A Self-Learning Framework For Validation of Runtime Adaptation in Service-Oriented Systems

Leah Mutanu

October 2017

Thesis submitted for the degree of Doctor of Philosophy at Lancaster University

School of Computing and Communications

InfoLab21, South Drive

Lancaster University

Lancaster LA1 4WA

United Kingdom

Abstract

Ensuring that service-oriented systems can adapt quickly and effectively to changes in service quality, business needs and their runtime environment is an increasingly important research problem. However, while considerable research has focused on developing runtime adaptation frameworks for service-oriented systems, there has been little work on assessing how effective the adaptations are. Effective adaptation ensures the system remains relevant in a changing environment. One way to address the problem is through validation. Validation allows us to assess how well a recommended adaptation addresses the concerns for which the system is reconfigured and provides us with insights into the nature of problems for which different adaptations are suited. However, the dynamic nature of runtime adaptation and the changeable contexts in which service-oriented systems operate make it difficult to specify appropriate validation mechanisms in advance. This thesis describes a novel consumer-centred approach that uses machine learning to continuously validate and refine runtime adaptation in service-oriented systems, through model-based clustering and deep learning. To evaluate the efficacy of the approach a medium sized health care case study was devised and implemented. The results obtained show that self-validation significantly improves the dynamic adaptation process by autonomously addressing changing user requirements at runtime. Further work in this area can improve the framework by integrating other learning algorithms as well as testing the framework on a larger case study.

Declaration

This thesis is my own work and has not been submitted in any form for the award of a higher degree elsewhere. The work has been carried out under the supervision of Dr. Gerald Kotonya of the school of computing and communications at Lancaster University.

Leah Mutanu

17th October 2017

Acknowledgements

I would like to thank my supervisor Dr. Gerald Kotonya, for accepting to supervise my PhD. His expertises, guidance, support and patience has been vital to the success of this research. I would also like to thank the Software engineering forum within the department for steering and reinforcing this research. Finally I would like to thank family, friends, and colleagues for supporting me through opportunities, encouragement, and funding.

Related Publications

Mutanu, L. & Kotonya, G. (2016). "Consumer-Centred Validation for Runtime Adaptation in Service-Oriented System," *Proceedings of 9th IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, Macau, pp. 16-23.

Mutanu, L. & Kotonya, G. (2017a). "A Self-Learning Approach For Validation of Runtime Adaptation in Service-Oriented Systems," *Journal of Service-Oriented Computing and Applications,* Springer 2017 (accepted for publication)

Mutanu, L. & Kotonya, G. (2017b). "What, Where, When, How and Right of Runtime Adaptation in Service-Oriented Systems," *Proceedings of International Conference on Service-Oriented Computing (ICSOC, 2017)*: *2nd Workshop on Adaptive Service Oriented and Cloud Applications*, Malaga, Spain, Nov 2017.

Contents

List of Figures

List of Tables

Chapter 1

1 Introduction

1.1 Problem Statement

Service-Oriented Architecture (SOA) provides the conceptual framework for realizing service-oriented systems (SOS's) by supporting dynamic composition and reconfiguration of software systems from networked software services (Erl, 2008). Service-Oriented Computing (SOC) is the computing paradigm that utilizes services as fundamental elements for developing applications/solutions. Services are self-describing, platform-agnostic computational elements that support rapid, low-cost composition of distributed applications, (Papazoglou, 2003). They are offered by service providers who procure the service implementations, supply the service descriptions, and provide related technical and business support. The design and deployment of SOS's is underpinned by standard service description languages, messaging and transport protocols. The main motivations for service-oriented architecture (SOA) are agility, flexibility, reuse, ease of integration and reduced development costs (Rosen et al., 2008). However, as more business organizations adopt SOA solutions, the need to ensure that the systems can adapt quickly and effectively to changes in their operating environment becomes an increasingly important research problem. One way to address the problem is by providing support for dynamic adaptation.

Taylor et al. (Taylor et al., 2009) defines dynamic adaptation as the ability of a software system's functionality to be changed at runtime without requiring a system reload or restart. Taylor points out that there is an increasing demand for non-stop systems, as well as a desire to avoid annoying users. However, while there is considerable research interest in runtime adaptation in SOA (Salehie, et al., 2009) (Lemos et al., 2013), very little research is concerned with assessing the effectiveness of the recommended adaptations and how to address situations where adaptations fall short of expectations (Papazoglou et al., 2007).

A growing consensus amongst researchers is that runtime adaptation in SOA should incorporate a *validation* element (Papazoglou et al., 2007) (Cardellini et al., 2009). In their research roadmap for self-adaptive systems, Lemos et al. (Lemos et al., 2013) emphasize the need for feedback control in the life cycle of self-adaptive systems and the need to perform traditional design-time validation and verification at runtime. In another survey, Salehie et al. (Salehie, et al., 2009) note that testing and assurance are probably the least focused phases in the engineering of self-adaptive software. Papazoglou et al. (Papazoglou et al., 2007) echo this view. They note that the bulk of research in adaptive service-oriented systems has focused largely on dynamic compositions.

The nature and quality of adaptation in service-oriented systems is influenced by a number of factors. These include; system changes (i.e. change triggers), the nature of the application and the logical area where it executes (i.e. application context), and the approach used to reconfigure the system in a particular change context (i.e. adaptation model) and the effectiveness of the adaptation (i.e. validation). However, the dynamic nature of service-oriented systems 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 overlapping or conflicting change requests. These changeable factors underscore a number of important points.

First, the rules used to inform adaptation decisions cannot be static and must constantly evolve to remain relevant. Secondly, validation is central to effective adaptation, it allows us to assess how well a recommended adaptation addresses the concerns for which the system is reconfigured and provides us with insights into the nature of problems for which different adaptations are suited. Thirdly, introspection and knowledge of previous adaptations can allow us to proactively find out when adaptations should occur and the optimal adaptations. Proactive adaptation is motivated by the fact that system changes such as service and quality failures can be costly and as such need to be predicted before they occur. Fourthly, change requests are likely to overlap and conflict. The ability to negotiate and resolve conflicting change requests is important for effective adaptation. Lastly, runtime adaptation in service-oriented systems is concerned with varied change requests, diverse application contexts and may require more than one adaptation model to address a problem. It is therefore important that the approach used is pluggable, flexible and supports diversity.

1.2 Key Challenges

A number of research initiatives are investigating effective ways to improve on runtime adaptation in service-oriented systems. These initiatives are however inadequate for addressing the issues identified in Section 1.1 for the following reasons:

- *Static adaptation rules.* Current approaches for supporting runtime adaptation in service-oriented systems 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 (Salehie, et al., 2009) (Lemos et al., 2013). Indeed, Di Nitto et al. (Di Nitto et al., 2008) 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 solutions 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 service-oriented systems offer poor support for validation (Cardellini et al., 2009) (Lemos et al., 2013). Like most autonomic systems, runtime adaptation in service-oriented systems is based on IBM's *Monitoring, Analysis, Planning, and Execution* model (MAPE) (IBM, 2006). However, MAPE does not support validation. The lack of mechanisms for validating adaptation make 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 (Mutanu & Kotonya, 2016). Validating adaptation goes beyond verifying that the adaptation conforms to its operational specification. Validation is concerned with verifying the acceptability of an adaptation (Sommerville, 2016), 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. (Cubo et al, 2008) and Tanaka and Ishida (Tanaka et al, 2008) describe approaches that are concerned with specific application contexts. Swaminathan (Swaminathan et al, 2008) and Cardellini (Cardellini et al, 2009) 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 service-oriented systems are reactive (Gomaa, et al., 2014) (Weyns et al., 2016). 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 adaption 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. Lastly, 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 service oriented systems.

1.3 Objectives

From the gaps identified in section 1.2 several research questions were raised that formed the basis for the research objectives. The key research questions were:

- *How can the effectiveness of current runtime adaptation frameworks be validated?* The growing research interest in dynamic software adaptation meant that there were various adaptation approaches and the need to find out whether these approaches were satisfying consumer needs had not been addressed.
- *What is the best way to address multiple conflicting triggers?* Runtime adaptation is triggered by changes in the application context, which are not mutually exclusive. Further these changes

can give rise to conflicting solutions, which call for a different adaptation approach. In many cases however adaptation approaches focused on single triggers rendering them inadequate in practical situations.

- *Can an adaptation technique be updated at runtime if it is no longer valid?* Changes in the user's environment call for changes in the adaptation rules indicating that they cannot be static but must evolve in order to remain relevant. To achieve the vision of autonomic computing this process cannot be static but should occur at runtime. Machine learning techniques can be used to ensure that the adaptive system has the ability to evolve following changes in the users requirements.
- *How can the increased resource constrains be mitigated?* The addition of runtime validation and self-learning components exert constraints on system resources such as reduced system performance and increased memory requirements. If these constraints are not managed they can counter the benefits of validating runtime adaptation.

From these research questions the objectives of this research were drawn. The general objective of this work is to address the challenges outlined in section 1.2. In particular, the primary goal of this research is to develop and evaluate a pluggable, self-learning validation framework for supporting runtime adaptation in service-oriented systems. The specific research objectives of this work included:

- Provide a runtime solution that supports user-centred validation in self-adaptive service-oriented systems.
- Provide runtime support for detecting and resolving multiple conflicting adaptation requirements.
- Providing self-learning mechanisms to support and improve the predictive accuracy of runtime adaptation.
- Providing strategy for mitigating runtime resource demands resulting from self-learning mechanisms.

1.4 Contributions

The solution proposed by this research is a consumer-centred framework that uses model-based clustering to assess the efficacy of runtime adaptation and to refine adaptation rules. In addition the solution incorporates deep learning to provide offline validation through periodic reviews of online adaptation decisions. The aim is to compliment online validation by improving the long-term accuracy of adaptation decisions.

The first contribution of this thesis is the development of a citizen observatory portal that can be used as an early warning system by providing information on infectious disease outbreaks (Mutanu and Kotonya, 2016). Many developing countries take a reactive approach to risk mitigation, which can have disastrous consequences if the problem is discovered late. This is particularly a major concern with infectious diseases. A proactive approach that detects the spread of such diseases before they are considered an outbreak would reduce the risks associated with disease outbreaks by allowing for better control planning. The case study was selected because it presented a scenario for adaptation that would be ideal for assessing the framework's ability to self-validate and modify adaptation rules at runtime. Because health information needs vary over time the system needs to adapt to identify the users present requirements. Additionally the type of devices owned by the users, the users' financial ability, and the users' external environment also influence the choice of web services selected to supply the information. Pre-defined adaptation rules developed to represent these relationships, need to evolve over time. The evolution is triggered by a set of multiple environmental triggers whose relationship is based on complex behavioral rules that cannot be predefined. The challenge is to ensure that adaptation rules evolved to reflect this change in user interests. The framework achieved this evolution through a user-centred self-learning process that continuously updated the adaptation rules. Whenever a change in the users' needs was detected the system learned and the systems adaptation behavior was observed to change accordingly by providing relevant health information or selecting suitable services.

The second contribution is the design of a novel framework for assessing the current approaches to runtime adaptation in service-oriented systems, together with a comprehensive review of the state of the art in runtime adaptation for service-oriented systems (Mutanu & Kotonya, 2017a). The survey uses five key factors that influence runtime adaptation in service-

oriented systems to examine 29 adaptation approaches intended to support service-oriented systems. The review highlights a number of gaps in relation to how well the factors are supported. Most of the approaches reviewed focused on specific problem areas with specific triggers, and were embedded in the application rather than pluggable. There was generally no support for validation and limited support for proactive adaptation.

The third contribution of this thesis is a pluggable, self-learning validation framework for runtime adaptation in service-oriented systems (Mutanu & Kotonya, 2017b). The adaptation component of the framework is pluggable and independent of the sensor component. This allows it to be ported to different applications and to mine different sensors as the need arises. Subsequently the framework is generic. Because consumer needs are always changing the adaptation component needs to constantly evolve. To address this need the framework incorporates an independent validation component, which evaluates and updates the adaptation component dynamically to keep it relevant.

The final contribution of this thesis is a prototype that was developed to show how the framework could be implemented. The prototype was designed using Java web services and XML-based rule description language, which allowed for the adaptation rules to be defined and annotated. To perform validation the prototype made use of two different machine-learning algorithms and showed how different algorithms can be used to improve on the accuracy and efficiency of self-validation. The prototype was evaluated on a healthcare support system (Namuye & Mutanu, 2014). Data was collected from both simulated and real life environments for testing purposes.

1.5 Thesis Structure

This thesis begins by presenting a background of the research and outlining the problem that inspired the research in Chapter 1. This chapter also identified four specific objectives that the research set out to fulfill and the key contributions to research in this area. Finally the chapter describes the structure of the thesis by explaining the content of each chapter in the thesis report.

Chapter 2 of the thesis provides a brief background of dynamic adaptation in service oriented systems. It draws an analogy between dynamic adaptation of physical systems and software systems and summarizes the dimensions of adaptation. This is followed by a detailed discussion

of the dimensions, which include the adaptation triggers, context, and models. A representative review of literature that informed the dimensions of adaptation is presented.

Chapter 3 of the thesis describes self-validation of autonomic systems based on a representative review of current work in this area. The review revealed that the requirements for validation start with a decision on when to conduct it i.e. involvement analysis. This is followed by the choice of a technique for detecting the problem and analyzing the extend of the problem. Finally a control mechanisms and a validation strategy is employed to outline how validation will occur. The control mechanism is described as either a feedback or a feed forward approach. The strategy is either a reactive or proactive approach. The details of each of these requirements are discussed in the chapter.

Chapter 4 starts with a description of the design and implementation of the consumer-centred self-validating framework for runtime adaptation of service oriented systems. The various components in the framework architecture are described namely the application context, sensor component, adaptation component, and validation component. A description of the implemented prototype follows with the client and server application components outlined. The web server used and knowledge-base developed are discussed. Finally the machine learning tools integrated with the application are explained.

Chapter 5 describes the evaluation of the proposed framework. It begins by giving an overview of the case study used to evaluate the framework. Various experiments set up to test the ability of the framework to answer the research questions outlined in chapter 1 are explained. The experiments accessed how well the framework dealt with multiple conflicting triggers. Further the accuracy of the framework and its impact on the system resources is evaluated. A discussion of the results is also presented.

Chapter 6 provides a conclusion to the thesis by comparing the results of evaluating the framework against the research objectives outlined in section 1.3. The chapter then discusses possible extensions to this research such as the use of alternative validation techniques or the evaluation of the framework on a different case study. Recommendations on the future direction that the work could take are also made. The chapter concludes by providing a reflection on the limitations encountered and lesson learned during the course of this research.

Chapter 2

2 Self-Adaptation Approaches in Service-Oriented Systems

2.1 Factors Affecting Runtime Adaptation

Most of the work on self-adapting software systems takes inspiration from control theory and machine learning, (Filieri et al., 2015). 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). Figure 2.1 shows the control loop for a typical dynamic physical system (Muller et al., 2013). Although it is difficult to anticipate when and how change occurs in software systems, it is possible to control when and how adaptation should react to change. Mining the rich experiences of control engineering to the dynamic adaptation of self-adaptive systems provides great promise for software engineering (Cheng et al., 2009).

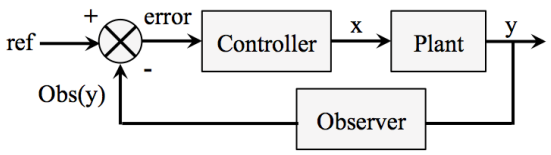


Figure 2.1. Dynamic physical system

Dynamic adaptive systems require information about the running applications as well as control protocols to be able to reconfigure a system. For example, keeping web services up and running for extended periods of time requires collecting of information about the current state of the system, analyzing 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. Figure 2.2(a) shows the control process for a software system equivalent of the physical system shown in Figure 2.1. The *controller* corresponds to the *adaptation* process, which reconfigures the runtime system to address the changing needs in its application context. Figure 2.2 (b) show how the adaptation process can be improved using validation. Validation tracks and assesses each adaptation to ensure that it reflects the user expectations.

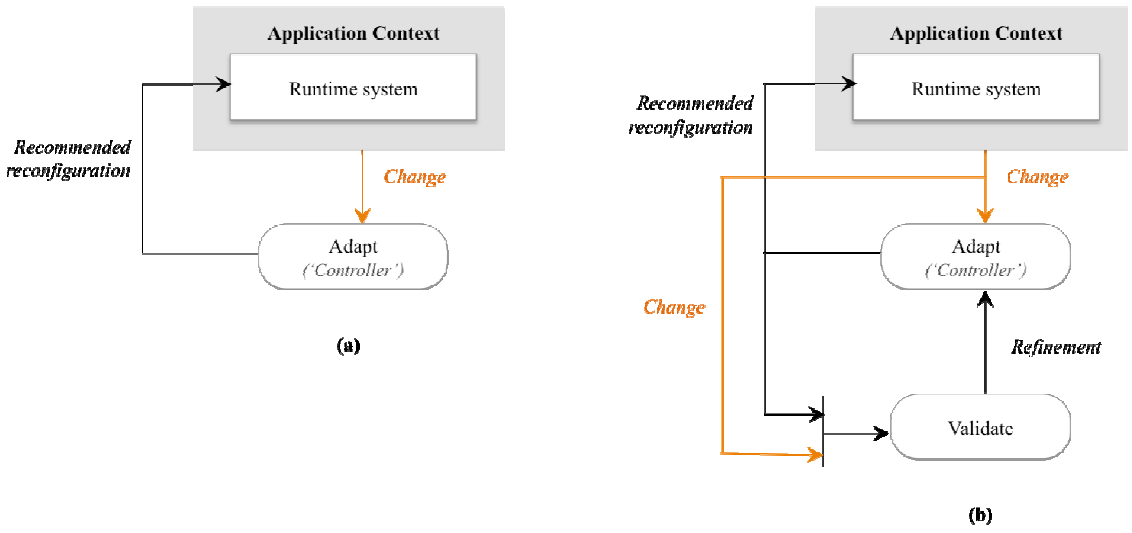


Figure 2.2. Dynamic software system

The nature and quality of adaptation in service-oriented systems is influenced by system changes (i.e. change triggers), the nature of the application and the logical area where it executes (i.e. application context), and 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. This chapter examines the first three factors i.e. *what*, *where, when* and *how*. Chapter 3 will examine importance of validation and how it can be used to improve relevancy and accuracy of adaptation i.e. the *right* of adaptation. It is important to mention that the factors that influence adaptation are not static, but constantly change making it difficult to specify adequate adaptation rules in advance. The problem is further compounded by competing adaptation requests. This means that the rules used to inform adaptation decisions cannot be static and must constantly evolve to remain relevant, which underscores the importance of validation in runtime adaptation.

In their research roadmap for self-adaptive systems, Lemos et al. (Lemos et al, 2013) 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. Key challenges with current approaches include defining models that can represent a wide range of system properties, the need for feedback control loops in the life cycle of self-adaptive systems and self-validation. Lemos et al. also note that there is need to perform traditional design time verification and validation at runtime.

Most approaches that support runtime adaptation are based on rules that reconfigure systems based on fixed decision points (Papazoglou et al., 2007) (Salehie et al., 2009). This also means that most adaptations in service-oriented systems are the result of responses to change rather than anticipated changes. Lastly, most adaptation approaches are embedded as part of the application making it difficult to provide portable and scalable validation.

Hirschfeld et al. (Hirschfeld et al., 2004) suggest that runtime adaptation in service-oriented systems should address the *what*, *when*, and *how* of adaptation (Figure 2.3). 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 operational in 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. However, a recent review of the state of runtime adaptation in service-oriented systems reveal that there are other important factors (dimensions) (Mutanu & Kotonya, 2017). These are examined next.

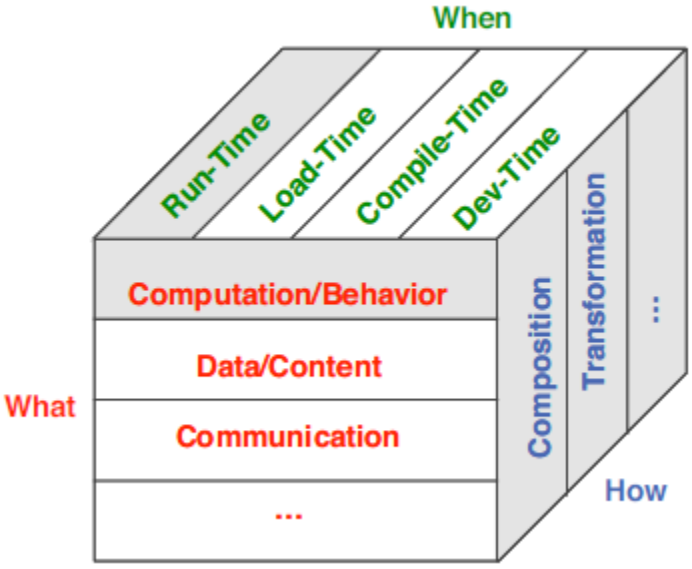


Figure 2.3. Adaptation Dimensions

2.2 Adaptation 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 supported by the systems, service failures, and changes in the system quality and its runtime environment.

