [15] describe approaches that are concerned with specific application contexts. Swaminathan [5] and Cardellini et al. [7] propose models that promote context variability, however, the author provides no information about the implementation or evaluation of the models. There is no evidence that the approaches support diversity.

- **Poor support for proactive adaptation.** Most approaches to adaptation in SOSs are reactive [16, 17]. They recompose the system as a reaction to change rather than anticipate change. While reactive adaptation has the advantage of requiring only a small set of recent system conditions to select an adaptation, allowing for a timely decision, it has a number of limitations. First, reactive adaptation is based largely on static system properties and conditions that do not take into account previous aspects of system behaviour that may inform better adaptation selection. Secondly, reactive adaptation lags behind current system conditions, which may be short-lived or change as the adaptation is being carried out resulting in unnecessary adaptations that may impact system quality. Finally, the inability to anticipate change makes it difficult to address disruptive system changes such as service and quality failures in a timely manner. These key challenges represent the gaps in the runtime adaptation of SOSs and are highlighted in this paper through a review of existing work in this area. This paper aims at highlighting the importance of addressing these gaps when developing dynamic self-adaptive service oriented applications.

### 3 Factors that influence adaptation

Most of the work on self-adapting software systems takes inspiration from control theory and machine learning. Control-theory splits the world into a controller and a plant. The controller is responsible for sending signals to the plant, according to a control law, so that the output of the plant follows a reference (the expected ideal output). Fig. 1 shows a typical control loop. Although it is difficult to anticipate when and how change occurs in software systems, it is possible to control when and how the adaptation should react to change.

Dynamic adaptive systems require information about the running application as well as control protocols to be able to reconfigure a system. For example, keeping web services up and running for a long time requires collecting of information about the current state of the system, analysing that information to diagnose performance problems or to detect failures, deciding how to resolve the problem (e.g. via dynamic load-balancing or healing), and acting on those decisions. Fig. 2a shows the control process for a software system equivalent of the physical system shown in Fig. 1. The controller maps onto an adaptation process that reconfigures the runtime system to address the changing needs in its application context. Fig. 2b shows how the adaptation process can be improved using validation. Validation tracks assesses and adjusts adaptations to ensure that they reflect user expectations. Lemos et al. [9] highlight the importance of understanding the factors that influence adaptation. They posit that this helps in the comprehension of how software processes change when developing self-adaptive systems. They describe these factors as design decisions pertinent to self-adaptive systems. These are observation, representation, control, identification, and adaptation mechanism. These decisions, however, do not include validation, which they state is the key to ensuring that the software system satisfies functional requirements and meets their expected quality attributes. The key challenges with current approaches include defining models that can represent a wide range of system properties and the need for feedback control loops in the life cycle of self-adaptive systems and self-validation. The nature and quality of runtime adaptation in SOSs is influenced by system changes (i.e. adaptation triggers), the nature of the application and the logical area where it executes (i.e. application context), the strategy used to reconfigure the system in a particular change context (i.e. adaptation model), and the effectiveness of the adaptation (i.e. validation). Together, these factors, represent the what, where, when and how, and right of runtime adaptation. It is also important...
to note that these factors constantly shift and evolve making it difficult to specify adequate adaptation rules in advance.

Hirschfeld and Kawamura [18] suggest that runtime adaptation in SOs should address the what, when, and how of adaptation (Fig. 3). In Hirschfeld and Kawamura [18], the what distinguishes between the basic properties of the system’s computation, state, and communication. The when addresses the time when adaptations can be made to operate and maintain the system during software development, i.e. at development time, compile-time, load-time, or runtime. The how studies tools and techniques that allow for adaptations to become effective. The what, when and how defined by Hirschfeld et al. relate adaptation to system properties, software development stages, and tools to effect adaptation. This is different from our classification, which relates what, when, and how, and right to change triggers, application context, adaptation model and validation [11]. Our classification is supported by a recent review of the state of runtime adaptation in SOs reveal that there are other important factors (dimensions) such as the application context, adaptation triggers, and validation.

Paktinat et al. [19] present a similar taxonomy of adaptation strategies for service-based systems. They classify adaptation according to what should be monitored when the change should occur, where the problem is located, how adaptation is delivered, why adaptation should occur, and who should be involved. Paktinat et al. do not distinguish triggers, provide no distinction between the different implementation models and do not support validation. Like [18] they also do not examine the effectiveness of the adaptation process. A comparison between the terminology and scope of our approach, and Hirschfeld and Kawamura [18] and Paktinat et al. [19] is shown in Table 1. The elements of our approach are discussed in detail next.

3.1 Change trigger (what)

A change trigger represents what causes adaptation and the reason for it. Change triggers are a function of changes in the business environment, service failure, and changes in the system quality and its runtime environment.

• Business Environment Triggers. Changes in the business environment that the system supports may trigger adaptation. This may be caused by changes in user requirements, business rules or platform. Zeng et al. [20] describe an adaptation approach that accepts changes in user requirements and business rules on the fly and composes services to address them. Similarly, Cubo et al. [6] describe an approach that uses changes in the context environment and platform to trigger adaptation. Because user requirements are not static and constantly change at runtime, any adaptation solution should monitor the business environment and adapt the system accordingly.