- *Business Environment Trigger.* 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. (Zeng et al., 2003) 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. (Cubo et al., 2008) describe an approach that uses changes in the system context and platform to trigger adaptation. Smart cities are advancing towards an instrumented, integrated, and intelligent living space, where Internet of Things (IoT), mobile technologies and next generation networks are expected to play a key role. Alti et al. (Alti et al., 2016) describe a situation where a diabetic person needs to monitor his or her health. He or she uses an intelligent environment, which includes smart devices, sensors (glucose meter, weight scale, video camera) and services. This intelligent environment specifies an ambient system that allows him or her to live safely and to identify urgent situations. For example, the situation of "diabetic coma or too high temperature" is deduced by referencing the average glucose level and the temperature level in the profile.

- *Service Provision Trigger.* Failures in provided services, for example, incompabilities that impact on service composition, network outages and poor service quality, could trigger adaptation. The quality of a service-oriented system 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 et al., (Robinson et al., 2009) describes 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 (Lorenzoli et al., 2006) emphasizes the need for automatic system diagnosis to detect runtime errors. It helps car drivers find the best route in or between towns, by suggesting optimal combinations of transportation according to local situations, such as traffic level, weather conditions and opening hours. They design a self-adaptive application that react to changes in the execution of services, and dynamically prevent mismatches and consequent failures.

- *Runtime Environment Trigger.* Changes in runtime system can also trigger adaptation. Interacting services may impose dependability as well as structural constraints on each other (e.g. performance, availability, cost and interface requirements). Dustdar et al. (Dustdar et al., 2009) describe a self-adaptation technique for managing the runtime integration of diverse requirements arising from interacting services, such as time, performance and cost. Swaminathan et al. (Swaminathan et al., 2008) propose an adaptation approach based on self-healing as a means for addressing runtime system errors. Runtime resource contentions between services in the orchestration platform can result in significant falls in service quality. This emergent quality of service is difficult to anticipate before system composition, as resource demands are often dynamic and influenced by many factors. This is particularly true of service-oriented embedded systems. Newman proposes a resource-aware framework that combines resource monitoring with dynamic service orchestration to provide a runtime architecture that mediates resource contentions in embedded service-oriented systems (Newman et al, 2012).

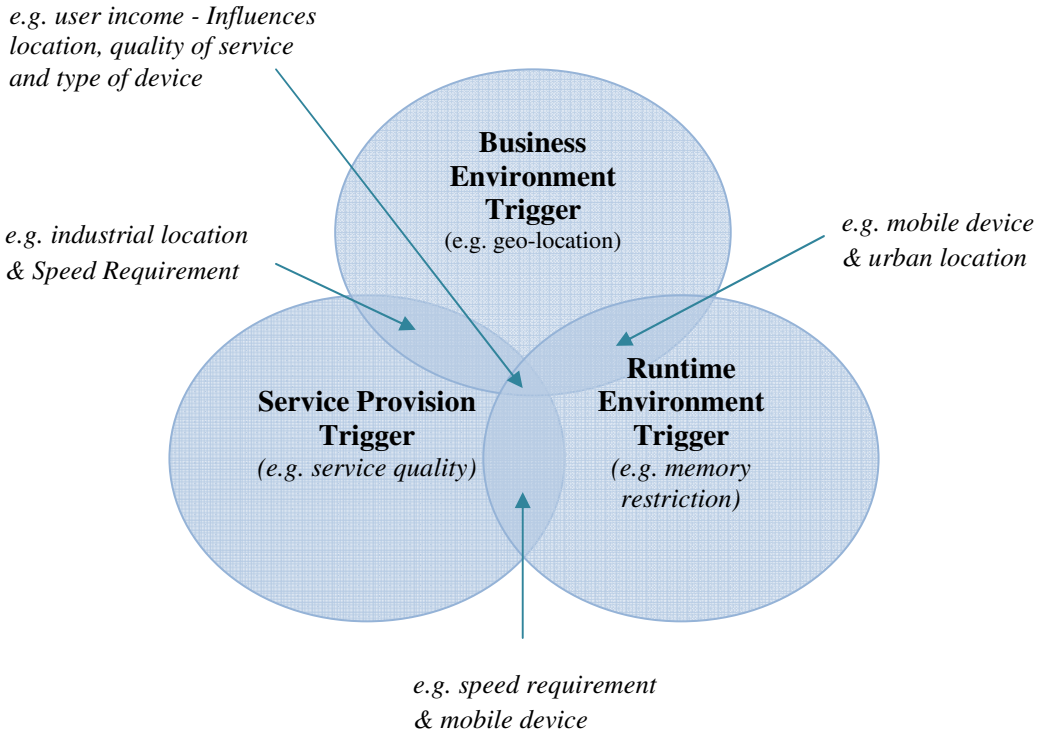


Figure 2.4. Triggers for runtime software adaptation

It is worth noting that adaptation triggers are not mutually exclusive; there is often significant overlap between them as shown in Figure 2.4. For example a user can access health information the service from a location where a certain disease is prevalent. Information about that disease should therefore be availed automatically based on the user's geo-location. A similar scenario would apply where a user is trying to access a travel assistant. The user's environment acts as the source of the trigger. Alternatively a user could be using a mobile device which has limited memory resources as compared to a personal computer. In this case the user's runtime environment then acts at the source of the trigger. A service provision trigger will arise from the quality of service required by the user. For example a user accessing the application to conduct a business transaction such as online banking will require the service to be always available and very reliable. 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 constrains. The geographic location then invokes the runtime and quality of service triggers. A change or failure may be the symptom of an unseen change or failure. Effective adaptation must address the real cause rather than the symptom. Taiani (Taiani, 2009) describes this as a key challenge in adaptive fault tolerant computing. Moyano et al. (Moyano et al., 2013) 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 optimizing 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.

The impact of a change trigger might affect a single service, a combination of services or the system as a whole. Figure 2.5 shows how this might happen. For example, a service failure trigger caused by an SLA violation may be resolved by simply replacing a service, a resource failure trigger might require that the workflow be modified, which might impact on other services. It is also important to mention that both the system level and service level change triggers can occur at the same time and need to be addressed together during the adaptation process.

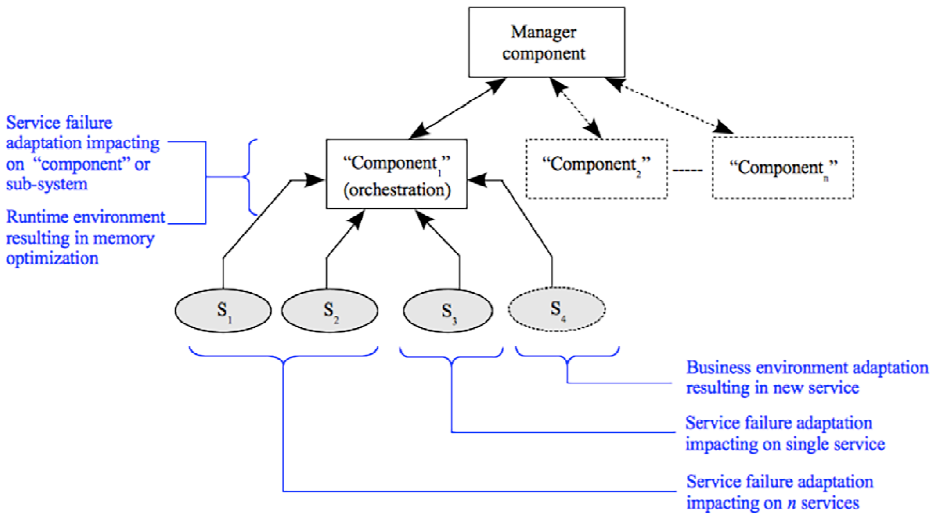


Figure 2.5. How adaptation impacts the system

2.3 Application Context (Where)

Application context defines nature of the application and the logical area *where* it executes and often includes service consumers and service providers (See Figure 4.1). In service-oriented computing (SOC), service providers publish reusable services, and service consumers subscribe them. However, there are potential service mismatch and service fault problems in using 3rd party services, which can affect service quality (La et al., 2011). Cubo et al. (Cubo et al., 2008) discuss the importance of creating adaptive systems sensitive to their application context (i.e. location, identity, time and activity). Cubo's work focuses on in-car software systems. While these factors are important for this application domain, other factors may need to be considered for other domains. Tanaka and Ishida (Tanaka et al., 2008) identify input language and target language as the context for a language translation application.

Most of the approaches surveyed focused on specific adaptation contexts. In their description for the *DigiHome* architecture Romero et al. (Romero et al., 2013) discuss the Integration of multi-scale entities. In the *DigiHome* scenario, they consider several heterogeneous devices (sensors, mobile devices, etc.) 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 as depicted in Figure 2.6. CEP employs complex patterns,

event correlation and abstraction, event hierarchies and relationships between events using causality, membership, and timing. In their work several heterogeneous devices (sensors, mobile devices, etc.) that generate isolated events are used to obtain valuable information and to make decisions accordingly.

A few approaches, including Swaminathan et al. (Swaminathan et al., 2008), Cardellini et al. (Cardellini et al., 2009), and Zeng et al. (Zeng et al., 2003) propose generic application contexts, but they only provide sketchy implementation details. Some approaches promote context variability. For example, Swaminathan et al. describe 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 et al. (Huang et al., 2004) 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. Generic systems and implementation details appear to be conflicting goals from this perspective as illustrated in Table 2.1. For example researchers who provided adequate implementation details such as Ivanovic et al. (Ivanovic et al., 2010) and Fu et al. (Fu et al., 2001) were very specific on the type of system used while those who were more generic in their approach such as Cardellini et al. (Cardellini et al., 2009) and André et al. (André et al., 2010) provided little or no implementation details.

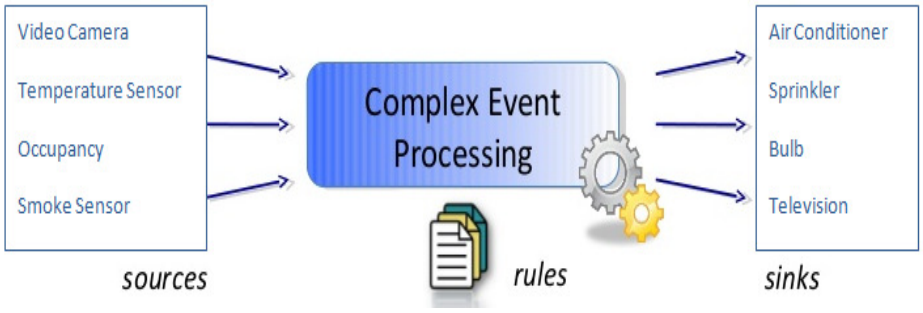


Figure 2.6. Interaction between heterogeneous devices in a smart home (Adapted from Romero et al. (Romero et al., 2013))

2.4 Adaptation Models (When and How)

An adaptation model shows *when* the adaptation process is carried out and *how* the model is implemented in relation to the systems 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. If the requirements are known before the system is running then adaptation can take place at design time, for instance introducing support for a new network communication protocol, or adding new attributes to a data model. However if the requirements are only known after the system has started executing then adaptation will take place at runtime. This is the typical situation in ubiquitous and mobile computing scenarios (Canal et al, 2007). 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. Gilani (Gilani et al., 2007) states that the support for adaptation is required statically, where the applications could be taken offline and adapted, and dynamically where going offline is not an available option. An adaptation model may be implemented to carry out adaptation at design time or at runtime. Further the adaptation module may be implemented as an intrinsic part of the system it manages or as a pluggable component that monitors change and effects re-composition from outside the system. The adaptation could also occur before a change trigger (predictive) or after a change trigger. A review of current adaptation models is discussed next.

2.3.1 Static Adaptation

The idea of self-adaptive systems was initially achieved through static adaptation techniques. Tartler et al. (Tartler et al., 2010) describe static adaptation as a process where the static structure of the program is altered to enable adaptation. In most aspect-oriented programming languages, static adaptation is achieved through the extension of classes by adding new elements like methods, state variables or base classes. In a survey of static adaptive systems (Kell, 2008), an analysis of various static techniques such as object oriented techniques programming techniques, scripting and linking languages is carried out. Some well-known static adaptation techniques in programming languages include:

- *MetaJava adaptation*. Golm et al. (Golm et al., 1997) describe a MetaJava system as an extended Java interpreter that allows structural and behavioral reflection. Reflection in this context refers to the capability of a computational system to "reason about and act upon itself.
- *Load-time adaptation*. Duncan et al. (Duncan et al., 1999) describe this as a code-transformation technique that allows a wide range of changes to programs and languages running on a virtual machine without sacrificing the benefits of portability.
- *Binary component adaptation*. Keller et al. (Keller et al., 1998) describe this as a mechanism that modifies existing components (such as Java class files) to the specific needs of a programmer.

In service-oriented systems, static adaptation has been used to overcome compatibility issues, for example, by authoring multiple content and service versions targeted at a specific delivery environment statically (Lei et al., 2001). While these adaptation techniques have made software more reusable hence more adaptable, they are unsuitable for managing runtime change.

The goal of static adaptation is to provide developers with a framework to build self-adaptive applications. The trust-aware application by Moyano et al. (Moyano et al., 2013) shown in Figure 2.7 is typical of this approach. At design-time, the developer uses the trust API that encapsulates some of the trust-related concepts by means of annotations (e.g. reputation value) from which code is generated for a trust aware self-adaptive system at compile time. This follows the binary component adaptation approach described by Keller et al. (Keller et al., 1998)

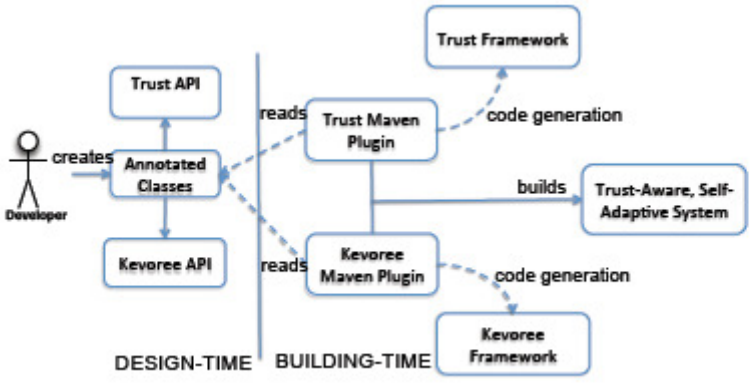


Figure 2.7. Design time adaptation of a trust-aware application. Source (Moyano et al., 2013)

2.3.2 Dynamic Adaptation

As mentioned earlier, the increasing demand for systems that can adapt quickly and effectively to their changing environment has fuelled the need for better runtime adaptation. One of the major challenges identified in the SOA research challenges roadmap (Papazoglou et al., 2007) is the need for services to equip themselves with adaptive capabilities without compromising operational and financial efficiencies. Papazoglou et al. (Papazoglou et al., 2007) identify the key techniques that can be used to achieve runtime adaptation as self-configuring, self-healing, and self-optimizing techniques.

- *Self-Configuring* is the automatic re-composition of services to adapt to changes in the service environment.
- *Self-Optimizing* is the automatic re-composition of services to improve quality of a service.
- *Self-Healing* is the automatic re-composition of services to address a service failure.

Most runtime adaptation approaches reviewed in Table 2.1 focus on only one of the above techniques, which often limits the effectiveness of the approach. The problem of focusing on a single technique is well illustrated with the example of a service-oriented inventory system. A fault detected in a service of the system may require that re-composition takes places to fix the error through self-healing. During re-composition the adaptation process may need to check for an alternative service, which may require adaptation in the system through self-reconfiguration. Lastly, the adaptation process may need to apply self-optimization as part of self-reconfiguration to avoid compromising the system quality.

Psaier et al. (Psaier et al., 2010) depict an architecture for runtime behavior Monitoring and self-adaptation for Service-Oriented Systems Figure 2.8. This feedback loop is known as the MAPE cycle consisting of four essential steps: *monitor, analyze, plan*, and *execute*. Systems that adapt themselves autonomously are enhanced with *sensors and effectors* that allow *network model creation* and *adaptation strategies*. This provides the necessary self-awareness to manage the system autonomously at runtime. Their work involves monitoring of collaboration networks where logs of the monitored behavior are kept at runtime. A *self-adaptation* engine matches this behavior to predefined policies and decides whether to trigger potential adaptation strategies.

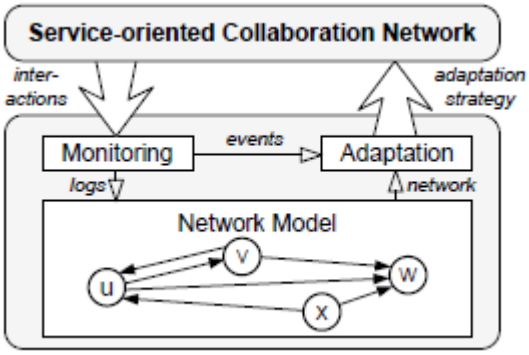


Figure 2.8. Runtime self-adaptation. Source (Psaier et al,. 2010)

An example of Dynamic adaptation model can be seen in the self-healing framework proposed by Li et al. (Li et al., 2013). In the framework an extractor gets the system fault information through a *monitoring* process. This information is then passed on to a case-based reasoner, *which* produces a solution through an *analysis* and *planning* process. Finally the solution is passed on to the business process for *execution*. *A case without a solution* is viewed as a new fault.

It is worth noting that current research initiatives on adaptation of service-oriented systems focus on dynamic rather than static adaptation however the importance of static adaptation cannot be ignored. For any system to be fully adaptable it requires some aspect of dynamic and static adaptation. Parraa et al. (Parraa et al., 2011) propose an approach that unifies adaptation at design and at runtime based on Aspect Oriented Modeling.

2.3.3 Adaptation Strategy - Reactive versus Predictive

Reactive adaptation controls adapt the system according to the users' situation. The system perceives its environment through sensors and reacts to changes as they occur. However, when adaptation occurs as a result of anticipated change, it's called predictive adaptation. 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. Šerbedžija. (Šerbedžija., 2012) describe a reactive approach as a feedback loop while a predictive approach as a feed forward loop.

However, Šerbedžija also notes that it is difficult to create a predictive adaptation approach because the impact and effects of the environment and the controlled actuators cannot be known

without measuring e.g. the effects of changing the lighting in a room must be completely known under all conditions – which may be too farfetched. A more practical approach would therefore be creating a feedback control system that *measures* and *reasons*. Therefore Šerbedžija recommends a predictive approach where satisfying an average user goals are needed, while a reactive approach can be used where a system has to meet more personal user needs through a self-tuning and self-correcting strategy.

In their event-driven quality of service prediction approach, Zeng et al. (Zeng et al., 2003) point out that most adaptation approaches focus on monitoring Quality of Service (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. Tanaka and Ishida (Tanaka et al., 2008) propose a different model of prediction that focuses on predicting the executability of services (i.e. if a message request will cause execution failure). However, they also provide limited detail on the implementation and evaluation of their approach. Wang et al. (Wang et al., 2010) proposes 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 maximizes the expected sum of rewards. While predictive adaptation shows some promise, there is very little research on them and even less information on their evaluation.

Table 2.1 summarizes the findings from the review conducted on 29 approaches to the dynamic adaptation of service-oriented systems. While the survey conducted is by no means exhaustive, it gives an insight into the state of runtime adaptation of service-oriented systems, and specifically on the factors that influence adaptation.

2.3.4 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. (Garlan et al., 2001) depict a pluggable approach where the adaptation

module is plugged on to legacy systems in Figure 2.10. In the figure an external model is used to monitor and modify a system dynamically.