• Service Provision Trigger. Failures in provided services, e.g. incompatibilities that impact on service composition, network outages and poor service quality, could trigger adaptation. The quality of an SO depends not only on the quality of the provided service but on the interdependencies between services and resource constraints imposed by the runtime environment. This type of corrective adaptation is typical of self-healing systems. Robinson and Kotonya [21] describe an approach that uses a consumer-centred, pluggable brokerage model to track and renegotiate service faults and changes. The framework provides a service monitoring system, which actively monitors the quality of negotiated services for emergent changes, SLA violations and failure. A similar approach, The Personal Mobility Manager, described in [22] emphasises the need for automatic system diagnosis to detect runtime errors. It helps car

![Fig. 3 Adaptation dimensions](image-url)

### Table 1 Runtime adaptation taxonomies

<table>
<thead>
<tr>
<th>Reference</th>
<th>Trigger (cause)</th>
<th>Logical execution area</th>
<th>When carried out</th>
<th>Strategy</th>
<th>Validation</th>
<th>Focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>[18]</td>
<td>—</td>
<td>—</td>
<td>when</td>
<td>how</td>
<td>—</td>
<td>development process</td>
</tr>
<tr>
<td>[19]</td>
<td>what</td>
<td>where</td>
<td>when</td>
<td>how, why, who</td>
<td>—</td>
<td>system</td>
</tr>
<tr>
<td>this work</td>
<td>what</td>
<td>where</td>
<td>when</td>
<td>how</td>
<td>right</td>
<td>system</td>
</tr>
</tbody>
</table>

Effective adaptation must address the real cause rather than the symptom. Taiani and Fabre [25] describe this as a key challenge in adaptive fault tolerant computing. Moyano et al. [26] describe a system that monitors service failure and runtime environment triggers. These are changes in hardware and firmware, including the unpredictable arrival or disappearance of devices and software component. For example, a low memory trigger may be the result of an SLA violation or runtime environment resource failure. The resolution to the problem might involve replacing the service with a more efficient alternative or optimising the runtime environment, or both. It is important that the adaptation process is not only able to find a good fit for the problem but the right fit.

It is worth noting that adaptation triggers are not mutually exclusive; there is often significant overlap between them. For example, when a user is trying to access a travel assistant the user's environment acts as the source of the trigger. Additionally, if the
user is using a mobile device, which has limited memory resources, then the runtime environment acts as the source of another trigger. A service provision trigger will arise from the QoS required by the user. Triggers can also invoke other triggers and therefore overlap. For example, a user accessing the application from an urban, industrial, or affluent geographic location will often require services of high reputation and may not have devices with resource constraints. The geographic location model based on the Q-Learning algorithm invokes the runtime and service provision triggers. A change or failure may be the symptom of an unseen change or failure. Effective adaption must, therefore, address the real cause rather than the symptom.

3.2 Application context (where)

An application context defines the nature of the application and the logical area where it executes. It helps us understand where adaptation takes place and the constraints involved. We distinguish between approaches that are intended to support specific application contexts and generic solutions that can be tailored for different application contexts. Examples of approaches intended for specific application contexts include Zeng et al. [20] and Autili et al. [27] whose work target automotive and manufacturing domains. Examples of generic approaches include Swaminathan [5] and Cardellini et al. [7].

Cubo et al. [6] discuss the importance of creating adaptive systems sensitive to their application context (i.e. domain, location, time and activity). Tanaka and Ishida [15] identify an input language and a target language as the application context for a language translation application. They, however, do not provide evidence that their approach can be used in a different application context. Most of the approaches surveyed in this paper were concerned with specific application contexts. Zeng et al. [20], for example, describe a runtime approach for supporting a business change in the automotive industry. Similarly, Newman and Kotonya [24] describe an adaptation framework specifically for embedded resource-constrained environments. Baresi et al. [28] describe an adaptation framework specifically for a smart home system. Specific application contexts contain few data which is easy to process for decision making. Generic application contexts, however, contain a lot of information beyond what is actually needed and this is difficult to process. Most of the work that addresses specific application contexts does not provide insight into how such systems would work in a different application or where different triggers existed.

In their description for the DigiHome architecture Romero et al. [29] discuss the integration of multi-scale entities where different application contexts are addressed. In the DigiHome scenario they consider several heterogeneous devices that generate isolated events, which can be used to obtain valuable information and to make decisions accordingly. They make use of Complex Event Processing (CEP), to find relationships between a series of simple and independent events from different sources, using previously defined rules. CEP is useful in getting better information at a real time in generic applications. A few other approaches, including Swaminathan [5], Cardellini et al. [7], and Zeng et al. [30] propose generic application contexts, but they only provide sketchy implementation details. Some approaches promote context variability. For example, Swaminathan describes a context-independent, self-configuring, self-healing model for web services. However, the author provides no information about the implementation or evaluation of the model. Huang and Steenkiste [31] describe an approach for developing self-configuring services using service-specific knowledge. They evaluate their approach on three different systems (i.e. a video streaming service, an interactive search service, and a video-conference service). However, it is evident from their discussion that the context needs to be known before the application is deployed.

Most of the approaches reviewed here were intended for specific application contexts making them difficult to transfer and reuse.

3.3 Adaptation model (when and how)

An adaptation model indicates when the adaptation process is carried out and how the model is implemented in relation to the system it manages. A decision on when to conduct adaptation is arrived at depending on when the adaptation requirements are known as well as the availability of the requirements for adaptation. Runtime adaptation corresponds to situations where the requirements are only known after the system has started executing. This is the typical situation in ubiquitous and mobile computing scenarios. The availability of the requirements for adaptation, such as system resources can also determine when to conduct adaptation. For example, if the resources are available online then dynamic adaptation can be conducted; otherwise, it can be pushed to a later time when they will be available. Table 1 provides a summary of current approaches for runtime adaptation. Papazoglou et al. [8] and Baresi [3] identify the key techniques that can be used to achieve runtime adaptation as self-configuring, self-healing, and self-optimising techniques.