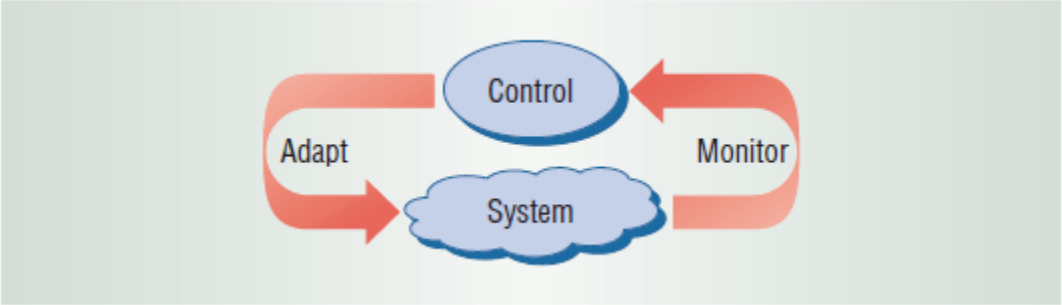


Figure 2.9. A pluggable approach to self-adaptation. Source (Garlan et al., 2001)

Most of the initiatives surveyed adopted an embedded approach. Zeng et al. (Zeng et al., 2003) and Cubo et al. (Cubo et al., 2008) are typical of this approach. However, there is growing acknowledgement amongst researchers that a pluggable approach offers a better engineering solution. Pathan et al. (Pathan et al, 2011) propose a generic approach to context aware modeling through the use a separate component for context reasoning. Garlan et al. (Garlan et al., 2001) state that, the use of external control mechanisms for self-adaptivity is a more effective engineering solution than localizing the solution. A pluggable engine can be analyzed, modified, extended, and reused across different systems.

Table 2.1. Summary of adaptation approaches

	Adaptation Trigger			Adaptation Model			
Approach \ **Factor**	Runtime environment	Service provision	Business environment	Predictive (P) Reactive (R)	Embedded (E) Pluggable (P)	Design-time support (D) Runtime support (R)	**Application Context** (G= Generic S= Specific)
Zeng et al. (2008)	○	○	◒	R	E	R	S
Swaminathan (2008)	○	◒	◒	R	N/A	R	G
Cubo et al. (2008)	○	○	●	R	E	R	S
Huang et al. (2004)	○	○	◒	R	E	R	S
Dustdar et al. (2009)	◒	○	◒	R	E	R	S
Autili et al. (2007)	◒	○	◒	R	E	R	S
Cardellini et al. (2009)	◒	◒	◒	R	P	R	G
Lorenzoli et al. (2009)	○	◒	○	R	P	R	S
He et al. (2008)	○	◒	○	R	E	R	S
Mateescu et al. (2008)	○	○	◒	R	P	R	S
Zeng et al. (2008)	●	◒	●	P	E	R	G
Robinson et al. (2008)	○	●	○	R	P	R	S
Tanaka et al. (2008)	○	○	●	P	E	R	S
Siljee et al. (2005)	○	●	●	R	E	R	S
Orriens et al. (2006)	○	○	●	R	E	R	S
Wang et al. (2010)	○	○	●	R	E	R	S
Sliwa et al. (2010)	○	○	●	R	E	R	S
Lins et al. (2007)	○	○	●	R	E	R	S
Ivanovic et al. (2010)	●	●	●	P	E	R	S
Hussein et al. (2011)	○	◒	○	R	P	R	S
Hirschfield et al. (2004)	○	●	○	R	E	R	S
Tosic et al. (2004)	○	◒	○	R	P	R	S

	Adaptation Trigger			Adaptation Model			
Approach / **Factor**	Runtime environment	Service provision	Business environment	Predictive (P) Reactive (R)	Embedded (E) Pluggable (P)	Design-time support (D) Runtime support (R)	**Application Context** (G= Generic S= Specific)
Maurer et al. (2010)	○	●	○	R	P	R	S
Romero et al. (2013)	○	○	●	R	P	R	S
Li et al. (2013)	◒	◒	○	R	P	R	S
Newman et al. (2012)	●	○	○	R	P	R	S
Motahari-Nezhad et al. (2012)	○	○	●	R	E	R	S
Cugola et al. (2012)	○	●	○	R	E	R	S
Andre et al. (2010)	◒	◒	◒	R	P	R	G

Key

● - Supported

◒ - Weakly Supported

○ - Not Supported N/A- Not Available

Most of the approaches surveyed offer poor support for diversity (i.e. their support is closely tied to specific applications and contexts). Zeng (Zeng et al., 2003) describe an adaptation approach for supporting business change in the automotive industry. The rules for managing the adaptation are all drawn from the industry. Similarly, Cubo (Cubo et al., 2008) describes an adaptive service composition approach for rent-a-car system. Of the approaches reviewed only Swaminathan (Swaminathan et al., 2008), Cardellini (Cardellini et al., 2009), and Zeng (Zeng et al., 2008) propose a generic approach to adaptation. However, in most cases only sketchy details are provided. Swaminathan (Swaminathan et al., 2008) and Cardellini (Cardellini et al., 2009), only provide a general plan of how their proposed adaptation should work, but no details where they have been evaluation or indeed their implementation. There is no evidence that the approaches support diversity. Zeng (Zeng et al., 2008) proposes a generic model that uses prior

knowledge to anticipate change and trigger adaptation. However, the approach has only been evaluated on a limited case study.

Most the approaches surveyed have limited support for system quality and service fault triggers. However they provide strong support for business environment triggers. Of the approaches surveyed only Zeng et al. (Zeng et al., 2008) provides a detailed discussion of the adaptation techniques used to address quality of service issues that arise from interacting services (i.e. system concerns). They provide a detailed discussion of the prediction of quality of service. Dustdar (Dustdar et al., 2009), Autili (Autili et al., 2007), and Cardellini (Cardellini et al., 2009), discuss the importance of service failure and changes in system quality as triggers for adaptation, but give little detail of how their approaches support these. Cardellini (Cardellini et al., 2009), Lorenzoli (Lorenzoli et al., 2006), He (He et al., 2008), Zeng (Zeng et al., 2008) and Valetto (Valetto, 2002) discuss how service failure is supported in their different approaches using limited examples. Robinson et al. (Robinson et al., 2008) and Siljee et al. (Siljee et al., 2005) provide more detailed discussions and more involved case studies.

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 (Cardellini et al., 2009), (Lorenzoli et al., 2006),(Nallur et al., 2009), (Mateescu et al., 2008), (Tanaka et al., 2008), and (Lins et al., 2007). The approach proposed by Swaminathan (Swaminathan et al., 2008) does not provide adequate implementation details, so it is unclear how it is implemented (i.e. as a pluggable or embedded solution).

2.5 Summary

This chapter has examined factors that influence the dynamic adaptation of Service Oriented Systems with a view of shedding light on what an effective adaptation technique calls for. It sought to identify what causes adaptation, where adaptation takes place, and how adaptation is modeled. To answer these questions an extensive survey was conducted on current research in the system adaptation of service-oriented systems.

From the survey it became apparent that there are three different types of triggers that initialize the software adaptation process based on their origin. These triggers originate from the business

environment, the runtime environment, or the service itself. Business triggers included triggers external to the application such as the geographic location, while triggers from the service itself such as failure affected the quality of the service. Runtime triggers originated from the service platform and included triggers such as memory or CPU resource constraints. It is noted that triggers are not mutually exclusive but often overlap. Majority of the work addressed the external environment triggers ignoring the runtime and internal environment. An effective self-adaptive application therefore should be able to deal with multiple overlapping triggers that may sometimes conflict. Very little research work focused on providing a generic solution that can work in any application context. Most of the work that gave adequate implementation details focused on a specific application context. The research that gave generic solutions provided little or no implementation details.

The survey also revealed that different software adaptation models exist. Static adaptation models conducted adaptation during the design or maintenance of the software while dynamic adaptation models performed adaptation at runtime. Only a few of the dynamic adaptation models from the survey incorporated elements of static adaptation, which can enhance dynamic adaptation. Embedded adaptation models implemented the adaptation module as part of the application while pluggable models implemented it as a separate model. A pluggable model presented the benefit of portability for a more generic approach. A predictive adaptation model called for adaptation before the adaptation trigger was detected by the application while a reactive model invoked adaptation only after the adaptation trigger was detected. The predictive approach called for learning and would be ideal for an early warning system where disasters risks needed to be avoided. Very few of the adaptation approaches reviewed were based on pluggable or predictive schemes.

The findings of this survey can be summarized as follows:

- Triggers for software adaptation originate from the users business environment, the runtime environment, or the service provision environment. Most of the research focused on business environment triggers
- Most of the work on dynamic adaptation of service oriented systems that provided implementation details focused on specific application context rather that presenting a generic solution that can apply on any application context.

- Solutions presented in existing literature were either pluggable or embedded with the majority focusing on pluggable approaches. This limited their portability.
- Finally dynamic adaptation is either reactive or predictive. Most of the research is reactive with adaptation taking place after triggers have occurred. This made it difficult to prevent catastrophic results.

Given the dynamic nature of the environment that applications run, adaptation techniques must constantly be evaluated to ascertain their effectiveness over time. Simply adapting an application in accordance to user requirements is not enough; the final adaptation decision has therefore to be validated for user acceptance.

Chapter 3

3 Software System Validation

3.1 Validation in Autonomic Systems

Sommerville (Sommerville, 2016) defines Software Validation as building the right product based on the users product acceptance. In their research roadmap for self-adaptive systems, Lemos et al. (Lemos et al., 2013) emphasize the need for feedback control in the life cycle of self-adaptive systems and the need to perform traditional design-time validation and verification at runtime. In another survey, Salehie et al. (Salehi et al., 2009) note that testing and assurance are probably the least focused phases in the engineering of self-adaptive software. Papazoglou et al. (Papazoglou et al., 2007) echo this view.

Most autonomic systems are underpinned by the IBM Monitoring, Analysis, Planning, and Execution model (MAPE). Figure 3.1 illustrates the MAPE cycle. However, while the model is evident in many self-adaptive frameworks for service-oriented systems, it does not provide support for validating runtime adaptation.

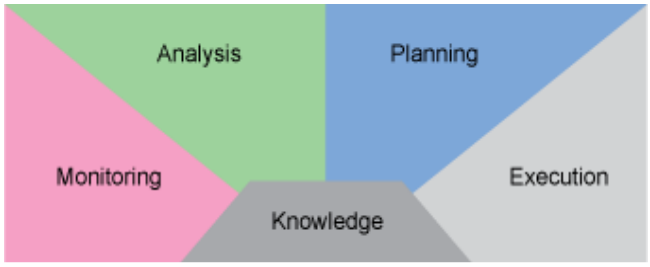


Figure 3.1 MAPE Architectural Model for Autonomic Computing

Eze et al. (Eze et al., 2011) highlight the importance of continuously validating an autonomic system to gauge its effectiveness. Eze suggests that validating the decision making process of autonomic systems is key, to achieving the full vision of Autonomic Computing. Weyns et al. (Weyns et al., 2016) describe a scenario where a traffic jam spans the viewing range of multiple cameras. If one camera fails, a self-healing system can instigate adaptation by suggesting a second camera that can serve the purpose. Analysis of the system operation over time can determine how effective the adaptation was and subsequently inform future adaptation strategies. Schumann et al. (Schumann et al., 2005) have looked at how the concept of self-validation can be applied to

adaptive aircraft controllers. In their work they explain how an estimate of the expected aircraft handling quality provides a kind of feedback that can be directly assessed by the pilot. The validation information from the pilot can be used to refine and improve the adaptability of the aircraft. Cheng et al. (Cheng et al., 2009) point out that while the goal of self-validation is simple, achieving it is not. Traditional verification and validation methods, static or dynamic, rely on stable descriptions of software models and properties. However, runtime system properties, system functionality, and the business needs served are dynamic and prone to change, making them difficult to predict in advance. For verification and validation methods to be effective they need to align themselves with business changing goals and requirements as well as variable system functionality. A partial example of this can be seen in the work of Calinescu et al. (Calinescu et al., 2011) who propose formal verification techniques such as quantitative verification and model checking to support the multiple stages of the MAPE autonomic computing loop. Their work focuses on using a mathematically-based technique for establishing the correctness, performance and reliability of systems that exhibit stochastic behavior. Their assertion that software must adapt continuously, to respond to changes in application objectives and in the environment in which it is embedded is a key underpinning of my research. Their work however looks at software verification, which is concerned with how well the system conforms to its specification (Sommerville, 2016). Software verification assesses whether we are building the product correctly. The primary focus of my research is validation, which is concerned with checking that the system meets the user's actual needs.

3.2 Validation in Service-oriented Systems

The next sections review current approaches that support verification and validation in self-adaptive service-oriented systems. For each approach, the technique used, involvement (i.e. offline or online), the control mechanism employed, the primary focus of the approach (i.e. verification or validation), and the strategy adopted (i.e. reactive or proactive) are examined. The strengths and limitations of each approach are discussed and an overall summary provided at the end in Table 3.1.

3.1.1 Techniques

A survey of current research in adaptive service-oriented systems 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 a number of techniques and some of the challenges faced.

(i) Formal methods

Salehie et al. (Salehi et al., 2009) suggest that formal methods can be used for verification and validation of self-adaptive software to ensure its correct functionality, and to understand its behavior. Weyns et al. (Weyns et al., 2012) state 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. However, Lemos et al. (Lemos et al., 2013) 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.

Arcaini et al. (Arcaini et al., 2015) 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 modeling approach for self-validation. Each camera is endowed with a data processing unit and a communication unit to interact with other cameras. Cameras have to collaborate to observe larger phenomena such as traffic jams and aggregate the monitored data. The need for self-adaptation is triggered by changes in the surrounding environment, namely the congestion and by the failures of other cameras. They note that over-specification is a challenge with the formal approach due to the rigidity of the formalisms Timed Automata. They avoided over-specification through separation of concerns (by focusing on one adaptation activity at a time), which can be a challenge given that adaptation activities are not mutually exclusive.

While promising productivity gain during development, service oriented systems pose a significant verification challenge on how to guarantee correctness of the complex composite services. Tarasyuk et al. (Tarasyuk et al., 2012) demonstrated how to build complex service oriented systems by refinement in Event-B. They however noted that they could not evaluate whether the designed fault tolerant mechanisms were appropriate enough to meet the desired quality of service. Therefore they proposed an approach of building a dynamic service architecture (an Event-B model) that formally represents the service orchestration. They do this by

refining the set of requirements that allow them to ensure that the modeled service architecture faithfully represents the dynamic service behavior. However, it is important to mention that their focus is on verification rather validation.

The use of formal methods can also be seen in the work of Fiadeiro et al. (Fiadeiro et al., 2011) 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 service-oriented systems which preserves correctness. In their work semantics of service modules are defined after which they formulate a property of correctness that guarantees that 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 continously evolving.

Armando et al. (Armando et al., 2012) propose a platform for the *Automated Validation of Trust and Security of Service-Oriented Architectures*. They emphasize that deploying services in network infrastructures such as SOAs entails a wide range of trust and security issues. Modeling and reasoning about these issues is 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 formalize ten application scenarios of SOAs from the e-Business, e-Government and e-Health application areas. For these application scenarios they write specifications that address different security aspects and the results show that formal validation technologies can have a decisive impact for the trust and security of SOAs and the Internet of Services. While this work provides some good insight into the modeling of dynamic aspects of service-oriented systems, 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.

(ii) Model-based

In this approach models check the behavior of a self-adaptive system during design and are later used to test the implementation during and after development. Gomaa et al. (Gomaa et al., 2014) 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 service-oriented systems, in which multiple service coordinators are organized in a distributed hierarchical configuration. Their work extends research in SOA systems by validating dynamic software adaptation pattern for distributed hierarchical service coordination. The model makes use of a validation component to check if the adaptation took place as expected i.e. transitioning from *Processing* to *Passive* to *Quiescent* states and then back to *Processing* state, during adaptation. They apply various change management scenarios to validate their model. Their work however does not explain what happens when an adaptation model does not behave as expected. User environments differ and as a result an adaptation model needs to be adaptive in nature to cater for changes in user requirement.

Based on formal interpretations of UML Models as graphs and graph transformation systems, Baresi et al. (Baresi et al., 2003), 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. The model is used to reason about possible reconfigurations results in order to check if all required configurations of the application are reachable.

Recent trends like virtualization and Cloud Computing come at the cost of increased system complexity and dynamics, making it challenging to provide Quality-of-Service (QoS) guarantees (e.g. availability and performance) in the face of highly variable application workloads (Huber et al., 2011). To address this challenge Huber et al. propose an approach that calls for the ability to predict at run-time how the performance of running applications would be affected if service

workloads change. They make use of a model based approach to self-adaptive resource allocation in virtualized environments based on online architecture-level performance models. This way they proactively adapt the system to the new workload conditions avoiding SLA violations or inefficient resource usage. A case study of a car dealer that places a large order with an automobile manufacturer is used to test the developed model. The results demonstrate that architecture-level performance models can be used effectively at runtime to validate self-adaptiveness.

Fleurey et al. (Fleurey et al., 2008) combine model driven and aspect oriented techniques when validating dynamic adaptation. They posit that a model driven approach avoids enumerating all possible configurations statically. A base model which contains the common functionalities and a set of variant models which capture the variability of the adaptive application is used. The actual configurations of the application are built at runtime by selecting and composing the appropriate variants. They simulate changes to test their validation component. Their work however looks at validating adaptation rules at design-time. Calinescu et al. (Calinescu et al., 2011) advocate for the use of both modeling 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 et al. (Cheng et al., 2009) highlight some key challenges with the approach. A key issue in this approach is to keep the run-time models synchronized 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 adaptation are needed for systems that could fail to satisfy their requirements due to side effects of change.

(iii) 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 (Alpaydin, 2010) defines Machine learning as programming computers to optimize 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 this statistics are used to make inferences from the

examples of data provided and efficient algorithms are used to solve optimization problems as well as represent and evaluate generalized models.

Machine learning algorithms essentially fall under two groups categorized based on the desired outcome of the algorithm according to Ayodele (Ayodele, 2010).

- *Supervised 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
- *Unsupervised learning* - Here labeled examples are not available. The goal is to organize the data in some way or to describe its structure. This can mean grouping it into clusters using algorithms such as k-mean clustering or Expectation-maximization (EM) clustering.

This research embedded a machine learning element in the framework to assess and update the adaptation decisions. Experiments are conducted that involve both a supervised and unsupervised approach to machine learning.

For the unsupervised approach EM clustering is considered because it provides better optimization 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. Decision logs record change triggers received and the adaptation decision made. If the decision made is inadequate, this is detected as the system almost immediately triggers another request for adaptation. EM clustering is performed in two steps (Kim et al., 2012):

- *Expectation* – Firstly the initial mixture model parameter estimates are input. A single iteration provides new parameter *estimates* that do not decrease the log likelihood of the model. The expected value of the log likelihood function, is given as:

$$Q(\theta|\theta^{(t)}) = E_{z|x,\theta^{(t)}} [log\ L(\theta; X; Z], \quad (3.1)$$

Where Q is the conditional distribution of Z given X under the current estimate of $\theta^{(t)}$.

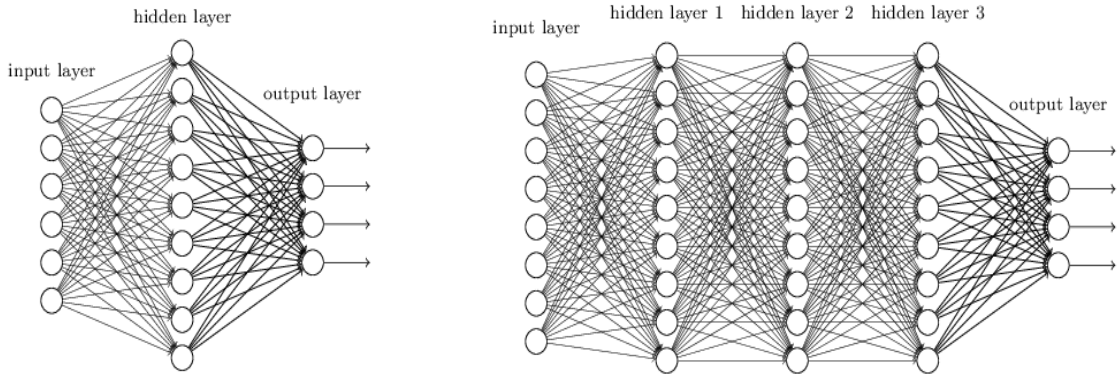
- *Maximization*– Secondly the process is repeated until the log likelihood of the mixture model, at the previous iteration, is sufficiently close (*Maximised*) to that of the current model. The parameter that maximizes this quantity is obtained as:

$$\theta^{(t+1)} = arg_{\theta} max\ Q(\theta|\theta^{(t)}) \quad (3.2)$$

Clustering algorithms analyze the data to find groups of maximum commonality. For example, if the framework establishes that at time t_n adaptation technique A_1 was the acceptable decision for adaptation trigger X. Then at time $t_{n+1}$the acceptable adaptation technique for the same trigger was A_2, the framework changes the adaptation rules accordingly. In this case decision logs act as adaptation triggers.