- **Self-Configuring** is the automatic re-composition of services to adapt to changes in the service environment. The work of [20, 32, 33] describes self-configuring adaptation techniques.
- **Self-Optimising** is the automatic re-composition of services to improve the quality of a service. The work of [23, 25, 30] describes self-optimising adaptation techniques.
- **Self-Healing** is the automatic re-composition of services to address a service failure. Self-healing techniques detect system malfunctions and initiate policy based corrective actions without disrupting the runtime environment [8].

Romay's et al. [33] review of self-adaptation techniques in SOA reveals that current research focuses largely on self-configuring techniques. There is very little research on self-optimising or self-healing techniques. Bucchiarelo et al. [34] note that focusing on only one technique limits the effectiveness of the approach. Our survey focuses on two aspects of the Adaptation Model — the nature of its implementation (i.e. pluggable versus embedded) and the strategy adopted to effect the implemented technique (i.e. reactive versus predictive). An implementation may be associated with any of the many adaptation techniques. We believe that this high-level view provides a more transferrable and reusable description of the underlying adaptation model.

3.3.1 Adaptation strategy: Predictive versus reactive: Adaptation can occur in response to anticipated changes (predictive) or in response to change trigger (reactive). Reactive adaptation controls and adapts the environment according to the users’ situation. The system perceives its environment through sensors and reacts to changes as they occur. An ideal predictive approach does not take into account the reactions of the system under control, but only the environment under which it operates. The control of the environment is not pegged on observing the reactions of the user. As a result, such an approach should offer quick response and high performance with few sensors required.

Tanaka and Ishida [15] propose a model that focuses on predicting the executability of services (i.e. if a message request will cause execution failure). Unfortunately, they provide limited detail on the implementation and evaluation of their approach. In their event-driven QoS prediction approach, Zeng et al. [20] point out that most adaptation approaches focus on monitoring QoS constraints and as such cannot provide early warning to prevent QoS degradation. They describe a model that makes use of data mining and prediction scoring to anticipate change. However, they provide only limited information on its evaluation. Wang et al. [35] propose a predictability model based on the Q-Learning algorithm using the Markov Decision Processes. They explain that human oriented services are rarely predictable. They point out that many service properties keep changing in a manner that prior knowledge of these changes may not be available. Instead, they suggest incorporating reinforced learning in adaptation techniques to ensure that adaptation techniques remain relevant. Their model uses a decision process that maximises the expected sum of
3.3.2 Model implementation: An adaptation model can be implemented as an intrinsic part of the system it manages or as a pluggable framework that monitors change variables and effects re-composition from outside the system. Garlan et al. [36] depict a pluggable approach where the adaptation module is plugged on to legacy systems. In their work, an external model is used to monitor and modify a system dynamically. Most of the initiatives surveyed however adopted an embedded approach. Zeng et al. [20] and Cubo et al. [6] are typical of this approach. However, there is growing acknowledgement amongst researchers that a pluggable approach offers a better engineering solution. Pathan et al. [37] propose a generic approach to context aware modelling through the use of a separate component for context reasoning. Garlan et al. [36] state that the use of external control mechanisms for self-adaptivity is a more effective engineering solution than localising the solution. A pluggable engine can be analysed, modified, extended, and reused across different systems. Most solutions presented in the existing literature were embedded which limits their reusability and portability.

3.4 Summary

Table 2 shows our results of surveying 29 approaches that provide runtime adaptation for SOSs. It is important to mention that the survey was intended to be representative rather than exhaustive. The approaches were carefully selected to provide a good coverage of current adaptation in SOSs. To ensure representative coverage all selected approaches to support runtime adaptation and provide some level of support for at least two factors that influence runtime adaptation. The survey list was a collated through multiple searches including; Google Scholar, IEEE and ACM search engines. To the best of our knowledge, we have not ruled out any other approach that might be original or distinctive (in comparison to the 29 selected ones), due to the ‘two factors’ criterion.

Each approach is reviewed in terms of the nature and extent of support for change triggers, adaptation model, validation and application context. Most of the approaches provide limited support for runtime and service provision triggers. However, they provide comparatively good support for business environment triggers. Only Ivanovic et al. [44] describe an approach for supporting all the three adaptation triggers. In their work, they talk about the computational cost of service networks as being dependent on internal and external factors. They recognise that triggers for adaptation are due to overlapping factors that are both internal and external to the service. Of the approaches reviewed, only a few provide support for adaptation validation. However, the support is very limited. There is poor support for diversity with most approaches designed to support specific application contexts. This limitation may be related to the fact that most of the approaches are embedded. Of the approaches surveyed only Zeng et al. [30] provide a detailed discussion of the adaptation techniques used to address the QoS issues that arise from interacting services (i.e. system concerns).