Experiments with EM clustering however show some gaps that are typical of natural data. This is the motivation behind using alternative algorithms that improve the machine learning process. In order to reinforce unsupervised learning, a supervised learning approach can be used such as neural network classification. Roohi (Roohi, 2013) point out that most of the problems that prop up in all the fields of human operations pertain to organizing objects or data in different categories or classes. The challenge therefore is to assign an object or data item to a class based on a number of observed attributes (features) related to that object. According to Cybenko (Cybenko, 1989) and Hornik et al. (Hornik et al., 1989) artificial neural networks are a good classification option as they have been able to approximate any function with good accuracy. Artificial neural networks, being nonlinear models, can be used to model any real world complex process. While neural networks have been around for a long time, having been developed in the 1950s, they have only become more successful in recent years due to the availability of inexpensive, parallel hardware (Graphic Processing Units and computer clusters) and massive amounts of data.

Single layer neural networks, sometimes referred to as shallow networks, were the first approach to neural networks. They comprised of a single hidden layer. Deep learning artificial neural networks (*ANN*) on the other hand have more than one hidden layer as shown in Figure 3.2. Bengio et al. (Bengio et al., 2009) describes Deep architectures as comprising of multiple levels of non-linear operations, such as in neural nets with many hidden layers or in complicated propositional formulae re-using many sub-formulae. According to Najafabadi et al. (Najafabadi et al., 2015) the hierarchical learning architecture of Deep Learning algorithms is motivated by artificial intelligence emulating the deep layered learning process of the primary sensorial areas of the neocortex in the human brain. It automatically extracts features and abstractions from the underlying data.



Shallow Neural Network Deep Neural Network

Figure 3.2 Artificial Neural Network Architecture

In machine learning a classification problem is one that calls for assigning a predefined class to a given input. Multilayer *ANN* using the back propagation algorithm have been used to solve classification problems with high accuracy. In order to train the neural network using this algorithm the dataset has to be normalized for the values to range from *0* to *1* (Arcaini et al, 2015). This research uses the feature scaling formula shown below for normalization. Where x is the value that should be normalized, x_n is the normalized value, x_{min} is the minimum value of x, and x_{max} is the maximum value of x.

$$x_n = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{3.3}$$

A key concept underlying *deep learning* methods is the use of distributed representations of the data, in which a large number of possible configurations of the abstract features of input data are feasible (Najafabadi et al, 2015). The training set has therefore to be carefully selected to allow for a compact representation of each sample, leading to a richer generalization. For example we can combine environmental and business triggers as the input features for the neural network – i.e. memory capacity, processor speed, device type, and geo-location. A multi-layer A*NN* architecture is made up of an input layer, one or more intermediate or hidden layers, and an output layer as illustrated in Figure 3.2. Each layer is made up of nodes called neurons with no connections between neurons in the same layer. Because a multilayer A*NN* is made up of several nodes, several inputs (x_n) and weights (w_n) exist. The final output is calculated by summing all the inputs multiplied by their weights plus the bias (b), and then run through a limiter. This can be expressed as:

$$Net = \sum_{i=0}^{n} x_i w_i + b \qquad \text{where} \qquad f(x) = \{\frac{x \le 0 \rightarrow 0}{x \ge 0 \rightarrow 1}\} \qquad (3.4)$$

The action of selecting the correct weights is the process that results in the learning rules/algorithms (referred to as adaptation rules in this research). The approach used in the framework is based on supervised learning where the output for a set of inputs is provided to train the network. The error-correction approach, that focuses on the fact that the output perceived in the training process does not always correspond to the desired output, is used. As a result the method calculates the error, modifying the weights to progressively reduce the error. Initially the weights are set to random small numbers and progressively updated during training based on the calculated error. This is the basis of the back propagation-learning algorithm. Training the perceptron calls for the modification of its weights.

Many experiments have shown that deep neural networks are particularly good with natural data such as speech, vision, or language, which exhibit highly nonlinear properties, (Le et al., 2015). Najafabadi et al. (Najafabadi et al., 2015) suggest one of the key benefits of Deep Learning as the analysis and learning of massive amounts of unsupervised data, making it a valuable tool for Big Data Analytics where raw data is largely unlabeled and un-categorized. They state that stacking up non-linear feature extractors (as in Deep Learning) often yield better machine learning results, e.g., improved classification modeling, better quality of generated samples by generative probabilistic models, and the invariant property of data representations.

The use of deep learning in adaptation is not a new concept. Zhou et al. (Zhou et al., 2012) for instance describe how a Deep Learning algorithm can be used for incremental feature learning on very large datasets, employing de-noising auto encoders. Calandra et al. (Calandra et al., 2012) introduce adaptive deep belief networks which demonstrates how Deep Learning can be generalized to learn from online non-stationary and streaming data. Their study exploits the generative property of deep belief networks to mimic the samples from the original data, where these samples and the new observed samples are used to learn the new deep belief network which has adapted to the newly observed data. Najafabadi et al. (Najafabadi et al., 2015) however point out that a downside of adaptive deep belief network is the requirement for constant memory consumption. They state that despite the potential benefits of deep learning, one of the challenging aspects of using deep learning in Big Data Analytics is dealing with streaming and

fast-moving input data. They emphasize the need to adapt Deep Learning to handle streaming data, i.e. large amounts of continuous input data on the fly. Additionally the slow learning process associated with a deep 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. Until such a point where the current infrastructure can support it, it remains a suitable approach for static learning. Deep learning can therefore be used as a static self-learning that can be used to compliment and improve the dynamic self-validation process of adaptation.

(iv) Machine-Learning in Service-Oriented System Adaptation

In order to accommodate the changing user expectations, a number of service-oriented systems have incorporated machine learning as part of adaptation. The works of King et al. (King et al., 2007), Skałkowski et al. (Skałkowski et al., 2013), Canfora et al. (Canfora et al., 2006), Hoffert et al. (Hoffert et al., 2009) and Hielscher et al. (Hielscher et al., 2008) are typical of this technique.

In their work, King et al. (King et al., 2007) emphasize the need to be sure that new errors have not been introduced after adaptation and that the final product behaves as intended. They propose a self-testing framework for autonomic computing systems that dynamically validates change requests through regression testing. To test their framework they develop a prototype of an autonomic container implemented as a distributed system. It provides data storage services to remote users, while performing dynamic self-configuration, self-protection, and self-testing. A mutation testing technique was used to evaluate the prototype with respect to its ability to detect faulty change requests for managed resources.

Skałkowski et al. (Skałkowski et al., 2013), 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. Canfora et al. (Canfora et al., 2006), suggest regression testing to check the evolution of service–oriented systems, since integrators are out of control of the service being used. When a service evolves, its functionality or QoS can vary, impacting over the systems using it. For instance the response time can be larger than the one experienced when the service was acquired and the SLA negotiated. They state that a key challenge with testing of quality of Service is that services run on the (foreign) infrastructures of providers and payment may be on a per-use basis, which makes stress testing prohibitively costly.

Monitoring data from past execution could help reducing the cost of testing by minimizing the needs for actual calls to services.

Hoffert et al. (Hoffert et al., 2009) supports the idea of machine learning in dynamic environments in maintaining of Quality of Service. They list possible 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. They investigate a decision tree and show how the algorithm attempts to create a tree from a set of decisions that can be used to accurately classify a new example. Hielscher et al., (Hielscher et al., 2008) propose a proactive approach to self-adaptation for service-based applications that makes use of regression testing. They however point out that the need to investigate the possible impact of online testing on the performance of the application. Although they only look at a proactive approach they state that proactive and reactive adaptation may work together in an integrated dynamic adaptation framework.

Jureta et al. (Jureta et al., 2007) in a similar study state that parameters such as quality of service, 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. Literature on engineering adaptive mechanisms for flight control systems inspires this according to Tamura et al. (Tamura et al., 2013). Schumann and Gupta (Schumann et al., 2005) proposed a validation method to calculate safety regions for adaptive systems around the current state of operation based on a Bayesian statistical approach. With this approach, they provide a confidence measurement on the probable accuracy of the system's model under a particular situation.

A Self-* Overload Control (SOC) algorithm that self-configures a dynamic constraint on the rate of incoming new sessions in order to guarantee the fulfillment of the quality requirements specified in a Service Level Agreement (SLA) is proposed by Bartolini et al. (Bartolini et al., 2009). It is based on a measurement activity that makes the system capable of *self-learning* and self-configuring in the case of rapidly changing traffic scenarios, dynamic resource provisioning or server faults. They conducted extensive simulations and showed how the system self-protects from overload, meeting SLA requirements even under intense workload variations. To enable the application to self-adapt without any prior assumptions about incoming traffic, they incorporate a

self-learning activity. The self-learning activity stores and updates information regarding the relationship between the observed traffic rate and the corresponding response time based on a fixed threshold policy. When incoming traffic is stable the policy works in normal mode, in which performance controls are paced at a regular rate. The policy switches to flash crowd management mode as soon as a rapid surge in demand is detected. They provide extensive results showing how their proposal permits a fast reaction to sudden changes in traffic intensity, and gives high system responsiveness.

From the survey of self-validating techniques conducted it is evident that very few researchers use machine-learning techniques to evaluate the effectiveness of an adaptation solution. The work on machine learning techniques does not evaluate several algorithms to improve on accuracy. Most of the research work on dynamic adaptation does not integrate both runtime and static validation. Further there is little evidence that it can work for different adaption triggers or different adaptation techniques.

3.1.2 Involvement – Online vs. 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 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. (King et al., 2007), recommend that dynamically adaptive behavior in autonomic software should include rigorous off-line and on-line testing. They propose an in-built test manager in autonomic systems to support this. Zhang et al. (Zhang et al., 2009) propose a run-time model checking approach for the verification of adaptation. Salehie et al. (Salehi et al., 2009) identifies a key challenge posed by the online approach as the presence of several alternatives for adaptable artifacts 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. (Cardellini et al., 2009) present an architecture that calls for the validation of the

adaptation decision off-line. This can be achieved by collecting statistics based on past adaptation decisions.

Autili et al. (Autili et al., 2007) 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 *off-line 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.

To assist existing Requirement Engineering methodologies Jureta et al. (Jureta et al., 2007) propose a Dynamic Requirements Adaptation Method which updates a requirements specification at runtime to reflect change due to adaptability and openness. In order to validate system behavior their solution determines how extensive the initial specification ought to be and what parts thereof are to be updated at runtime to reflect system adaptation. Online validation enables the specification of requirements that may vary at runtime. A similar view is held by Canfora et al. (Canfora et al., 2006) who argue that traditional testing is unable to cope with certain aspects of validating service–oriented systems, because of the impossibility to unforeseen system's configurations. Validation is required because run–time adaptation on its own is unable to provide confidence that an adapted system will behave correctly before it is actually deployed.

The sole use of design time (model based approaches) for self-validation is not adequate. Dustdar et al. (Dustdar et al., 2009) 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 modeling 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 the human users to observe the system and react on problems and critical situations. However, the goal of autonomic computing is to eliminate human intervention. A self learning approach could provide one solution.

3.1.3 Control mechanism

Control Mechanisms have been at the heart 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, (Janert, 2013). A key requirement for any self-adaptive system is to make use of control values that tell the system how to adapt. 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. A controller is used to achieve self-validation by producing a signal that is suitable as input to the controlled self adaptive application. This is achieved through numerical transformations. Although any component that transforms an input to an output can be used as a controller in a feedback loop, only two types of controller are encountered frequently: the on/off controller and the three-term (or PID) controller (Janert, 2013).

The on/off controller is simple to implement because it simply turns the signal on when the tracking error is positive and off when it is negative. Despite its simplicity a key drawback of this approach is that in practice systems do not need to change as soon as a deviation is detected but rather when the deviation reaches a threshold value. For example when identifying consumer preferences for a product, we cannot conclude that the preference has changed whenever a few customers start buying the product. The PID controller however will not send a signal unless the tracking error exceeds some threshold value and as such is a more practical approach. With the PID controller the signal is dependent on the occurrence of the error (unsatisfied consumer), size of the error (number of unsatisfied consumers), the frequency with which the errors occurs in a given time period (frequency of consumer dissatisfaction).

Tamura et al. (Tamura et al., 2013) describe feedback control loops as validation that depends on online measurements on 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. Dustdar et al. (Dustdar et al., 2009) present a solution that incorporates a model-driven compliance support, runtime interaction mining, run-time management of requirements, and explicit 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. (Cardozo et al., 2014) 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. (Fredericks et al., 2013) 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.

3.1.4 Strategy

The strategies used for validation in self-adaptive service-oriented systems are either reactive or proactive as shown on Table 3.1, with reactive strategies being most common. Hielscher et al. (Hielscher et al., 2008) describe reactive approaches as those that trigger adaptation based on monitored events. Consequently validation occurs after monitoring. A proactive approach on the other hand is closely tied to predictive adaptation where the adaptation occurs before monitoring and therefore it can be validated before monitoring.

Fleurey et al. (Fleurey et al., 2008) 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 validation is reactive because it occurs after monitoring. Similarly Baresi et al. (Baresi et al., 2003) also proposal 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 service-oriented systems, what is guaranteed at development time may not be true at run time. They advocate for a reactive approach by arguing that it is virtually impossible to predict all the evolutions and changes that might occur in the services we use, and the same is true for the environment. Therefore, monitoring allows them to take notice of infringements of expectation and react to them. Reactive adaption strategies suffer from several limitations as discussed at the

end of Section 1.1. Proactive adaptation is motivated by the fact that system changes such as service and quality failures can be costly and as such need to be predicted before they occur.

Hielscher et al. (Hielscher et al., 2008) outline some of these consequences as loss of money, unsatisfied users, and reduced system performance. Autili et al. (Autili et al., 2007) propose a proactive strategy that explores how to validate extra-functional issues during the service development and execution. They state that Bayesian Reliability Models and Queueing Networks can be analyzed at development time to validate the Service Model characteristics and a decision made available at run-time on how 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 performed (on the available models) and, based on the validation results; a new set of models is selected. Validation then occurs before adaptation is triggered based on past performance.

Another proactive approach is presented by Hoffert et al. (Hoffert et al., 2009) who point out the difficulty in maintaining the Quality of Service (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 complexity of managing multiple interrelated QoS settings. For such systems there is need to predict the conditions of their environment and adapt accordingly. Machine learning techniques such as decision trees, neural networks, and linear logistic regression classifiers, provide a promising solution. They can be trained on existing data to interpolate and extrapolate for new data. In particular, decision trees answer questions about which measurements and variables are most important when considering the reliability and latency of service systems. As an example they describe a Search and rescue (SAR) operations that helps locate and extract survivors in a large metropolitan area after a regional catastrophe, such as a hurricane, earthquake, or tornado. Infrared scans along with GPS coordinates are provided by unmanned aerial vehicles (UAVs) and video feeds are provided by existing infrastructure cameras to receive, process, and transmit *event stream* data from sensors and monitors to emergency vehicles that can be dispatched to areas where survivors are identified. Due to the dynamic environment inherent in the aftermath of a disaster, SAR operations must adjust in a timely manner as the environment changes. If SAR operations cannot adjust quickly

enough they will fail to perform adequately given a shift in resources. A proactive approach would be ideal in such a scenario.

Huber et al. (Huber et al., 2011) also emphasize the need for a proactive approach to self-validation. They state that in order to provide QoS guarantees (e.g, availability and performance) for virtualization and Cloud Computing environments; the ability to predict *at run-time* how the performance of running applications would be affected is required. They refer to it as *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.

Table 3.1. Current self-validation approaches for service-oriented systems

Approach	**Technique/mechanism**	**Involvement** (when applied)		**Primary focus**		**Control mechanism**		**Strategy**
		Off line	Online	Verification	Validation	Feedback	Feed forward	
King et al. (2007)	Regression testing	●	●	○	●	●	○	Reactive
Fleurey et al. (2008)	Model Based Techniques	●	○	●	○	○	●	Reactive
Baresi et al. (2003)	Model based	●	○	●	○	○	●	Reactive
Autili et al. (2007)	Model Based - Bayesian Reliability Models	◒	◒	○	◒	◒	◒	Reactive
Arcaini et al. (2015)	Formal Modeling	●	○	●	●	●	○	Proactive
Dustdar et al. (2009)	Model-driven approach	◒	◒	○	◒	◒	○	Reactive
Morin et al. (2009)	Model Based Techniques	●	○	○	●	○	●	Reactive
Salehie et al. (2009)	Formal methods	○	◒	◒	◒	○	◒	Reactive
Hoffert et al. (2009)	Decision tree, Artificial Neural Net-work, and Linear Logistic Regression Classifier	○	●	●	○	●	○	Proactive
Skałkowski et al. (2013)	Clustering algorithm	○	●	○	●	○	●	Reactive
Jureta et al. (2007)	Markov Decision Processes	○	●	●	○	●	○	Reactive
Canfora et al. (2006)	Regression testing	○	◒	○	◒	○	◒	Reactive

Approach	Technique/mechanism	Involvement (when applied)		Primary focus		Control mechanism		Strategy
		Off line	Online	Verification	Validation	Feedback	Feed forward	
Cardellini et al. (2009)	Model-Based, Linear programming	●	●	●	●	●	○	Reactive
Gomaa et al. (2014)	Model-based	○	●	●	○	○	●	Reactive
Cardozo et al. (2014)	Static symbolic exploration	○	●	●	○	○	●	Proactive
Wang et al. (2004)	State monitoring & Formal Specification	○	◒	◒	○	○	◒	Reactive
Bartolini et al. (2009)	Statistical metric	○	●	○	●	○	●	Reactive
Hielscher et al. (2008)	Regression Testing	◒	○	○	○	◒	○	Proactive
Weyns et al. (2016)	Model based	○	●	●	○	○	●	Reactive

Key

● - Supported

◒ - Weakly Supported

○ - Not Supported N/A- Not Available

The survey of self-validation techniques for adaptation in service-oriented systems presented in Table 3.1 shows that majority of the work on validation does not provide adequate details on how adaptation occurs. As a result most of the work shown in Table 3.1 is not presented in Table 2.1, which reviews work on self-adaptation. This observation reinforces the widely held view that validation is poorly supported in service-oriented adaptation (Papazoglou et al., 2007) (Cardellini et al., 2009) (Salehie, et al., 2009) (Lemos et al., 2013). King et al. (King et al., 2007), for example, provide details of a self-validating approach for testing adaptive Autonomic computing systems. They however do not describe how the adaptive system works and simply refer to the MAPE architectural blue print for Autonomic Computing put forward by IBM. This makes it challenging to adequately review the adaptation aspect of their work in order to understand how adaptation works. As a result they do not appear in Table 2.1, which focusses 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 3.1.

Only a handful of approaches tackle the issue of validation in self-adaptation. Dustdar et al. (Dustdar et al., 2009) 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. (Cardellini et al., 2009) propose an approach to adaptation and validation, however only sketchy details are provided on adaptation. They emphasize 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 off-line. This can be achieved by collecting statistics based on past adaptation decisions.

3.3 Summary

This Chapter began by highlighting the centrality of validation in self-adaptive service-oriented systems. The survey revealed that support for validation in self-adaptive service-oriented systems is still poor. For a system to self-validate at runtime, it needs to know when to involve the validation process, how to perform self validation, what error measurement to perform, and which control to use for self validation. These constitute the key requirements of a self-validating system.

The onset of validation is marked by an **involvement analysis** to arrive at a decision on *when* to involve the Self-validation process. Validation of a dynamically adaptive system can be conducted either online or offline. The decision is arrived at depending on whether the requirements for validation are know before or when the system is running. Following this decision a mechanism on *how* to evaluate the adaptation (**problem detection**) process has to be chosen in order to detect a problem. The survey revealed that researchers use various validation techniques such as formal methods, model-based, and machine learning techniques. However, most of approaches use a single technique. Different techniques offers contrasting strengths and weaknesses, and providing a mechanism to integrate them that can improve the quality and effectiveness of validation.

After selecting a suitable adaptation technique a decision on *what* to measure has to be arrived at for purposes of detecting deviations from the norm (**deviation assessment**). A self validating

system will signal the need to take corrective action whenever the output of the adaptor deviates from expectations. This deviation is referred to as the tracking error and is useful for gauging the adaptation decision for purposes of validation. For instance it can be used to describe consumer satisfaction of an adaptive system. This is achieved through controller numerical transformations such as the on/off controller and the three-term (Proportional-Integral-Derivative controller or PID) controller.

Finally, it is important to identify *which* **control mechanism** will be used to control the system based on the output of the validation process. This is synonymous to the way controllers work in industrial plants. In control theory there are two basic types of control mechanisms: *feedback* and *feed-forward*. The input to a feedback controller is the same as what it is trying to control and as a result the system can only react to a disturbance after it has occurred.

The survey also revealed that machine learning techniques enabled self-validating systems to accommodate the changing expectations of users through a self-learning process. However few researchers use machine learning techniques to evaluate the effectiveness of an adaptation solution. Further research work on machine learning techniques does appear to make use of several algorithms to improve on accuracy. Most of the research work on dynamic adaptation does not integrate both online and offline validation. There was also little evidence that the proposed solutions could work for different adaption triggers or different adaptation techniques.

Chapter 4

4 Theory and Design and Implementation

In Chapter 3 the key requirements for effective runtime validation in self-adapting service-oriented systems are outlined. For a service-oriented system to support runtime validation, it needs to know *when* to involve the validation process, *how* to perform self-validation, w*hat* error measurement to perform, and *which* control mechanism to employ. The review of self-adapting service oriented systems, discussed in Chapter 2, revealed the need to support different application contexts and change triggers. The review also highlighted the need for a pluggable solution that can be ported and tailored to different system and environmental settings. Lastly, online and offline validation approaches offer different strengths and limitations to an adaptation process. The best solution is one that finds a practical trade-off between the two. All these factors were taken into account in the design and implementation of the validation framework. In this chapter the system design and the prototype implementation of the validation framework is described.

4.1. Theory and Design

The Kenya government provides an online portal for public health information (GoK, 2015). As most Kenyans have no access to 'wired' Internet, the portal is also accessible wirelessly via mobile devices. The case study began by conducting a research on information access, funded by the government. The study used 1250 respondents, in 5 different geographical locations in Kenya. The research revealed that people in rural Kenya were experiencing difficulties accessing health-related information from the portal due to the varied capabilities of the mobile devices used (Namuye *et al.,* 2014). The research also revealed that people in different geographical areas had diverse health concerns and as a result sought varied health information. Considerable public interest as well as the potential for varied triggers made the health information portal a good case study for this research.

Figure 4.1 shows the architecture of the validation framework. The framework is service-oriented and comprises of an *application context, sensor, adaptation,* and *validation sub-systems*. Sensor services, adaptation services and validation services are hosted on the cloud. The sub-systems are discussed next.

4.1.1. Application context

The application context represents the logical area where the application executes. It captures the pertinent characteristics of the SOA system, including the consumer application, its orchestration platform, and system and user preferences. A service broker manages service requests on behalf of the consumer application and provides interfaces to the sensor and adaptation sub-systems. The application context sub-system contains functionality for accessing application components, generically loading file resources, publishing events, and resolving messages.

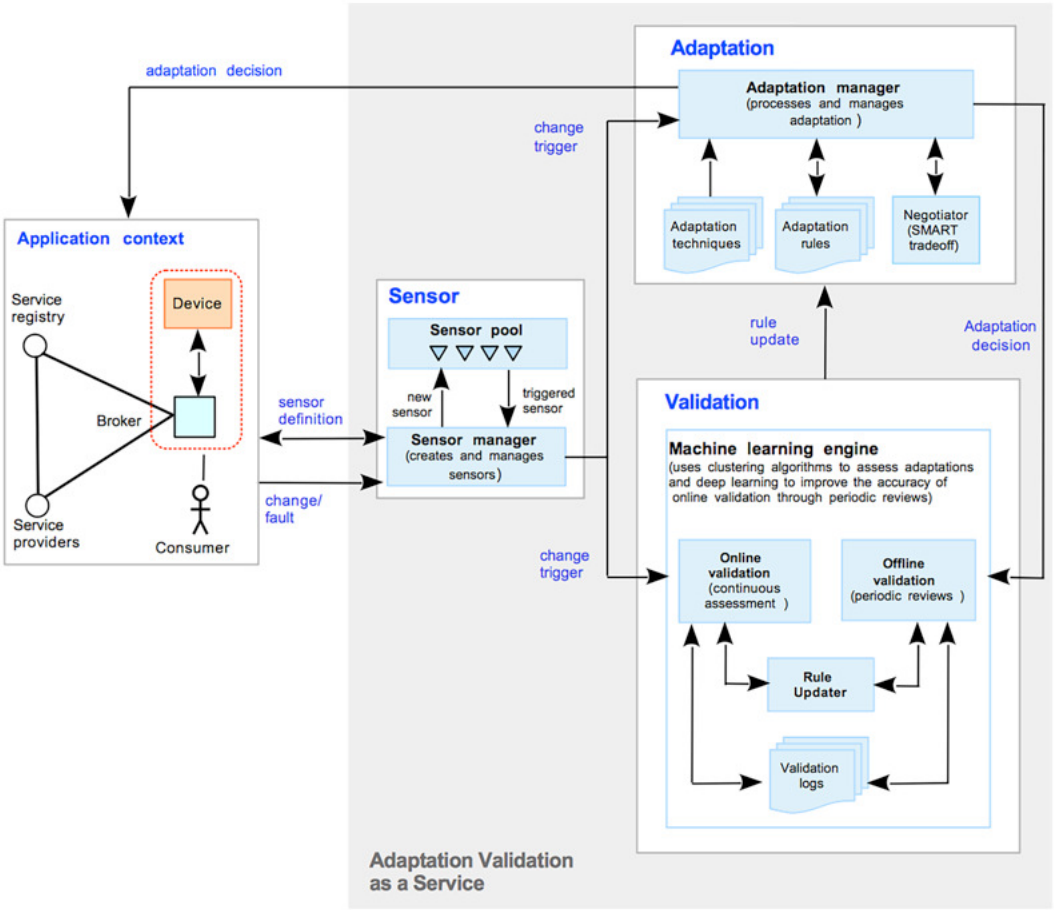


Figure 4.1 Self-validating runtime adaptation framework

4.1.2. Sensor

A sensor is a device that produces a measurable response to a change in a physical condition. The American National Standards Institute (ANSI) defines a sensor as a device which provides a usable output in response to a specified measurand. The broadest way that we can classify sensors is according to their nature, i.e. natural sensors and manmade or artificial sensors. In order to have more control over the environment, the creation of artificial sensors soon became an area of research interest. This was driven by the fact that natural sensors were not sufficient to detect all the changes in the environment and perhaps not as accurately as desired. Research then began on how to develop mechanical sensors. Mechanical sensors have been successfully implemented in building, utilities, industrial, home, shipboard, and transportation systems automation where they aid in detecting changes in the physical environment. The increasing interest in computer automation as well as adaptable software inspired the development of software sensors that can detect changes in the software environment dynamically and adapt accordingly.

Software sensors are increasingly finding applications in the area of context aware computing that aids systems in capturing user needs. In a survey of context aware models Strang and Linhoff-Popien (Strang et al., 2004) give an insight into various methods that software sensors can use to capture user needs. They look at key-value pair's models, Mark up scheme models, Object oriented models, Unified Modeling language models, Logic based models and Ontology based Models. These models are evaluated against requirements for ubiquitous computing and they concluded that the ontology approach came closes to fulfilling the requirements. The performance of sensors can be enhanced by introducing concepts such as evaluation and learning. Shavlik et al (Shavlik et al ., 2004) evaluate the effectiveness of a Software Sensors for detecting abnormal computer usage by using a machine learning algorithm that reduces the false alarm rates. In service oriented computing it is also important to provide users with the best service by taking into account the context information as well as historical information of service composition.

Therefore in the context of this research a software sensor is a computer application that provides a usable output in response to a specified measurement that is based on certain system changes. The sensor is responsible for detecting changes that are internal or external to a certain computer system. This way it differs from mechanical sensors in that it is not a physical device.

Software sensors operate in a manner similar to other sensors by incorporating a receiver (to detect changes), transducer (to measure), a knowledge base (holds the reasoning rules) and an actuator (that provides output).

The *sensor sub-system* is responsible for monitoring the system for changes and for the initial decision phase of the adaptation process. Its key components are the sensor manager and the sensor pool. The *sensor manager* is responsible for creating and keeping track of sensors. Sensors are created from user-defined change specifications or defined from pre-existing sensors. The sensor manager also acts as a low-pass filter to block spurious changes that may not warrant system adaptation. The *sensor pool* is a collection of sensors that are triggered by the event conditions that they monitor. A sensor has various elements including type, measurement units, sensitivity, trigger expression and life-cycle. A sensor expression is a logical expression that describes the conditions under which specific system properties may 'fire' the sensor. The sensors monitor changes in business environment of the system, system runtime environment, failures in service provision or a combination of these. When a sensor is triggered, the sensor manager refers this information to the adaptation manager after assessing its adequacy (i.e. whether it meets the preset conditions to be referred to the adaptation manager). For example, a geo-sensor might detect the user's geographic location whenever the client accesses the application.

4.1.3. Adaptation

When a systems change or changes match the conditions specified on a sensor, the sensor manager invokes the adaptation process by passing it the change information. The adaptation manager is responsible for assessing the request and recommending a suitable adaptation technique. The decision often requires the analysis and negotiation of a number of competing requirements as well as consideration of application context constraints. An adaptation solution describes how the system is reconfigured in a particular change context and the side effects of the reconfiguration. Papazoglou et al. (Papazoglou et al., 2007) and Baresi et al. (Baresi et al., 2003) identify the key techniques used to achieve runtime adaptation as self-configuring, self-healing, and self-optimizing models. The framework implements these adaptation techniques as a set of structured XML definitions. Each adaptation technique has a category, a description, and an action. For each technique the course of action calls for either re-organization of the workflow pattern, selection of alternative services, or the use of a wrapper service.

Figure 4.2 shows how adaptation techniques are described in the system. Techniques are mapped on to corresponding triggers by adaptation rules stored as XML documents. A typical rule comprises of a trigger and a decision process. The decision changes dynamically as the triggers are detected. Once an adaptation is selected, the framework keeps a structured log of the decision and the contextual information that informed the decision. Decision data from the process forms a feedback system that informs the validation process (see Figure 4.1). A number of workflow patterns are implemented with different trade-offs to evaluate the framework. For example, if a resource-constrained mobile phone is used to access the application, pre-defined resource sensors will detect this and cause the adaptation manager to re-orchestrate the application to make use of a work pattern that utilizes minimal system resources.

```
<?xmlversion="1.0"encoding="UTF-8"standalone="no"?>
<ASRules>
<ATech>
       <ATType>selfOptimizing</ATType>
       <ATDesc>WF1</ATDesc>
       <ATWorkflow>CLientServer</ATWorkflow>
       <ATAlternative></ATAlternative>
       <ATWrapper></ATWrapper>
</ATech>
<ATech>
       <ATType>selfOptimizing</ATType>
       <ATDesc>WF2</ATDesc>
       <ATWorkflow>BrokerCLientServer</ATWorkflow>
       <ATAlternative></ATAlternative>
       <ATWrapper></ATWrapper>
</ATech>
<ATech>
       <ATType>selfHealing</ATType>
       <ATDesc>geoS1</ATDesc>
       <ATWorkflow></ATWorkflow>
       <ATAlternative>geoS2</ATAlternative>
       <ATWrapper></ATWrapper>
</ATech>
```

Figure 4.2 Snippet of adaptation techniques

4.1.4. Supporting Negotiation

Because adaptation triggers are not mutually exclusive the framework needs to be able to deal with overlapping change triggers. Overlapping triggers may result in conflicting adaptation decisions. To address this, the framework includes an implementation of the multi-attribute utility model, *Simple Multi-Attribute Rating Technique (SMART)* (Alfares et al., 2008), to support negotiation through a process of trade-off analysis. Trade-off analysis is intended to support the

process of finding an acceptable balance amongst competing requirements and attributes. Rather than relying on pair-wise comparisons, an assumption of the *SMART* approach is that performance attributes can ultimately be measured, and a value assigned. *SMART* also allows the weighting of attributes in order to recognize their respective priority. *SMART* provides a means for assessing competing change requirements to reflect their relative importance to the adaptation decision. By refining the scores with the relative weights of competing change requirements, the utility value or contribution for each alternative solution can be computed. The utility function used in the framework is shown in equation 4.1; where w_j is the scaling value (weight) assigned to the j^{th} of k quality concerns, s^*_{ij} is the utility for alternative j on criterion i, and n is the number of alternative decisions.

$$\mu_j \; - \sum_{i=1}^{m} w_i s_{ij}^* \; / \sum_{i=1}^{m} w_i \quad for \; j - 1...n \qquad (4.1)$$

A maximum score of 1 for the utility indicates the highest probability of the concerns being achieved, whereas the minimum of zero indicates the least acceptable trade-off. A high rating increases the likelihood of a particular decision being selected, specific analysis of its contribution to individual change concerns may be needed to provide a better understanding of the choices at every level. Once a decision is selected, it is evaluated by the validation component before it is executed.

Decision-making in this research involved the complex search for information amid uncertainty, conflicting requirements, and personal preferences. To show how *SMART* was used to support negotiation, multiple attributes such as geo-location, memory, processor speed, device type, Platform, reputation, and cost are considered. User preferences may be based on attributes/criteria that are conflicting such as a service with a good reputation and a low cost. Often a high reputation service will cost more. A decision process is required that will factor in all the user preferences and obtain scores that will arrive at a negotiated decision.

The first task when using SMART called for identifying the alternative services and the values for the criteria associated with each alternative as shown in Table 4.1.

Table 4.1 Sample Criteria for Alternative Services

Alternative	Geolocation	Memory	Speed	Device Type	Platform	Reputation	Cost
	(C1)	(C2)	(C3)	(C4)	(C5)	(C6)	(C7)
S1	1	1GB	1.2GHz	Mobile	Windows 8	very high	very low
S2	1	1GB	1.2GHz	Mobile	Windows 8	high	very low
S3	1	4GB	2.3GHz	Laptop	Windows 7	very high	low
S4	2	4GB	2.3GHz	Laptop	Windows 7	high	low
S5	1	4GB	3.4Ghz	PC	Windows 7	very high	high

Next the weights for each criterion were identified from the validation logs and used to allocate weights (in the interval of 0-30) based on users priorities as shown in Table 4.2. All the values are then summed. The weights of each criterion are normalized by dividing the weight of criteria weights (W_j) with a total weight value ($\sum W_j$) to give a value between 0 and 1.

Table 4.2 SMART Criteria Weights

No	Criteria (C)	Weight (W_j)	Normalized weight ($W_j/\sum W_j$)	
1	C1	20	20/100=	0.2
2	C2	7	7/100=	0.07
3	C3	13	13/100=	0.13
4	C4	5	5/100=	0.05
5	C5	5	5/100=	0.05
6	C6	25	25/100=	0.25
7	C7	25	25/100=	0.25
sum($\sum W_j$)		100		

To score the alternative services the parameters in Table 4.1 must be assigned values. This is values are grouped as illustrated in Table 4.3.

Table 4.3 SMART parameter values

	Reputation		Cost		Device		Platform
Group	**Parameter value**	**Group**	**Parameter value**	**Group**	**Parameter value**	**Group**	**Parameter value**
very low	1	very low	4	Ipad/Tablet	1	Windows xp	1
low	2	low	3	Laptop	2	Windows 7	2
high	3	high	2	PC	3	Windows 8	3
very high	4	very high	1	Mobile	4	Windows 10	4

The alternative services in table 4.1 can now be described using the rating criteria as shown in Table 4.4.

Table 4.4 Parameter values for Alternative Services

Alternative	Geo-location	Memory	Speed	Device Type	Platform	Reputation	Cost
	(C1)	(C2)	(C3)	(C4)	(C5)	(C6)	(C7)
S1	1	1	1.2	4	3	4	4
S2	1	1	1.2	4	3	3	4
S3	1	4	2.3	2	2	4	3
S4	2	4	2.3	3	2	3	3
S5	1	4	3.4	3	2	4	2

The table is then normalized using the normalization *Equation 3.3*. The alternative services with the normalized criteria now appear as shown in Table 4.5.

Table 4.5 Normalized Parameter Values For Alternative Services

Alternative	Geo-location	Memory	Speed	Device Type	Platform	Reputation	Cost
	(C1)	(C2)	(C3)	(C4)	(C5)	(C6)	(C7)
S1	0.00	0.00	0.00	1.00	1.00	1.00	1.00
S2	0.00	0.00	0.00	1.00	1.00	0.00	1.00
S3	0.00	1.00	0.65	0.00	0.00	1.00	0.50
S4	1.00	1.00	0.65	0.50	0.00	0.00	0.50
S5	0.00	1.00	1.00	0.50	0.00	1.00	0.00

The final stage is the calculation of the score of each alternative. This is done by multiplying the utility value criteria with the normalized weight value corresponding to each criterion given in Table 4.2. The final shore results are illustrated in Table 4.6.

Table 4.6 Final SMART Scores for Alternative Services

Alternative	Final Score
S1	0.60
S2	0.35
S3	0.51
S4	0.49
S5	0.48

The results tell us that the most recommended alternative is service S1 that has the highest score. SMART recommends that a consumer using a mobile phone in location 1 should use service S1 rather than service S2. If on the other hand a laptop is used then service S3 is preferred. This decision is based on the weights supplied as identified by consumer priorities for each criterion

4.1.5. Validation

One way in which validation can be achieved is through techniques that prompt the user for feedback or through crowd sourcing approaches. However, although these approaches provide a useful means for data collection through collaborations within the community, there are challenges where services that offer similar functionalities compete (Bouguettaya et al., 2017). Key amongst these is how to gauge the quality and trustworthiness of crowd sourcing contributors. Further, there is no standardized testing platform to compare trust and reputation models. To address these limitations, our framework uses the past behavior of consumers rather than their opinion to gauge reputation and quality of feedback. If similar users in the recent past have repeatedly accepted a particular adaptation decision then the decision is most likely valid. The user is not prompted for feedback and therefore they cannot intentionally manipulate the feedback they provide.

The framework uses machine learning to assess adaptation decisions and to modify the rules that inform the decisions. Figure 4.3 shows the validation process where a change trigger

evaluates existing adaptation rules from decision logs as well as consumer feedback. This is a representation of the workflow process for the validation component depicted in Figure 4.1. The validation component receives the change trigger from software sensors and the adaptation decision from the adaptation component. These two inputs are compared to existing validation logs by the validation component in order to vet the adaptation decision. The vetting process will reveal if there is a need to update the adaptation rules or simply retain them as they are. The final validation decision (retained adaptation rules or updated rules) is logged for future machine learning purposes.

Figure 4.3 shows how the validation decision is arrived at. The process begins by performing validation using machine learning algorithms to determine the validity of the adaptation decision based on the change triggers received. Both supervised and unsupervised machine learning techniques are used. The unsupervised approach used is cluster analysis while the supervised approach is classification using deep neural networks. Cluster analysis is performed at runtime due to its efficient utilization of system resources. However experiments reveal that the approach is not as accurate when dealing with natural data. Therefore to reinforce the technique a supervised machine learning approach is used that is based on Neural Network classification. This technique however places heavy restrains on system resources and therefore is not ideal for runtime validation. This research takes advantage of both by performing validation online using clustering and offline using deep Neural Networks. A clustering algorithm is used to assess and refine the adaptation decisions and deep learning to review and improve the accuracy of validation predictions.