Most of the approaches surveyed provide strong support for dynamic adaptation, which is not surprising as they are intended to support runtime change. However, most of them are implemented as part of the application they manage (i.e. embedded) rather than pluggable. Pluggable approaches include [7, 15, 22, 39, 43]. The

<table>
<thead>
<tr>
<th>Approach</th>
<th>Runtime environment</th>
<th>Service provision</th>
<th>Business environment</th>
<th>Strategy (Reactive (R), Predictive (P))</th>
<th>Adaptation model implementation (Embedded (E), Pluggable (P))</th>
<th>Application context (G = generic, S = specific)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zeng et al. [20]</td>
<td>NS</td>
<td>NS</td>
<td>WS</td>
<td>R</td>
<td>E</td>
<td>S</td>
</tr>
<tr>
<td>Swaminathan [5]</td>
<td>NS</td>
<td>WS</td>
<td>WS</td>
<td>R</td>
<td>N/A</td>
<td>G</td>
</tr>
<tr>
<td>Cubo et al. [6]</td>
<td>NS</td>
<td>NS</td>
<td>S</td>
<td>R</td>
<td>E</td>
<td>S</td>
</tr>
<tr>
<td>Huang et al. [31]</td>
<td>WS</td>
<td>NS</td>
<td>WS</td>
<td>R</td>
<td>E</td>
<td>S</td>
</tr>
<tr>
<td>Dustdar et al. [23]</td>
<td>WS</td>
<td>WS</td>
<td>WS</td>
<td>R</td>
<td>E</td>
<td>S</td>
</tr>
<tr>
<td>Autili et al. [27]</td>
<td>WS</td>
<td>NS</td>
<td>WS</td>
<td>R</td>
<td>E</td>
<td>S</td>
</tr>
<tr>
<td>Cardelli et al. [7]</td>
<td>WS</td>
<td>WS</td>
<td>WS</td>
<td>R</td>
<td>P</td>
<td>G</td>
</tr>
<tr>
<td>Lorenzoli et al. [22]</td>
<td>NS</td>
<td>WS</td>
<td>NS</td>
<td>R</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>He et al. [38]</td>
<td>NS</td>
<td>WS</td>
<td>NS</td>
<td>R</td>
<td>E</td>
<td>S</td>
</tr>
<tr>
<td>Mateescu et al. [39]</td>
<td>NS</td>
<td>S</td>
<td>WS</td>
<td>R</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>Zeng et al. [30]</td>
<td>S</td>
<td>WS</td>
<td>S</td>
<td>P</td>
<td>E</td>
<td>G</td>
</tr>
<tr>
<td>Robinson and Kotonya [21]</td>
<td>NS</td>
<td>S</td>
<td>WS</td>
<td>R</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>Tanaka and Ishida [15]</td>
<td>NS</td>
<td>NS</td>
<td>S</td>
<td>P</td>
<td>E</td>
<td>S</td>
</tr>
<tr>
<td>Siljee et al. [40]</td>
<td>NS</td>
<td>NS</td>
<td>S</td>
<td>R</td>
<td>E</td>
<td>S</td>
</tr>
<tr>
<td>Orients et al. [41]</td>
<td>NS</td>
<td>NS</td>
<td>S</td>
<td>R</td>
<td>E</td>
<td>S</td>
</tr>
<tr>
<td>Wang et al. [35]</td>
<td>NS</td>
<td>NS</td>
<td>S</td>
<td>R</td>
<td>E</td>
<td>S</td>
</tr>
<tr>
<td>Silva et al. [42]</td>
<td>NS</td>
<td>NS</td>
<td>S</td>
<td>R</td>
<td>E</td>
<td>S</td>
</tr>
<tr>
<td>Lin et al. [43]</td>
<td>NS</td>
<td>NS</td>
<td>S</td>
<td>R</td>
<td>E</td>
<td>S</td>
</tr>
<tr>
<td>Ivanovic et al. [44]</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>P</td>
<td>E</td>
<td>S</td>
</tr>
<tr>
<td>Hirschfeld et al. [18]</td>
<td>NS</td>
<td>S</td>
<td>NS</td>
<td>R</td>
<td>E</td>
<td>S</td>
</tr>
<tr>
<td>Toscic et al. [45]</td>
<td>NS</td>
<td>WS</td>
<td>NS</td>
<td>R</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>Maurer et al. [46]</td>
<td>NS</td>
<td>S</td>
<td>NS</td>
<td>R</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>Romero et al. [29]</td>
<td>NS</td>
<td>NS</td>
<td>S</td>
<td>R</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>Li et al. [47]</td>
<td>WS</td>
<td>WS</td>
<td>NS</td>
<td>R</td>
<td>P</td>
<td>S</td>
</tr>
<tr>
<td>Motahari-Nezhad et al. [48]</td>
<td>NS</td>
<td>NS</td>
<td>S</td>
<td>R</td>
<td>E</td>
<td>S</td>
</tr>
<tr>
<td>Cugola et al. [49]</td>
<td>NS</td>
<td>S</td>
<td>NS</td>
<td>R</td>
<td>E</td>
<td>S</td>
</tr>
<tr>
<td>Andre et al. [50]</td>
<td>WS</td>
<td>NS</td>
<td>WS</td>
<td>R</td>
<td>P</td>
<td>G</td>
</tr>
</tbody>
</table>

Key: S = supported; WS = weakly supported; NS = not supported; N/A = not applicable.
approaches that support runtime adaptation are based on rules that
need to conduct it at runtime unlike conventional approaches
anticipation. One way to address the problem is through the
business, application and runtime environment. Validation serves
dynamic, and comparatively resource-constrained conditions, a
reconfigure systems based on fixed decision points. This means
loop against some optimisation objective. This would include
more inclusive validation approach would address this challenge.
product based on the user's product acceptance \[14\]. The most
validation of adaptation decisions.