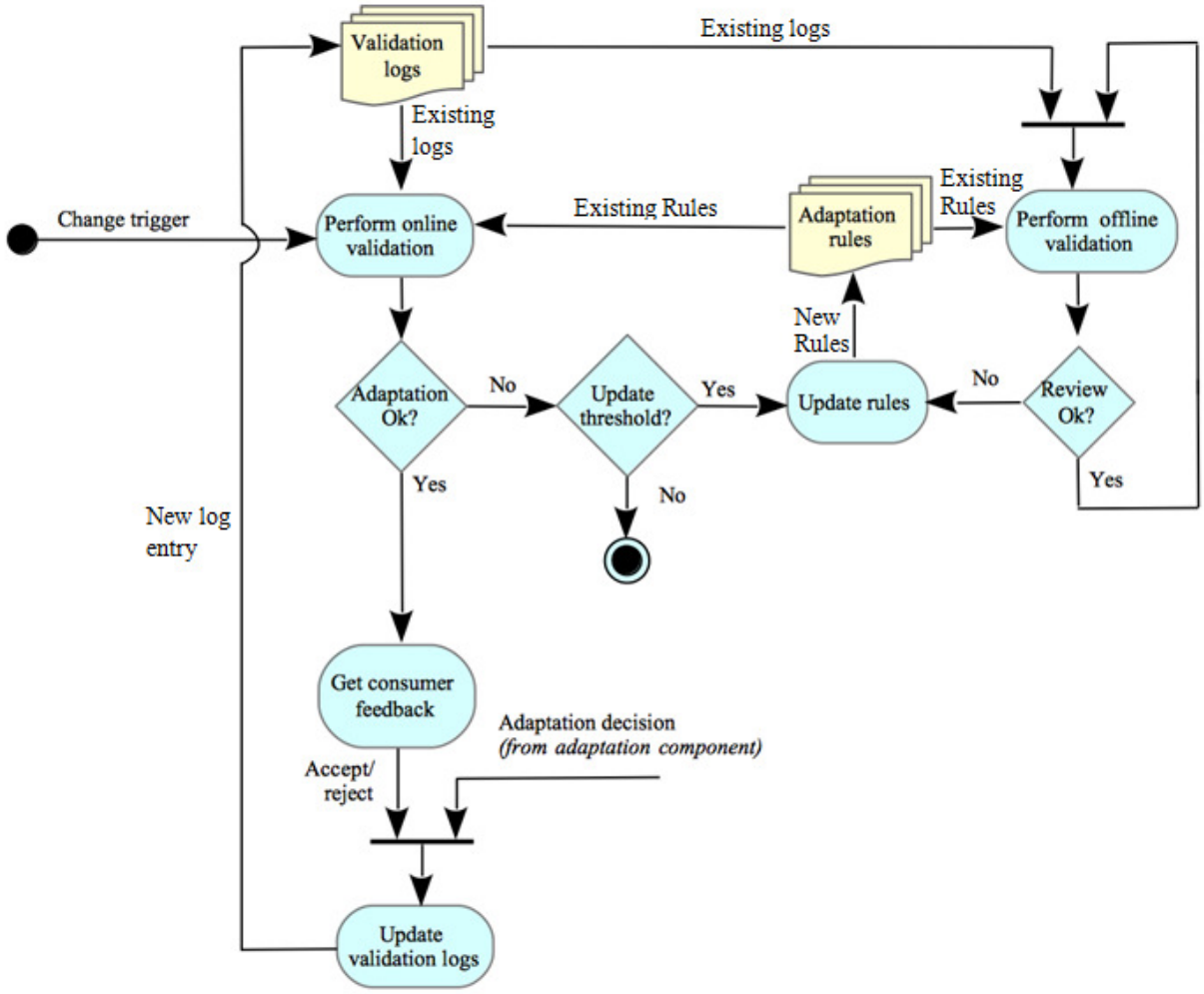


Figure 4.3 Adaptation validation

Clustering analysis identifies clusters that exist in a given dataset, where a cluster is a collection of cases that are more similar to one another than cases in other clusters. The *Expectation-Maximization* (EM) algorithm is used, a statistical model that makes use of the finite Gaussian mixtures model that assumes all attributes to be independent random variables (Pardalos, et al., 2013). The parameters are re-computed until a desired convergence value is achieved. The probability density function for a normal distribution is used to compute the cluster probability for each instance. EM clustering is chosen because it provides better optimization than distance-based or hard membership algorithms, such as *k-Means*. Decision logs record change triggers received and the adaptation decision made. If the decision made is inadequate, this is detected as the system almost immediately triggers another request for adaptation. When a mismatch reaches a

stipulated threshold, adaptation rules are updated by the validation component. EM clustering is performed in two steps:

- *Expectation* – First the initial mixture model parameter estimates are input. A single iteration provides new parameter *estimates* that do not decrease the log likelihood of the model. The expected value of the log likelihood function, is given as:

$$Q(\theta|\theta^{(t)}) = E_{z|x,\theta^{(t)}} [log\ L(\theta; X; Z], \qquad (4.2)$$

  Where Q is the conditional distribution of Z given X under the current estimate of $\theta^{(t)}$.

- *Maximization*– Secondly the process is repeated until the log likelihood of the mixture model, at the previous iteration, is sufficiently close (*maximised*) to that of the current model. The parameter that maximizes this quantity is obtained as:

$$\theta^{(t+1)} = arg_{\theta} max\ Q(\theta|\theta^{(t)}) \qquad (4.3)$$

Clustering algorithms analyze the data to find groups of maximum commonality. For example, if the framework establishes that at time t_n adaptation technique A_1 was generally the acceptable decision for adaptation trigger *X*. Then at time t_{n+1} the acceptable adaptation technique for the same trigger was A_2, the framework changes the adaptation rules accordingly. In this case decision logs act as adaptation triggers.

To minimize the impact of resource and performance overheads, the framework uses deep learning to statically improve the accuracy of online validation by periodically reviewing validation logs and updating adaptation rules. In order to train the neural network using this algorithm the dataset has to be normalized for the values to range from *0* to *1*. The feature scaling formula given by equation 4.4 is used for this. Where *x* is the value that should be normalized, x_n is the normalized value, x_{min} is the minimum value of *x,* and x_{max} is the maximum value of *x.*

$$x_n = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \qquad (4.4)$$

A key concept underlying deep learning methods is the use of distributed representations of the data, in which a large number of possible configurations of the abstract features of input data are

feasible. *S*everal nodes with several inputs (x_n) and weights (w_n) exist and the net final output is calculated by summing all the inputs multiplied by their weights plus the bias (b), and then run through a limiter. This can be expressed as:

$$Net = \sum_{i=0}^{n} x_i w_i + b \qquad where \qquad f(x) = \{\frac{x \leq 0 \rightarrow 0}{x \geq 0 \rightarrow 1}\} \qquad (4.5)$$

The action of selecting the correct weights is the process that results in the learning rules/algorithms (referred to as adaptation rules in this research study). Environmental and business triggers are combined as the input features for the neural network - i.e. memory capacity, processor speed, device type, and geo-location. The error-correction approach that focuses on the fact that the output perceived in the training process does not always correspond to the desired output is used. As a result the method calculates the error, modifying the weights to progressively reduce the error. Initially the weights are set to random small numbers and progressively updated during training based on the calculated error. This is the basis of the back propagation-learning algorithm. Training the perceptron calls for the modification of its weights.

Following validation a decision on the validity of the existing adaptation rules is arrived at. If the adaptation decision is valid then there is no need to update the adaptation rules. The system then provides the consumer with the adaptation decision as recommended by the adaptation component. The consumer behavior is then observed to find out if the decision is acceptable to the consumer. If it is acceptable the decision is logged to inform future validation processes. If on the other hand the consumer rejected the decision for an alternative one, the alternative is logged. The logs are an indication of current consumer preferences. The validation decision could alternatively have indicated that the adaptation decision is not valid. This however does not immediately indicate that the adaptation rules should change unless the stipulated threshold values are exceeded.

4.2. Prototype Implementation

The prototype framework was implemented using Java Web Services, which are software component that communicate with each other over a network, (Deitel et al., 2011). Web services communicate using such technologies as XML, JSON and HTTP**.** Java Web Services are created

using the netbeans platform, which is open source. The framework is made up of client and server applications, each with several modules, which are discussed next. It is important to mention that the framework is intended to support runtime adaptation in general service-oriented systems. However, to illustrate various aspects of its implementation, specific examples drawn from a healthcare case study are included and used to evaluate the framework. The healthcare case study is an online portal for public health information set up by the Kenyan government (Namuye et al., 2014). A detailed description of the case study and evaluation of the framework is provided in Chapter 5.

4.2.1 The Client Application

Client applications run in either the client's device or server depending on the runtime resources available to the client. The client has only one web service to ensure that it is light enough for the client's machine, if it has to run on the client's machine. The role of the client is to send a HTTP request and get a HTTP response from the client's machine via a servlet. The response contains details on the client's environment that is used by the systems sensors. The client application then calls the sensor manager in the adaptation service from the server to evaluate the information obtained from sensors. The sequence diagram in Figure 4.4 shows how the user interacts with the client application.

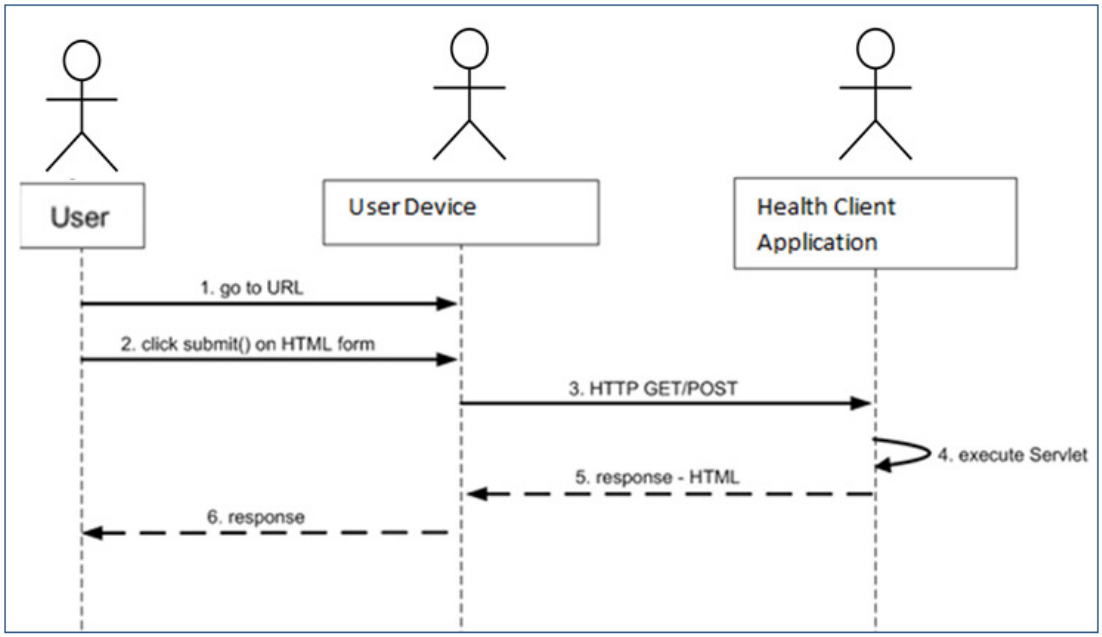


Figure 4.4 User Interaction with the Client Application

4.2.2 The Server Application

In this particular case, the server application is made up of several web services and updates several files used by the system. A description of these services follows:

- *Sensor Services* – This is a group of services that are used to decipher information received from the user's environment. It contains services such as the geo-sensor, which deduces the user's location ID from the latitude and the longitude received from the user's environment. The sensors are the first services invoked in the server application. The service is represented by the *callSensor* service shown in Figure 4.5.
- *Decision Service* – The role of this service is to make a decision on whether to adapt or not based on predefined adaptation rules. For instance it can retrieve the details of the user geo-location as well as the disease prevalent in that location based on predefined adaptation rules. The service is represented by the *checkLocation* service shown in Figure 4.5. It must therefore read the adaptation rules. A key role of this web service is to call a validation service that vets the adaptation rules using a machine-learning algorithm (EM clustering).
- *Validation service* – This service uses validation statistics collected over time as the training set for the machine learning algorithm. If significant changes are detected in the adaptation rules it will update the adaptation rules. For example if users have recently shown a keen interest on a certain disease that is different from the one the system automatically brings up (e.g. due to a new disease outbreak) then the adaptation rules need to change to reflect this as the relevant disease for that location. The service is represented by the *adaptRules* service shown in Figure 4.5.
- *Adaptation Service* – If a decision to adapt was made by the decision service then this service performs adaptation based on the updated adaptation rules retrieved by the decision service. For example it can bring up the details of the disease retrieved by the decision service e.g. the disease description and possible causes or symptoms. The service is represented by the *checkDisease service* shown in Figure 4.5.
- *Feedback Service* – This service gives the user the option of rejecting the adaptation decision in favor of another. For example the user can invoke it to search for an

alternative disease if they are not happy with the disease information provided by the adaptation service based on existing adaptation rules. It invokes the adaptation service again prompting it to check for an alternative disease, (*checkDisease* service, Figure 4.5). It is this feature that is used to instrument user acceptance by providing data to gauge the systems performance.

- *Logging Service* – Just before the user exits the application; this web service is called to update the validation statistics file. A log of the adaptation trigger (e.g. location) and the final user accepted decision (e.g. disease interest) is kept. This information is used by the *Validation Service* in future adaptation as part of the training set for the machine-learning algorithm. The service is represented by the *updateSats* service shown in Figure 4.5.

An illustration of how all these system components work together is shown in Figure 4.5.

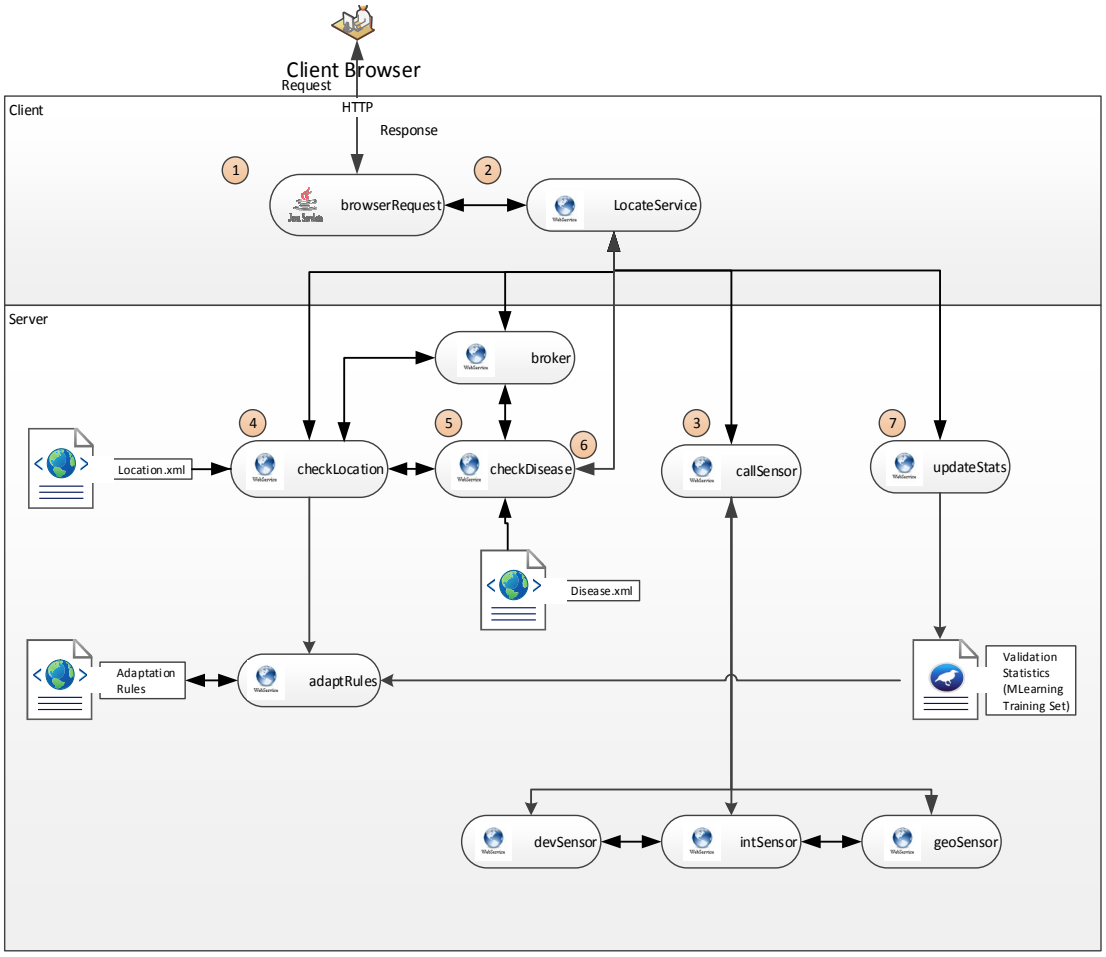


Figure 4.5 SOA Application System Workflow

The figure shows how the different services interact with each other. Only the browser request and location service run on the client application to minimize resource consumption on the user's device. The role of the client is to detect the user's geographic location and device, as well as interact with the user. This information is passed on to sensors in the server for purposes of reasoning about adaptation. For example the device sensor will receive information on the device model after which it deduces the memory capacity and processor speeds. If these measurements are below the threshold values then the system needs to adapt to select workflows or services that utilize minimum system resources. The adaptation (check location and disease) and validation (adapt rules) processes are then invoked next and take place in the server, running in the service providers cloud environment. This ensures efficient utilization of the users resources.

4.2.3 Web server

The prototype runs on a lightweight open web server designed for building and deploying enterprise Java applications on the cloud. This server is chosen because of its lightweight micro-container architecture and portability, which is critical for dynamic applications on the cloud. The server also, provides support for the development of interactive applications of HTML5 clients or native mobile applications, as well as support for *JSON* processing, which is used in the prototype.

4.2.4 Knowledge base

The application needs to store data about the triggers that it will look out for, the different adaptation techniques that it can use, the rules governing the choice of adaptation techniques, and validation statistics to evaluate the adaptation rules. This section describes how this data is stored.

4.2.4.1. Trigger specification

A change trigger is specified in XML notation and comprises of the change type (*CType*), change properties (*CProp*) that represent salient aspects of the system together with their associated types (*CPType*) and a change expression. The trigger expression is represented by the trigger check (*CTCheck*), and the limit (*CTLimit*). Examples of trigger types include service failure, resource failure and requirement change. It is worth mentioning that there are numerous reasons why a service can fail including if the service is not available, or has an error or cannot integrate with the other services. The different reasons are represented as the trigger properties. A

resource failure trigger for instance could be memory capacity or CPU time described as the trigger property. This is further expressed as an Integer that should be less than a specified limit.

```
<CMRule>
    <CType>Env</CType>
    <CProp>Data</CProp>
    <CPType>s</CPType>
    <CTExpress>
        <CTCheck>==</CTCheck>
        <CTLimit>M</CTLimit>
    </CTExpress>
</CMRule>
```

Trigger description

change type: aChangeType
property{$prop_1$, $prop_2$, …,$prop_n$}
property type: {numeric/string/other}
trigger expression: {check, checkLimit}
check: {&lt, ==, &gt, !=, and, or, not}
ckeckLimit: aThresholdValue **or** aThresholdLabel

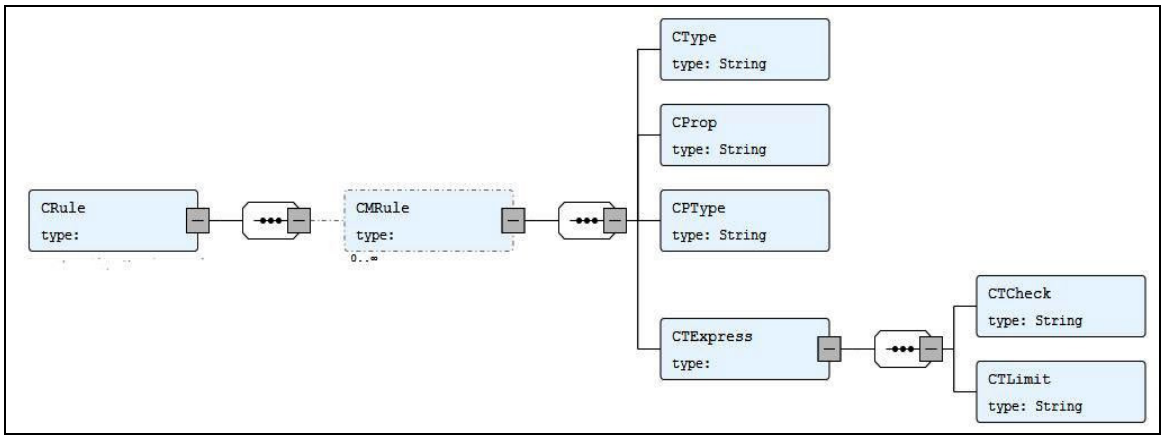
```
<CMRule>
    <CType>QoS</CType>
    <CProp>Mem</CProp>
    <CPType>I</CPType>
    <CTExpress>
        <CTCheck>&lt;</CTCheck>
        <CTLimit>200000</CTLimit>
    </CTExpress>
</CMRule>
```

Figure 4.6 Example of trigger specification and threshold values

The different reasons are represented as the trigger properties. Figure 4.6 shows a code snippet of the XML trigger specification. It shows a specification of a runtime trigger that requires memory to be less than 200,000. Additional triggers can be added as required to accommodate new changes and constraints. To ensure that trigger specifications are valid, they are checked against a *change meta model* that maintains the rules on valid triggers specifications. Figure 4.7 (a) shows part of the meta model description.

```
<xs:schema attributeFormDefault="unqualified"
elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="CRule">
    <xs:complexType>
      <xs:sequence>
       <xs:element name="CMRule" maxOccurs="unbounded" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="CType"/>
              <xs:element type="xs:string" name="CProp"/>
              <xs:element type="xs:string" name="CPType"/>
              <xs:element name="CTExpress">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="CTCheck"/>
                    <xs:element type="xs:string" name="CTLimit"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
```

(a) Change Meta model



(b) Visualization of the change meta model

Figure 4.7 Change Meta-Model

As users and service providers discover additional change triggers and make updates, the change model is validated against an existing Meta model to ensure consistency. A visualization

of this meta model is shownin the Figure 4.7 (b). Figure 4.8 shows how a SAX Parser (Simple API for XML) transformstheXML document into java readable linear events.