Such an approach should perform validation for the entire control
challenging concern for validation is uncertainty. This highlights
reconfiguration process, and the new configuration, against the
nature of problems for which different adaptations are suited. Most
validating the reconfiguration process, the executability of the
A typical adaptation process uses a predefined decision model to
select an appropriate adaptation in response to a change trigger. This
relationship is often predefined and stored as a set of adaptation rules. However, the dynamic nature of SOs means
these factors are constantly changing, which makes it difficult to
specify adequate adaptation rules a priori. This is further
complicated by the likelihood of competing for adaptation requests.
This means that rules used to inform adaptation decisions cannot
be static and must constantly evolve to remain relevant. Most
approaches that support runtime adaptation are based on rules that
reconfigure systems based on fixed decision points. This means
that most adaptations in SOs are responses to change rather than
anticipation. One way to address the problem is through the
validation of adaptation decisions.

As mentioned earlier validation refers to building the right
product based on the user's product acceptance \[14\]. The most
challenging concern for validation is uncertainty. This highlights
the need to conduct it at runtime unlike conventional approaches
that focus on conducting it before deployment. Because self-
adaptation targets environments with hard to predict, highly
dynamic, and comparatively resource-constrained conditions, a
more inclusive validation approach would address this challenge.
Such an approach should perform validation for the entire control
loop against some optimisation objective. This would include
validating the reconfiguration process, the executability of the
reconfiguration process, and the new configuration, against the
business, application and runtime environment. Validation serves
two key roles. First, it provides a mechanism for assessing the
effectiveness of an adaptation decision, i.e. how well a
recommended adaptation addresses the concerns for which the
system is reconfigured. Second, it provides us with insights into the
nature of problems for which different adaptations are suited. Most
autonomic systems are underpinned by IBM's Monitoring, Analysis, Planning, and Execution model (MAPE) \[13\]. Fig. 4
illustrates the MAPE-K cycle. MAPE model for autonomic computing intelligent control loop works as follows:

- The monitor function provides the mechanisms that collect, aggregate, filter and report details on adaptation triggers. To
detect the triggers sensors are used. For example, a sensor could
be used to detect the client device used and provide a report. Baresi and Guinea \[28\] describe a service-monitoring approach
as the support for the dynamic selection and execution of
monitoring rules at runtime.
- The analyse function provides the mechanisms that correlate and
model complex situations. To do this, it analyses the report
provided by the monitor and issues an alert if certain threshold
values are reached based on preexisting rules. For most
approaches surveyed, this is a simple process as it addresses
only one type of change. The process however would become
complex where multiple changes have to be monitored. Psaila et
al. \[51\] attribute the difficulty in management of adaptation of
service-oriented applications to the changing interaction and
behaviour patterns that possibly contradict and result in faults
from varying conditions and misbehaviour in the network.

- The plan function provides the mechanisms that construct the
actions needed to achieve goals and objectives. Most approaches
surveyed focus on the identification of a suitable alternative
service as can be seen in the work of Cervantes et al. \[52\]. He et
al. \[38\] however propose the adaptation of web service
composition based on workflow patterns re-composition.
- The execute function provides the mechanisms that control the
execution of a plan with considerations for dynamic updates. It
invokes the adaptation technique. For example, it would call for
the re-orchestration of the service-oriented application to make
use of the suggested workflow pattern.

While the MAPE model is evident in many self-adaptive frameworks for SOA, it lacks a runtime mechanism for supporting
validation. This is also evident in the survey we conducted as most
researchers do not consider validation as part of the adaptation
process. Kephart and Chess \[53\] highlight the importance of
continuously validating an autonomic system to gauge its
effectiveness. A separate survey of current research in validating
SOs reveals that various verification and validation techniques are
used to support adaptation. These include formal methods, model-
based, and machine-learning techniques. The next section discusses
these techniques and some of the challenges faced.

3.5.1 Formal methods: Salehie and Tahvildari \[10\] suggest that
formal methods can be used for verification and validation of self-
adaptive software to ensure its correct functionality and to
understand its behaviour. Weyns \[54\] states that formal methods set
out to show that a system has some desired properties by proving
that a model of that system satisfies those properties. The use of
formal methods can also be seen in the work of Fiadeiro et al. \[55\]
who set out to develop models through which designers can
validate properties of composite services. Assembly and binding
techniques such as the ones provided by Service Component
Architecture can then be used to put together heterogeneous service
components. They define a mathematical model of computation and
an associated logic for SOs which preserves correctness. In
their work, the semantics of service modules are defined after
which they formulate a property of correctness that guarantees
services programmed and assembled (as specified in a module)
provide the business functionality advertised by that module.
However, the model does not take into account shifting user
requirements, the changeability of services and unpredictable
runtime environments that are continuously evolving.

Armando et al. \[56\] propose a platform for the Automated
Validation of Trust and Security of Service-Oriented Architectures.
They emphasise that deploying services in network infrastructures
such as SOAs entail a wide range of trust and security issues.
Modelling and reasoning about these issues are complex because
SOAs use different technology, can interfere with each other and
run on unpredictable environments. They propose the use of a
validator that takes any model of a system and its security goals
and automatically checks whether the system meets its goals under
the assumption that the network is controlled by a Dolev-Yao
intruder (a formal model used to prove properties of interactive
cryptographic protocols). As proof of concept, they formalise ten
application scenarios of SOAs from the e-Business, e-Government
and e-Health application areas. While this work provides some
good insight into the modelling of dynamic aspects of SOs, it is
very closely concerned with validating aspects of security and
trust. It is not easy to port the model to address other quality and
system aspects in SOAs.

Another example of how formal methods can be used is seen in
the work of Arcaini et al. \[13\] who model and validate a
distributed self-adaptive service-oriented application. They design
a traffic monitoring system with a number of intelligent cameras
along a road and apply a formal modelling approach for self-
validation. In their work, a framework for formal modelling,
validating, and verifying distributed self-adaptive systems based on
the multi-agent Abstract State Machines formalism is presented.
They state that formal methods can be used as a rigorous means for
specifying and reasoning about self-adaptive systems’ behaviour, both at design time and at runtime. They, however, note that the over-specification is a challenge with the formal approach due to the rigidity of the formalisms Timed Automata. This challenge can be avoided through separation of concerns where one adaptation concern is handled at a time. Lemos et al. [9] also agree that this approach can be challenging to use at runtime. They state that formal methods can be too expensive to be executed regularly at run-time when the system adapts, due to their time and space complexity.

3.5.2 Model-based approaches: In this approach, models check the behaviour of a self-adaptive system during design and are later used to test the implementation during and after development. Gomaa and Hashimoto [16] use patterns to model how the components that make up an architecture pattern cooperate to change the software configuration at run-time. They propose a model-based run-time adaptation pattern for distributed hierarchical service coordination in SOSSs, in which multiple service coordinators are organised in a distributed hierarchical configuration.

Based on interpretations of UML Models as graphs and graph transformation systems, Baresi et al. [57], posit that the consistency between platform and application can be validated using model based approaches. In order to reason about planned or unanticipated reconfigurations of architectures, they use graph transformation rules to capture the dynamic aspects of architectural styles. As a case study, they make use of the reference architecture for a supply chain management system that involves a consumer component, a retailer service, a warehouse service, a shipping service, and a manufacturer service. Their model of the architectural style supports the architect when deciding whether the style is suitable for his application.

However, many researchers agree that the use of model based techniques on their own is inadequate. Fleuréy et al. [58] combine model driven and aspect oriented techniques when validating dynamic adaptation. Aspect orientation provides modularisation mechanisms to separate the crosscutting concern at the programming level. Model Driven Engineering (MDE) techniques, on the other hand, consider models as the primary development artefact and use them as a basis for obtaining an executable system in different ways. Fleuréy et al. combine both techniques by designing a base model and different variant architecture models at design time that are processed to produce a correct system configuration at runtime. The actual configurations of the application are built at runtime by selecting and composing the appropriate variants. Their work, however, looks at validating adaptation rules at design-time. Calinescu et al. [59] advocate for the use of both modelling techniques and mathematically based techniques to plan the adaptation steps necessary to identify requirement violations at runtime. They, however, point out model learning as a key challenge of their work. In their discussion on the use of models at runtime for self-assurance, Cheng and De Lemos [60] highlight some key challenges with the approach. A key issue in this approach is to keep the run-time models synchronised with the changing system. They recommend the use of probability distribution functions, the attribute value ranges, or using the analysis of historical attribute values. More advanced and predictive models of adaptations are needed for systems that could fail to satisfy their requirements due to side effects of change.

3.5.3 Machine learning: In order to assess the effectiveness of an adaptation decision, a self-adapting system needs to learn. The learning process can yield results that can be used to update the adaptation process with a goal of remaining relevant. Alpaydin [61] defines machine learning as programming computers to optimise a performance criterion using example data or past experience. He further explains that machine learning is used where human expertise does not exist and the solution changes with time. Learning occurs by building models that are good and useful approximations from examples of data provided. To achieve these statistics are used to make inferences from the examples of data provided and efficient algorithms are used to solve optimisation problems as well as represent and evaluate generalised models.

Machine learning algorithms can generally be categorised as supervised, unsupervised and reinforced learning. Supervised learning algorithms make predictions based on a set of examples. Classifiers, decision trees, neural networks, and regression are some examples of supervised learning that can be seen in the work of Hoffert et al. [62]. Unsupervised learning occurs where labelled examples are not available. The goal is to organise the data in some way or to describe its structure. This can mean grouping it into clusters using algorithms such as k-means clustering or expectation-maximisation (EM) clustering. For the unsupervised approach, EM clustering is often considered because it provides better optimisation than distance-based or hard membership algorithms, such as k-means. EM easily accommodates categorical and continuous data fields making it the most effective technique available for proper probabilistic clustering. Skalkowski and Zielinski [63] recommend the dynamic adaptation of services using machine learning. They show how a clustering algorithm can be used to provide automatic recognition of similar system states and grouping them into subsets (called clusters), based on information provided by the monitoring element interface. Reinforced learning provides a method for the machine to quantify its performance in the form of a reward signal. Markov decision processes are used to model the problem to be solved as seen in the work of Jureta et al. [64] and Wang et al. [35]. Experiments with EM clustering, however, show some gaps that are typical of natural data. In order to reinforce unsupervised learning, a supervised learning approach can be used such as neural network classification. Roohi [65] points out that most of the problems that prop up in all the fields of human operations pertain to organising objects or data in different categories or classes. The challenge therefore is to assign an object or data to a class-based on a number of observed attributes (features) related to that object. On the other hand, reinforced learning is based on exploration that relies on a trial and error process. This presents safety challenges in risky application contexts as well as learning challenges where rewards are sparse. According to Cybenko [66] and Hornik et al. [67], artificial neural networks are good classification options as they have been able to approximate any function with good accuracy. Artificial neural networks, being non-linear models, can be used to model any real-world complex process.