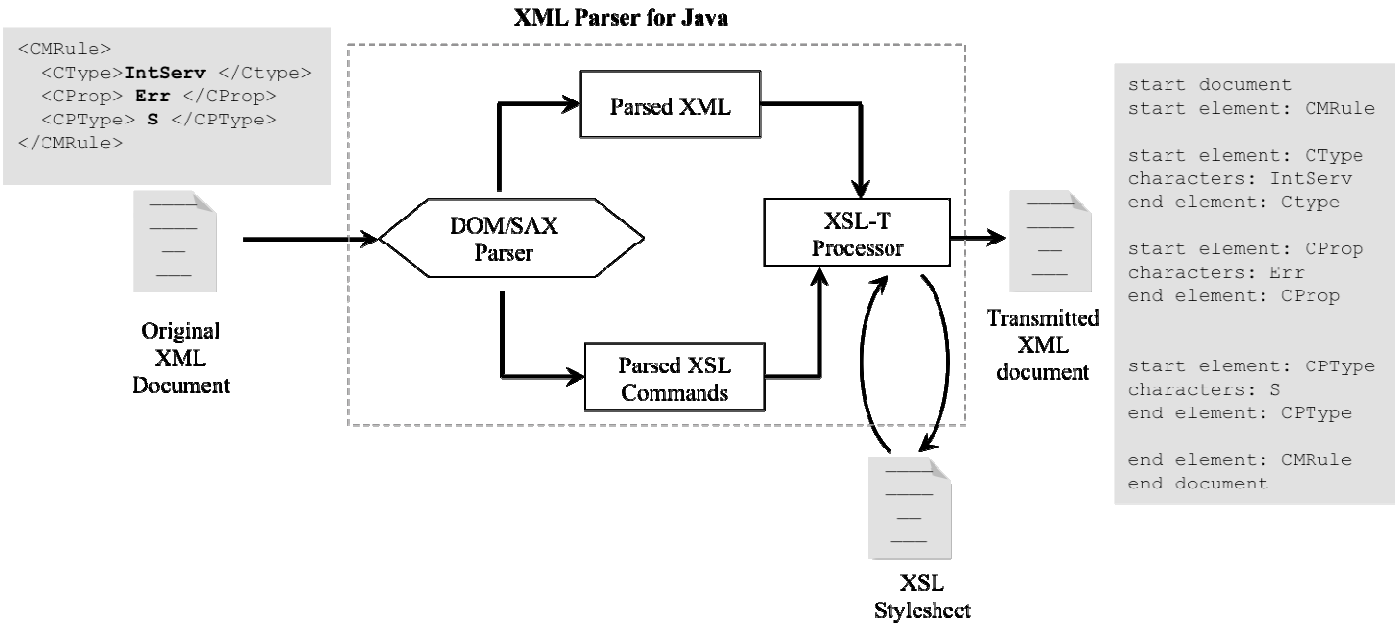


Figure 4.8 SAX Parser for the trigger specification

4.2.4.2. Adaptation Techniques

As mentioned in section 4.1.3, adaptation techniques can be self-configuring, self-optimizing, or self-healing. Adaptation techniques are described in the frame work using XML notation as illustrated in Figure 4.2. Each adaptation technique has a category, a description, and an action. For example, the first adaptation technique displayed is a self-optimizing technique, `<ATType>`, which works by recommending a certain workflow pattern, `<ATDesc>`. For each technique the course of action may require re-organization of the workflow pattern, selection of alternative services, or the use of a wrapper service. In the first adaptation technique the action is re-organizing the workflow pattern so the other options are empty i.e. `<ATAltenative>` and `<ATWrapper>`. The workflow pattern in this case is simply where the client calls the server. In the second adaptation technique, which also calls for the re-organization of workflow patterns, a broker is used to call the client that in turn calls the server.

Because adaptation techniques are invoked depending on the triggers detected, it is possible that multiple triggers can be detected that call for conflicting adaptation techniques. For instance

the user's geographic location could indicate that users prefer a high reputation service therefore the adaptation technique should call for reconfiguration with high reputation services. For example if the user is located in a big city in the central business district they often need high reputation services. The user on the other hand could be accessing the application for personal use rather than for business and therefore requests a low cost application. This request would call for reconfiguration with low cost services. High reputation services will often cost more than low reputation services therefore there would be a conflict in the adaptation techniques. In such cases a compromise has to be arrived at through the negotiation process described in section 4.1.4 in order to identify the most acceptable adaptation technique.

4.2.4.3. Adaptation Rules

Adaptation rules match a detected trigger to the corresponding adaptation behavior. For example a detected geo-location can be matched to the corresponding health information (*self-configuring adaptation technique*), or a detected device matched to the corresponding workflow pattern (*self-optimizing adaptation technique*). The framework implements these adaptation rules as a set of XML rules. Figure 4.9 shows the adaptation rules created for a *self-configuring* adaptation technique. These adaptation rules are designed from a preliminary survey conducted to show user preferences i.e. what are the health concerns for users in a certain geographical location.

```
<DRule>
    <LiD>1</LiD>
    <LName>Ngumba</LName>
    <LLatt>-1.2332443</LLatt>
    <LLong>36.885997</LLong>
    <LRad>1</LRad>
        <DName>Cholera</DName>
        <DDesc>Water Borne Disease</DDesc>
        <DLoc>1</DLoc>
</DRule>
<DRule>
    <LiD>5</LiD>
    <LName>Lancaster University</LName>
    <LLatt>54.010385</LLatt>
    <LLong>-2.787772</LLong>
    <LRad>5</LRad>
        <DName>Lyme Disease</DName>
        <DDesc>Infectious disease transmitted by ticks</DDesc>
        <DLoc>5</DLoc>
</DRule>
```

Figure 4.9 Self-Configuring Adaptation Rules

To create rules for a *self-optimizing* adaptation technique, preliminary experiments were conducted that identified four workflow patterns (i.e. *WF1, WF2, WF3* and *WF4*), representing different performance and efficiency trade-offs. These workflow patterns were chosen to represent different architectural styles such as the Client-Server, Message Bus, and Layered Architecture. When the service-oriented system is orchestrated in different ways, some quality aspects of the system were observed to change as illustrated in Figure 4.10. Based on the user's runtime resource constraints, the appropriate workflow can be invoked. For example, if a user is accessing information from a PC, then the user has sufficient resources and the system does not need to adapt to address resource limitations in the runtime environment. It simply calls the service-oriented system and runs it as it is. If however a user decided to use a mobile phone instead, the system detects that the user is on a platform that has limited resources in terms of memory and possibly bandwidth and must therefore invoke a workflow that uses minimal system resources. The flexibility of the framework allows the user to add additional workflow patterns.

Resource / Workflow	CPU Performance (milliseconds)	MEMORY (Bytes Allocated)
WF1	781	245,472
WF2	675	228,312
WF3	758	226,472
WF4	4271	185,856

(a)

```
<?xmlversion="1.0"encoding="UTF-8"standalone="no"?>
<WFRules>
<WFRule>
    <WFlow>1</WFlow>
    <ETime>3</ETime>
    <MCap>4</MCap>
</WFRule>
<WFRule>
<WFlow>2</WFlow>
    <ETime>1</ETime>
    <MCap>3</MCap>
</WFRule>
```

(b)

Figure 4.10 Self-optimizing Adaptation Rules

From these experimental results sample adaptation rules were developed. The optimization algorithm, based on *SMART*, recommends *WF3* when a mobile phone is detected for high CPU efficiency. *WF2* can be an alternative if *WF3* fails although it does not perform as well in terms of CPU time thus a trade of on the High CPU efficiency is experienced. For a PC, *WF1* is used. The adaptation rules in XML have symbolic numbers to represent this. For example workflow *WF2* has a low value, 1, for low CPU time and a high figure, 3, for high memory capacity.

4.2.4.4. Validation Logs

The adaptation component writes the output of the decision process to a file as logs of decision statistics to be used for validation. For purposes of this research, this output is the detected geo-location (`LiD`) as well as the users accepted decision (`sname`) i.e. the desired health information. An extract of this file is illustrated in Figure 4.11 (a).

The validation logs act as a training dataset that will be manipulated by the EM Clustering algorithm for learning and subsequently updating the adaptation techniques. A second copy of these validation logs is also created and stored in CSV format periodically to enable the

application to validate the adaptation decision using deep learning. Figure 4.11 (b) illustrates an extract of this file. This is performed offline.

```
@relation validstats2

@attribute LiD numeric
@attribute sname numeric

@data
1,2
1,2
1,2
1,2
1,2
1,2
1,2
1,1
```

(a) EM Clustering validation logs

Input 1 ***(Memory)***	**Input 2** ***(Device)***	**Input 3** ***(CPU)***	**Input 4** ***(Location)***	**Output 1**	**Output 2**	**Output 3**	**Output 4**	**Output 5**
1	1	0	0.75	1	0	0	0	0
1	1	0	0.75	1	0	0	0	0
1	1	0	0.75	1	0	0	0	0
1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	0	1	0

(b) Deep learning validation logs

Figure 4.11 Validation logs

4.3. Machine Learning Tools

Adaptation rules need to be continuously evaluated and updated because user and system requirements are not static. In autonomic computing this can only be achieved through machine learning. Two machine learning algorithms have been implemented to test the frameworks ability to validate and update the adaptation decision i.e. Clustering and Neural Networks. A description of the tools used follows.

4.3.1 EM Clustering Algorithm

In order to make a decision on whether the systems decision (adaptation rules) needs to change, machine learning techniques that mine the validation statistics to arrive at a decision are used. Machine learning algorithms include classification, regression, clustering and association. Libraries from WEKA (Waikato Environment for Knowledge Analysis), an open source tool that makes use of machine learning algorithms implemented in Java, are imported. The machine learning algorithm is developed as a java web service based on the work of Shetty et al, (Shetty *et al*, 2010). Shetty *et al.,* state that so far data mining applications are run from the desktop, which calls for installation in the user machines to solve the data mining tasks. This occupies a lot of memory on the system for storing the data repository and the software. If data is to be availed

across the net, implementing the machine learning algorithms as web services would make it easier to access them and cheaper.

Clustering analysis identifies clusters that exist in a given dataset, where a cluster is a collection of cases that are more similar to one another than cases in other clusters. Clustering algorithms are ideal forthecase study as they can be can be used to detect the type of disease information that users in a certain location are searching for. If the users start searching for a different disease the clusters change and the new clusters can be used to update the adaptation rules. Figure 4.12 illustrates the web service response and requests generated by the validation web service created. The results indicate that for the requested location numbered 1 the response is the disease numbered 2 marked by the detected cluster 0.

```
request-1504271962455

<?xml version="1.0" encoding="UTF-
8"?><soap:Envelopexmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header/>
  <soap:Body>
    <ns1:execute xmlns:ns1="http://ws.geohealthf.com/">
      <input>c:/validstats2.arff</input>
      <slid>1</slid>
    </ns1:execute>
   </soap:Body>
</soap:Envelope>
```

(a) Service Request for validation cluster for location ID 1

```
response-1504271969604

EM
==

Number of clusters selected by cross validation: 4

            Cluster
Attribute     0      1      2      3
             (0.05)   (0.27)  (0.21) (0.21)
=========================================================
LiD
mean        1.0016   2.1632   4      2.6073
std. dev.   0.0397   0.3944   1.1999 0.4888

sname
mean        1.5104   3.3127  5       1.5556
std. dev.   0.631    0.5551  1.5383  0.5494

<?xml version="1.0" encoding="UTF-
8"?><S:Envelopexmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns0:executeResponse xmlns:ns0="http://ws.geohealthf.com/">
      <return>2</return>
    </ns0:executeResponse>
  </S:Body>
</S:Envelope>
```

(b) Service Response returns validation results as cluster 2

Figure 4.12 Validation Web Service Request and Response

4.3.2 Deep Learning Algorithm

To implement deep learning as a learning algorithm for adaptation, Neuroph Studio was used. Neuroph is an open source, lightweight java neural network framework used to develop common neural network architectures, (Sevarac, 2014). It is built on the netbeans platform. Figure 4.13 illustrates the artificial neural network architecture used. It is run periodically for offline validation to compliment the online validation performed by EM clustering at runtime due to the heavy constraint it places on system resources.

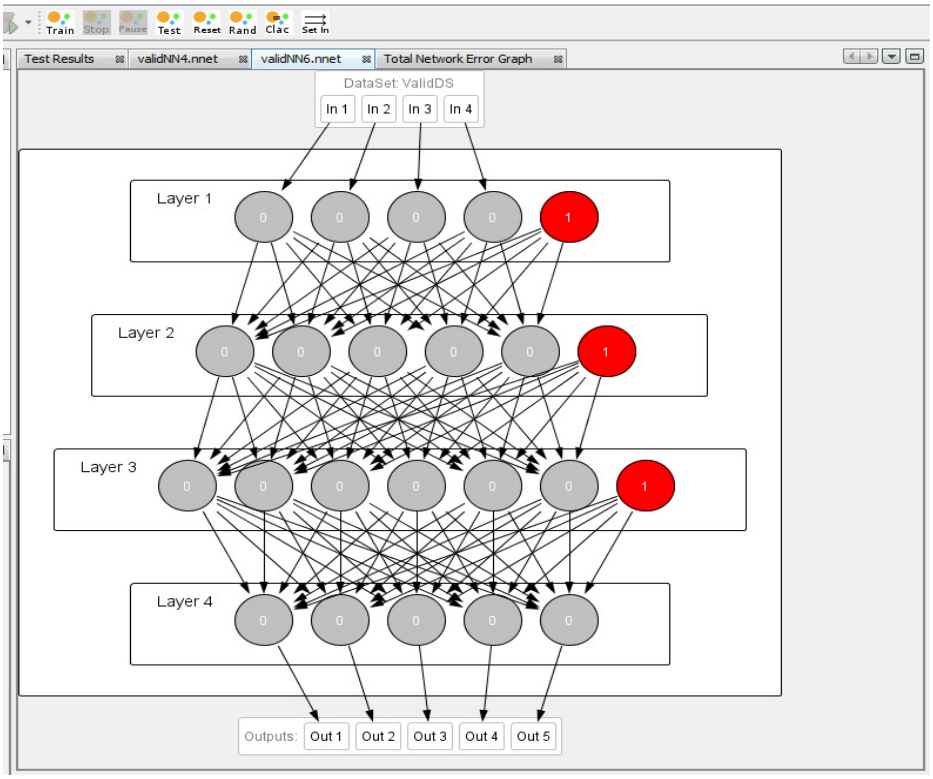


Figure 4.13 Neuroph Studio Artificial Neural Network Architecture

4.4. Summary

This chapter discussed the architecture and implementation of the proposed self-validating system for the dynamic adaptation of service oriented systems. The key components of the framework design are the application context, sensor, adaptation, and validation sub-systems. The role of each of these components in the framework was highlighted.

Specifically the application context comprises of a service oriented system made up of a service provider that creates and publishes the web service; a service registry who supplies the service information; and a service client who requests for the web service based on information found on the registry. Adaptation is triggered when a service request is made. The sensor component then executes to monitor the service quality, runtime environment, and business environment. The monitored information is then passed on to the adaptation component, which is responsible for decision-making. It vets the information against predefined threshold values to determine whether adaptation should take place. Because adaptation triggers can overlap and sometimes conflict, the *Simple Multi-Attribute Rating Technique (SMART)* is used to support negotiation through a process of trade-off analysis. If these threshold values have not been exceeded then the service

request is granted and the user is able to use the application. Alternatively if the values have been exceeded then the adaptation takes place based on predefined adaptation rules.

However a key contribution of this work is that adaptation rules change over time and subsequently should evolve in order to remain relevant. To achieve this, the validation sub system is then invoked before adaptation takes place. The primary role of the validation component is to evaluate the existing adaptation rules against logs of user preferences and determine if these rules are still relevant or need to change. Additionally the validation component autonomously updates the adaptation rules if they are not relevant. This validation takes place both at runtime using the EM clustering algorithm and statically using deep learning. Although EM clustering is very efficient in terms of utilizing system resources it is not as accurate. A more accurate algorithm, deep learning, is used to reinforce it. It is however executed offline due to its heavy demand on system resources.

These components are implemented as web services that are hosted on the cloud and are independent of the application context. As a result they are flexible, pluggable and portable. The framework is designed to support a variety of application contexts, triggers, and adaptation techniques. Further it can self-validate and evolve to take care of changing user needs as well as support a variety of learning approaches.

The chapter also examined the various tools used to implement the framework proposed. A web application interface is developed using HTML, JSP, and Servlets. An SOA system that has a client server architecture was created using java web services. The data was stored using XML, WEKA, and CSV files. WEKA machine learning algorithms were imported to develop the online validation module. Neuroph Studio, an open source, lightweight java neural network framework, was used for static validation.

Chapter 5

5. Evaluation

This chapter presents an evaluation of the validation framework using a series of experiments that test the objectives set out in Section 1.3. The evaluation uses a medium-sized case study based on an online health information portal. As part of the evaluation a service-oriented system layer was developed on top of the health portal as shown in Figure 4.1 to allow the portal to act as one of many service providers. Portal users (service consumers) access the service via mobile and fixed devices. The consumer node is interfaced to the validation framework running on a cloud server. A number of visualisation tools are used to monitor and analyse the framework behaviour at runtime, including a hardware resource profiler and the *Waikato Environment for Knowledge Analysis* tool (Weka).

5.1 Evaluation Techniques

There are many methodologies for evaluating a software system (Gediga and Hamborg, 2001). This section briefly outlines the different ways, in which software can be evaluated, and the benefits and drawbacks associated with each approach. A brief discussion of the suitability of the approach with regard the validation framework is provided. There are two common techniques used to evaluate software systems: descriptive and predictive evaluation techniques.

- *Descriptive evaluation techniques.* These are often used to gauge the status and problem space of a software system and are typically user-driven. There are three types of descriptive evaluation techniques:

  (i) *Behaviour-based* techniques record the actions and behaviour of a user while using the system. The methods of data gathering include observational techniques (i.e. ethnography) and user descriptive methods such as the *thinking-aloud* protocol.

  (ii) *Opinion-based* techniques are targeted at the elicitation of a user's opinion through different media, including interviews, surveys, and questionnaires. It is important to note that the majority of the data gathered by these means is subjective in nature.

(iii) *Usability testing* relies on using a combination of behavioural and opinion-based techniques to evaluate a system. Normally, this testing involves some level of experimental control, which is chosen by the expert/developer of the system.

- *Predictive evaluation techniques*. The primary aim of these techniques is to elicit recommendations and future requirements for the development of the software system. Unlike descriptive-based evaluation approaches, these methods often require the intervention of experts, and include walkthroughs and inspection-based techniques. Another important distinction is their reliance on data, which is often generated in a reliable and predictable manner by an expert user that simulates the interaction of a *real* user.

5.1.1. Evaluation Justification

Given the different ways in which a software system can be evaluated, it was important to decide on an evaluation technique that helped achieve the measurement of criteria within the constraints of the proposed approach in this thesis. Before an evaluation technique could be chosen, it was essential that a set of criteria was chosen; the criteria could be used to determine which technique would likely yield the type of data required to evaluate the validation framework. As such, it is important to revisit the research objectives stated in Section 1.3.

1. Provide runtime support for user-centred validation in self-adaptive service-oriented systems.
2. Provide support for managing overlapping and conflicting change triggers
3. Provide self-learning mechanisms to support and improve the predictive accuracy of runtime adaptation.
4. Provide support for mitigating framework resource overheads

The primary motivation of these objectives was to establish whether it is possible to improve the relevance and predictive accuracy of runtime adaption in service-oriented systems through user-centred validation and self-learning. Objective 1 was intended to address the lack of support for validation in current adaptation frameworks. As mentioned in 4.1.3 a typical adaptation process uses a predefined decision model to select an appropriate adaptation in response to a change trigger. This relationship is often stored as a set of fixed adaptation rules. However, the

dynamic nature of service-oriented systems 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 change requirements. This means that rules used to inform adaptation decisions cannot be static and must constantly evolve to remain relevant. One way to address the problem is through validation of adaptation decisions. The evaluation of this objective is intended to assess the impact of validation on runtime adaptation, in particular how it improves the quality of adaptation.

Objective 2 was intended to address the poor support for managing conflicting change requirements in current adaptation frameworks. The intention of evaluating this objective is to assess how effectively the framework resolved conflicting and overlapping triggers. As was noted in Section 2, change triggers are not mutually exclusive; there is often significant overlap between them. For example, a low memory trigger may be the result of an SLA violation (i.e. a service with higher than requested memory requirement) or runtime environment resource failure. An effective adaptation process should be able to establish the actual cause of the change trigger and provide a 'best fit' resolution. This might require some form of trade-off or negotiation. For example, the low memory trigger problem might require replacing the service with a more efficient alternative or optimizing the runtime environment or both depending on the cause.