Hoffert et al. [62] support the idea of supervised machine learning in dynamic environments in maintaining of QoS. They list possible supervised machine learning techniques to include decision trees, neural networks, and linear logistic regression classifiers that can be trained on existing data to interpolate and extrapolate for new data. Hoffert et al. [64] in a similar study state that parameters such as QoS, deadline, reputation, cost, and user preferences can be used as criteria in learning algorithms. Bayesian probabilities have also been used to express evidence about stakeholders’ satisfaction in terms of degrees of belief. Schumann et al. [68] proposed a validation method to calculate safety regions for adaptive systems around the current state of operation based on a Bayesian statistical approach.

Many experiments have shown that deep neural networks are particularly good with natural data such as speech, vision, or language, which exhibit highly non-linear properties [69]. Najafabadi et al. [70] state that stacking up non-linear feature extractors (as in deep learning) often yield better machine learning results, e.g. improved classification modelling, better quality of generated samples by generative probabilistic models, and the invariant property of data representations. However, they also point out that a downside of deep learning is the requirement for constant memory consumption. Additionally, the slow learning process associated with a deeply layered hierarchy of learning data abstractions and representations from a lower-level layer to a higher-level layer makes it challenging to use at runtime. In addition to the foregoing validation techniques, the nature of involvement, control mechanism and strategy adopted are also important. The next section reviews these three factors.
3.5.4 Involvement – online versus offline validation: Validation can be performed at design-time, runtime or during system maintenance. Traditionally, this has been conducted offline at design-time. However, at this stage validation can only address requirements that were known during development. The shift towards self-adaptive systems called for validation to be performed dynamically at runtime, i.e. online validation. This, however, introduces the challenge of ensuring that the recommended adaptation is timely, right and has adequate system resources available to support it. This section examines some representative involvements.

There have been attempts to use self-test mechanisms at runtime to validate the changes. King et al. [71], recommend that dynamically adaptive behaviour in autonomic software should include rigorous offline and online testing. They propose an in-built test manager in autonomic systems to support this. Zhang et al. [72] propose a runtime model checking approach for the verification of adaptation. Salehie and Tahvildari [10] identify a key challenge posed by the online approach as the presence of several alternatives for adaptable artefacts and parameters in the system. They note that this leads to several paths of execution in different scenarios. Further, the dynamic decision-making approach makes it even more complex. Cardellini et al. [7] present an architecture that allows for the validation of the adaptation decision offline. This can be achieved by collecting statistics based on past adaptation decisions.

Autili et al. [27] propose a model-based solution for self-adapting context-aware services. They provide methodologies to generate adaptable code from UML service models during development. Model-To-Code transformations are performed by means of a code generator offline. They perform both online validations (to generate test cases, before the service execution) by taking into account both the service model and the service code) and offline validation (whilst the service is running and uses the generated test cases). They, however, only give an overall description but no real-world case studies that would validate the whole framework.

The sole use of design time (model based approaches) for self-validation is not adequate. Dusdtar et al. [23] recommend combining both design time and runtime management to build evolvable systems. In their work, model-driven development techniques are first adopted and adapted to support the modelling and design of compliant Web services and processes at design time. They conclude that online and/or offline monitoring components must be introduced as well as tools such as a dashboard to allow human users to observe the system and react to problems and critical situations. However, the goal of autonomic computing is to eliminate human intervention. A self-learning approach could provide one solution.

3.5.5 Control mechanism: Control mechanisms have been at the center of engineering practice for several decades now. The purpose of a controller is to produce a signal that is suitable as input to the controlled plant or process [73]. A key requirement for any self-adaptive system is to make use of control values that tell the system how to adjust. A self-validating system signals the need to take corrective action whenever the output of the adapter deviates from expectations. This deviation is referred to as the tracking error.

Tamura et al. [74] describe feedback control loops as validation that depends on online measurements of past performance from the target system and the adaptation mechanism. They posit that measured outputs are important for making adaptive system quality decisions at runtime. Dusdjar et al. [23] present a solution that incorporates a model-driven compliance support, runtime interaction mining, runtime management of requirements, and control loop architecture for self-validation. They develop a Web service information model to provide a holistic view of past and present requirements associated with services. Then, based on these requirements explicit feedback control techniques are used to perform adaptation strategies.

Feed-forward control techniques, on the other hand, take environmental or external context into account, i.e. the current situation. This can also provide validation information of the adaptation process. Cardozo et al. [75] propose a feed-forward approach to validate the dynamic adaptation of software. Their approach uses a symbolic execution engine to reason about the reachable states of the system, whenever contexts are activated or deactivated. Context activation and deactivation requests are allowed depending on the presence of erroneous states within reachable states. Fredericks et al. [76] point out that traditional testing techniques treat inputs and expected outputs as fixed, static values throughout the testing process. However, requirements specification and the environment can change and thereby cause input and expected output values to no longer be representative test cases. This would make a feed-forward approach inadequate.

3.5.6 Strategy: The strategies used for validation in self-adaptive SOs are either reactive or proactive as shown in Table 3, with reactive strategies being most common. Hielscher et al. [77] describe reactive approaches as those that trigger validation based on monitored events. Consequently, validation occurs after monitoring. In a proactive approach, validation occurs before monitoring and is based on a predictive model trained on historical data. The objective of a proactive approach to validation is to avoid the cost of an unsatisfactory adaptation process. Achieving this, however, is not a simple task as it calls for dealing with uncertainty. A more common approach to validation is the reactive approach, which is easier to implement but may need the adaptation process to iterate several times before an acceptable decision is arrived at. The reactive and proactive adaptation processes discussed in Section 3.3.1 are independent of the validation strategies discussed in this section. For example, a reactive adaptation approach that occurs before triggers is fired can be validated proactively or reactively. It is validated proactively if past validation behaviour is used to determine the acceptable action before a trigger is detected. The validation decision then waits for the trigger. On the other hand, it could be validated reactively after if validation occurs after the trigger is detected. This process waits for the adaptation decision to be arrived at, after which user feedback could be used to validate adaptation. Most of the work reviewed provided implementation details on either adaptation (Table 2) or validation (Table 3) making it difficult to compare the relationship between both strategies.

Fleurey et al. [58] propose a reactive model-based approach to self-validation, which includes invariant properties, and constraints that allow the validation of the adaptation rules at design time. During runtime, the adaptation model is processed to produce a correct system configuration that can be executed. This is achieved through monitoring of the system state and the execution context (such as memory, or CPU usage, or available network bandwidth, or battery level) after which adaptation rules are triggered. Validation then occurs by comparing the woven model with the reference model. It is worth noting that the validation is done because it occurs after monitoring. Similarly, Baresi et al. [57] also propose a reactive approach to validating the adaptation of service oriented systems. However, they recommend runtime validation rather than design time validation. They state that in the dynamic world of SOAs, what is guaranteed at development time may not be true at runtime. They advocate for a reactive approach by arguing that it is virtually impossible to predict all the evolutions and changes that might happen in the services. Once a service is true for the environment. Therefore, monitoring allows them to take notice of infringements of expectation and react to them. Hielscher et al. [77] outline some of these consequences like loss of money, unsatisfied users, and reduced system performance. Autili et al. [27] propose a proactive strategy that explores how to validate extra-functional issues during service development and execution. They state that Bayesian Reliability Models and Queuing Networks can be analyzed at development time to validate the service model characteristics and a decision made available at runtime on how the adaptation of the service will occur for the detected execution context. Validation at design time is therefore performed to generate test cases, before the service execution, by taking into account contextual information and possible changes of the user needs. When a service is invoked, a run-time analysis is
Another proactive approach is presented by Hoffert et al. [62] who point out the difficulty in maintaining the QoS properties (such as reliability and latency) in dynamic environments such as disaster relief operations or power grids. They state that the challenge arises from the slow human response times, and the inefficient resource usage. Their work describes how they use machine-learning techniques to evaluate the effectiveness of an online performance prediction, which allows for the proactive adaptation of the system to the new workload conditions, thereby avoiding SLA violations or inefficient resource usage. Their work describes how they use software performance models to predict the effect of changes and to decide which actions to take.

From our survey, it is evident that very few initiatives use machine-learning techniques to evaluate the effectiveness of an adaptation solution. Further, the work on machine learning techniques does not evaluate several algorithms to improve on past performance. Interestingly, most of the research work on dynamic adaptation does not integrate both runtime and static validation. There is also very little evidence to show that the validation techniques can work for different adaptation triggers or different adaptation techniques. Additionally, the approaches presented in Table 3 show that the majority of the work on validation does not provide adequate details on how adaptation occurs. As a result, they do not appear in Table 2, which focuses on dynamic adaptation. However, they provide adequate details for a self-testing framework. This is expected because self-testing is the focus of their work and as a result, we review them in Table 2.

Only a handful of approaches tackle the issue of validation in self-adaptation. Dustdar et al. [23] describe a self-adaptation technique for managing the runtime integration of diverse requirements arising from interacting services, such as time, performance and cost. They also recommend combining both design time and runtime management to build evolvable systems. They note that current work in adaptive systems provides no integrated support for validating design rules, which affect both the design time and runtime of a system. Although they describe both adaptation and validation they do not provide adequate implementation details on how they work. Cardellini et al. [7] propose an approach to adaptation and validation, however only sketchy details are provided on adaptation. They emphasise the importance of service failure and changes in system quality as triggers for adaptation but only provide a general plan of how their proposed adaptation should work. On the other hand, they provide detailed descriptions of an architecture that calls for the validation of the adaptation decision offline. This can be achieved by collecting statistics based on past adaptation decisions.

4 Conclusion

The paper has discussed the importance of runtime adaptation in SOs and identified the design decisions that must be made when developing these systems. These decisions describe the when, where, and how of adaptation. Specifically, adaptation triggers tell us what causes adaptation, the application context tells us where to adapt, the adaptation models tell us when and how to adapt and validation tells us how effective the adaptation is. We have used these factors to review the current state of runtime adaptation in SOs. Our survey reveals that most of the approaches provide patchy support for the key factors that influence adaptation. Most adaptation approaches are tied to particular application contexts, focus on specific aspects change and are embedded in the application they manage. It is also clear that there is limited empirical evidence to indicate the effectiveness of the approaches reviewed. Lastly, we have provided a possible
solution that integrates and extends the streams of current approaches to support validation. We believe this paper makes a significant contribution towards understanding and addressing a challenging problem.

5 References