Objective 3 was intended to address the poor support for predictive adaptation in service-oriented systems. Combining consumer-centred validation with self-learning provides a novel mechanism for tailoring adaptation rules to ensure that adaptation decision remain relevant and reflect user expectations. The evaluation of this objective examines two key issues:

- First, it assesses the speed and accuracy of the framework in predicting adaptation based on an EM clustering algorithm.
- Secondly, it assesses the extent to which complimenting shallow learning (i.e. EM clustering algorithm) with deep learning improves the framework's predictive accuracy.

Objective 4 was intended to explore ways of mitigating the high resource demands of deep learning. Deep learning models conventionally demand a level of system resources (e.g., memory and computation) that makes them problematic to run directly on constrained devices. The proposed solution was to employ deep learning *offline*, using it to periodically examine validation

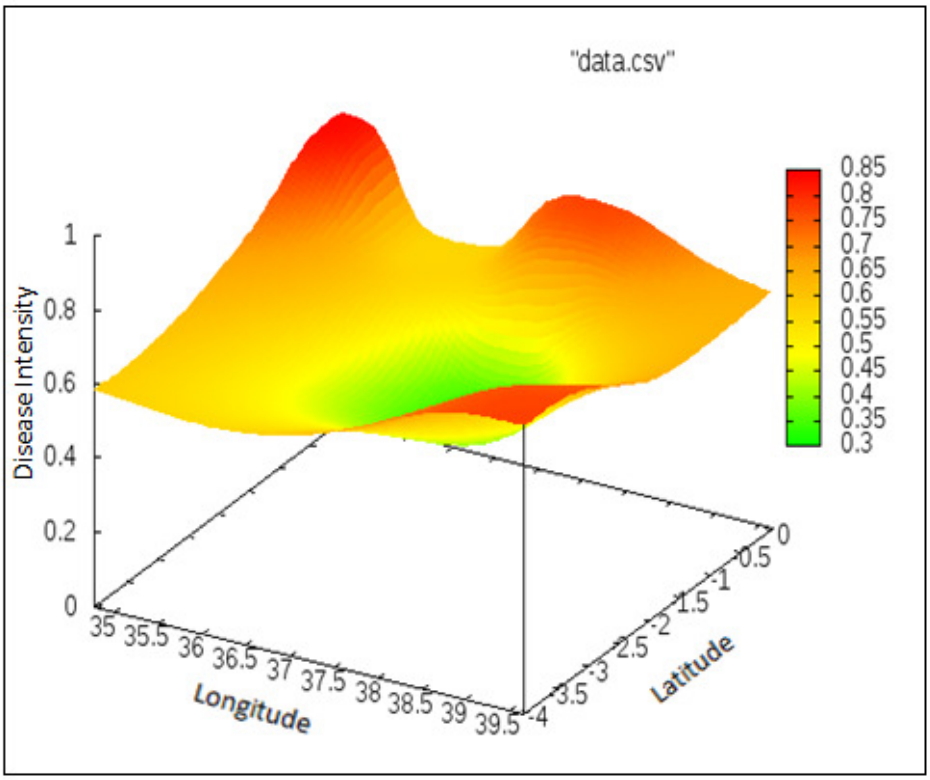
logs and update the adaptation rules. The evaluation of this objective assesses the efficiency of the proposed solution.

The dynamic nature of the problem and lack of similar studies to baseline the evaluation support the choice of a predictive evaluation technique. This approach allows for complex application scenarios and configurations to be evaluated using simulated data.

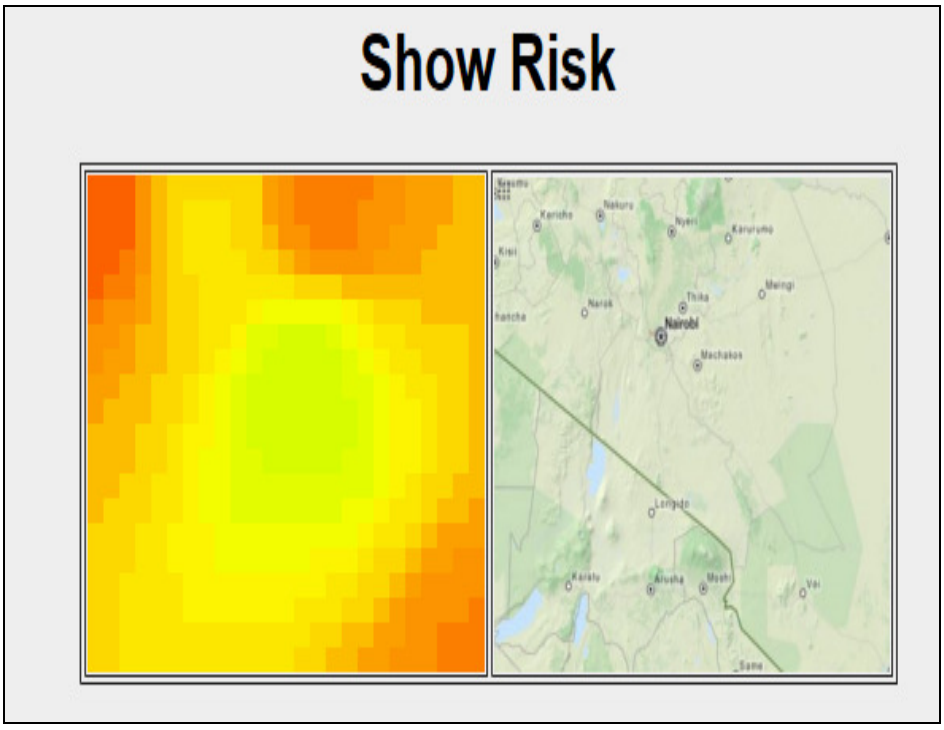
5.2 Case Study Overview

In Kenya health practitioners take a reactive approach to disease outbreaks where in many cases a disease is only detected after it has spiraled out of control. The proposed solution would therefore provide health practitioners with an early warning of infectious disease outbreaks to enable them to manage such outbreaks before they become catastrophic. Additionally the government can also deploy resources to such areas hence curbing the outbreak with minimum resources. The solution would also increase awareness amongst citizens as well as foreigners intending to visit a location so that they can take the necessary precautions and minimize the spread of infectious diseases.

A free available Javascript tool, *EWARS Spatial Fuzzy Logic Demo,* is used to visualize the results based on the work of Platz *et al.* (Platz *et al.*, 2014). Figure 5.1 (a) gives the interpolated spatial data as generated on a graph. It shows public interest in Typhoid disease information. The same results are also shown on the map alongside in Figure 5.1 (b). The results shows that in some regions of Kenya, citizens are interested in information on Typhoid (in red), for example, in the eastern, central and south-eastern parts of the country. This is attributed to the fact that different geographical regions are affected by different health issues and have different needs.



(a) Typhoid Disease Intensity



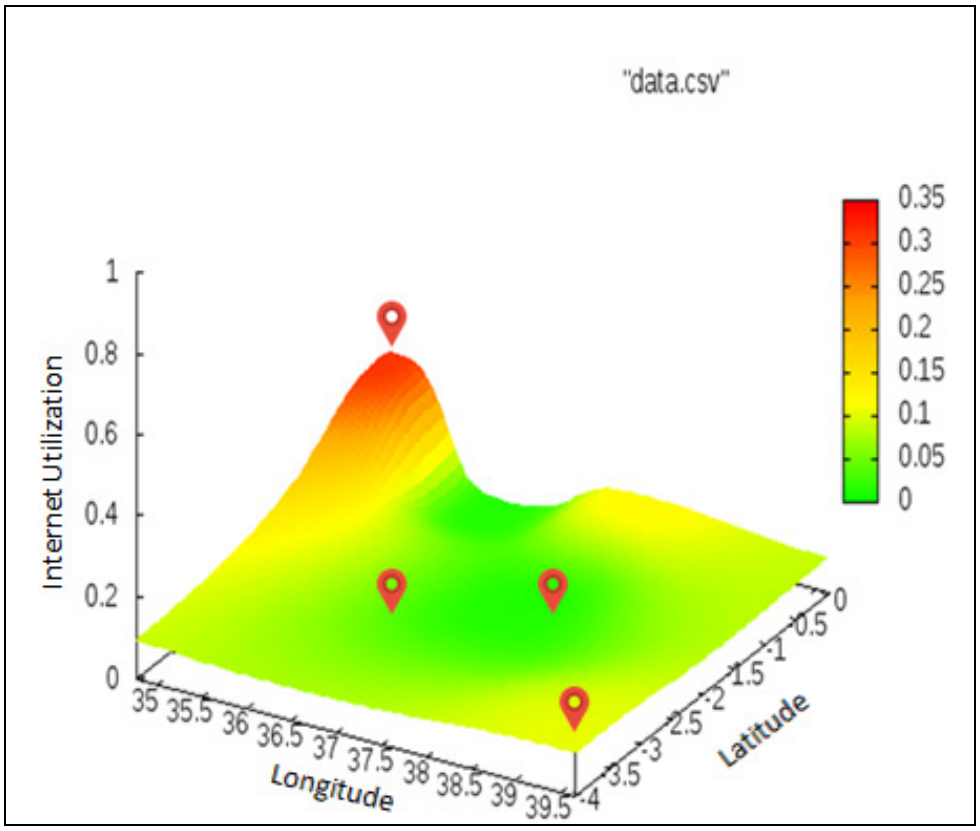
(b) Disease risk map

Figure 5.1 Interest in Information on Typhoid Disease

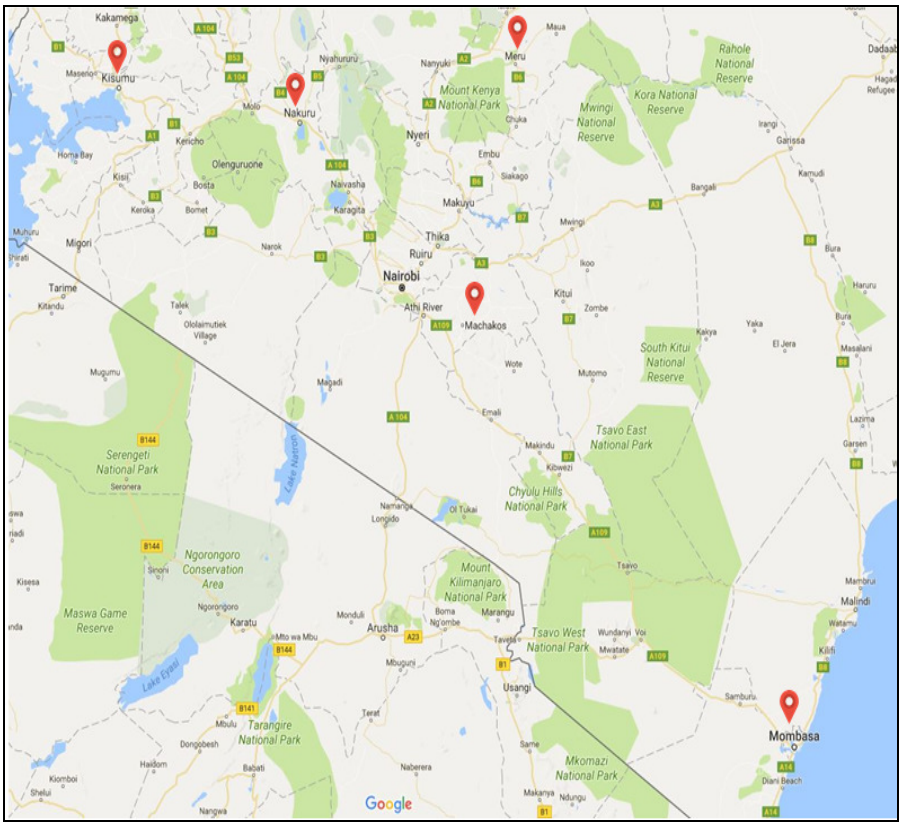
This information was used a basis for the initial adaptation rules. Five experiments were devised to evaluate the effectiveness of the prototype framework. As part of the evaluation a service-oriented system layer was developed on top of the health portal as shown in Figure 4.1. The broker uses a GPS or IP service to look up the clients' location, which is then passed on to a disease mapping service to retrieve information on prevalent diseases in the location. The client application comprises of a webpage running on a mobile device or a desktop PC and a stub that is responsible for collecting information about the user's runtime platform. For the experiments three devices were used which include a mobile phone, a laptop, and a desktop PC. The choice of devices was motivated by the fact that in Kenya most users access Internet applications through these devices. According to the communications Authority of Kenya (CAK, 2015) 88% of the population own mobile phones. Further 99% of Internet users accessed it via a mobile phone. Statistics from the Kenya National Bureau of Statistics (KNBS, 2017) show that less than 3% of the population has access to computer services. The percentage of the population that has access to mobile tablets is even less. In most government and health institutions personal computers are used while citizens use mobile phones. The fact that the key beneficiaries of the system would access it through either a mobile phones or a personal computer, motivated the experiments to be conducted using similar devices. The rest of the prototype comprises of the user application (Health System) and the framework (sensors, adaptation, and validation services), hosted on a cloud platform.

5.3 Experiment 1 – Impact of Validation on Runtime Adaptation

The aim of this experiment is twofold; (i) to assess the framework's ability to self-validate and modify adaptation rules at runtime and (ii) to assess the impact of validation on the quality of adaptation. Figure 5.2 shows how geographic location influences the use of Internet to access health related information in four locations in Kenya. The figure shows that users from affluent cities such as *Mombasa* on the east coast and *Kisumu* in the west make use of Internet services more than users from marginalized locations such as *Machakos* and *Meru* despite the fact that all the locations have reliable internet infrastructure. This could be attributed to the fact that users in the more affluent locations can afford the costs associated with access to these services.



(a) Use of internet to access health information



(b) Geographical location of cases used

Figure 5.2 Access of health related information using Internet services

The findings were used as a basis for formulating the initial adaptation rules. However user needs are not static due the dynamic nature of the environment they live in. For example, other studies (Namuye & Mutanu, 2014) have noted that patterns of interest in diseases can change suddenly following disease outbreaks and health emergencies in country. The challenge is to ensure that adaptation rules evolve to reflect this change in interest. The validation component uses an EM clustering machine-learning algorithm to identify changing requirements for different geo-locations. The framework incorporates an option for the user to select alternative disease information if they are not satisfied with the information provided by the existing adaptation rules - i.e. perform a custom search. The user's decision is logged and used as a training data set to inform future adaptation rules. The behavior of the framework in the presence of predefined adaptation rules as well as when user requirements changed is described next.

5.1.2. Predefined Adaptation Rules

The framework was tested using 46 simulated cases as the training data set and a number of clusters and their attributes were generated by the validation component as seen in the web services response in Figure 5.3. As part of the simulation exercise "noise" data is deliberately included, i.e. sporadic requests for diseases rarely found in the tropics. The aim of this was to assess how effectively the validation process dealt with "noise" data.

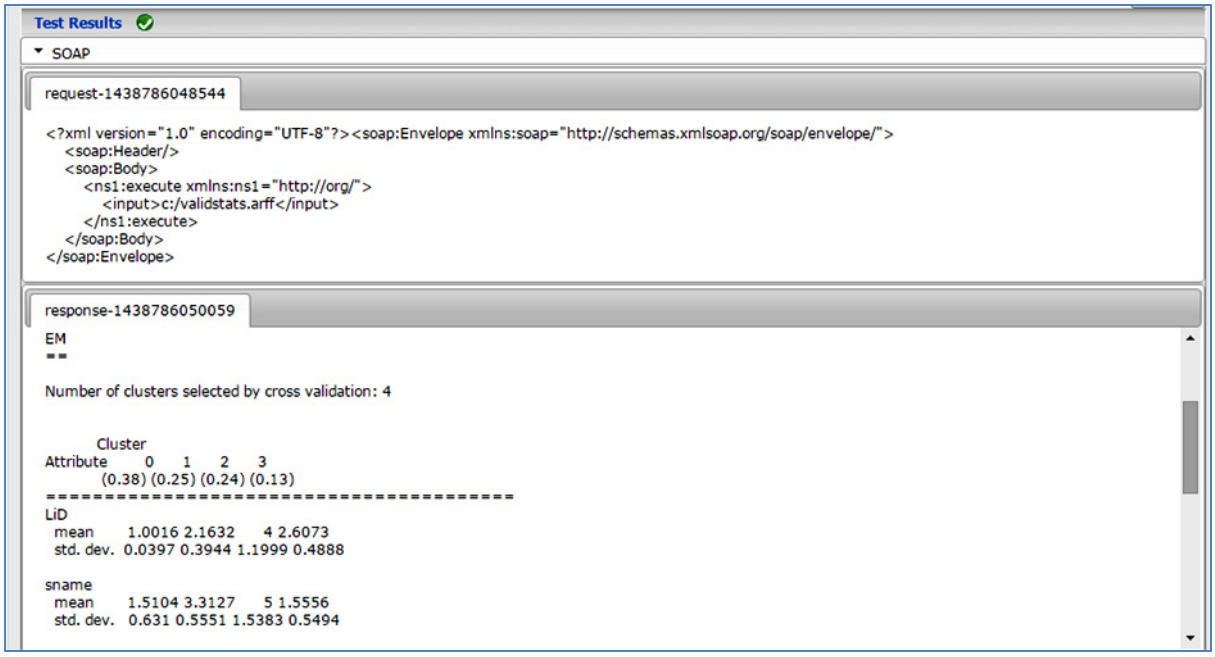


Figure 5.3 Web Service request and response

The same data set was run on the *Waikato Environment for Knowledge Analysis* (Weka) tool (Hall *et al,* 2009) to independently verify the decision generated by framework. A visualization of the clusters produced by Weka confirmed that the decision generated by the framework was accurate as shown in Figure 5.4. Four different clusters formed representing the different disease information retrieved from the different locations. This was similar to the four clusters generated by the validation web service shown in Figure 5.4.

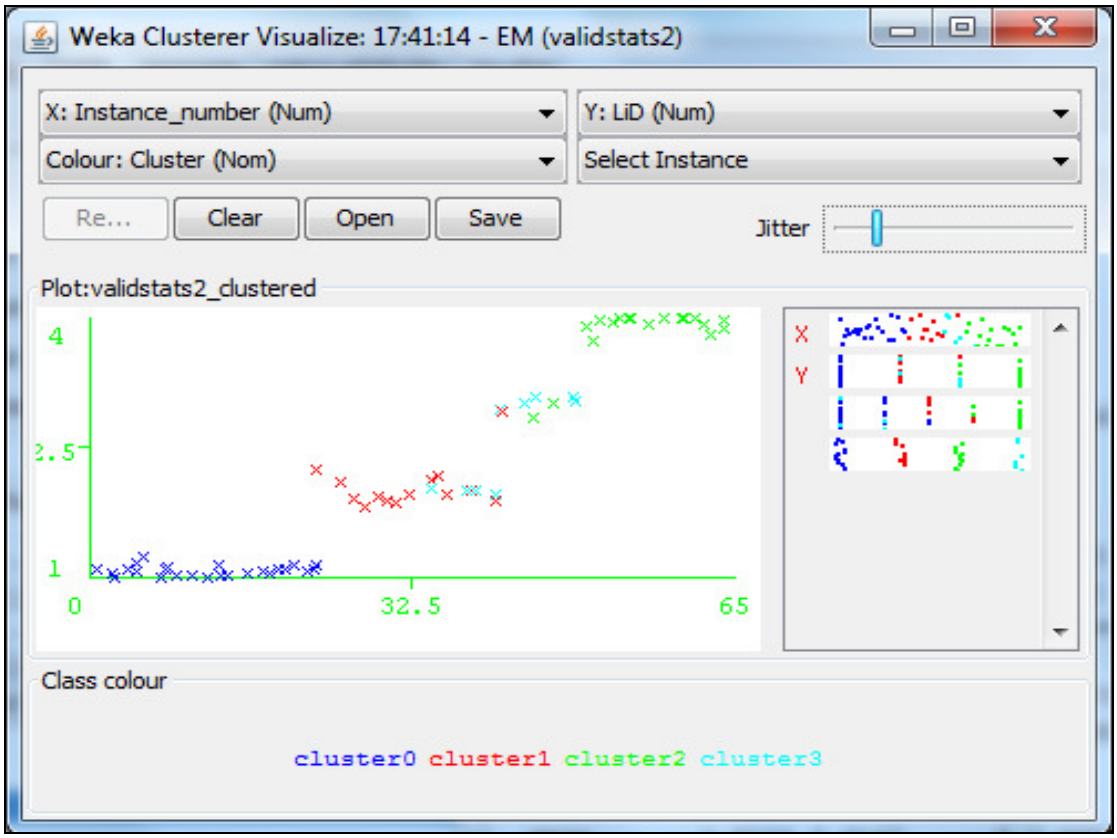
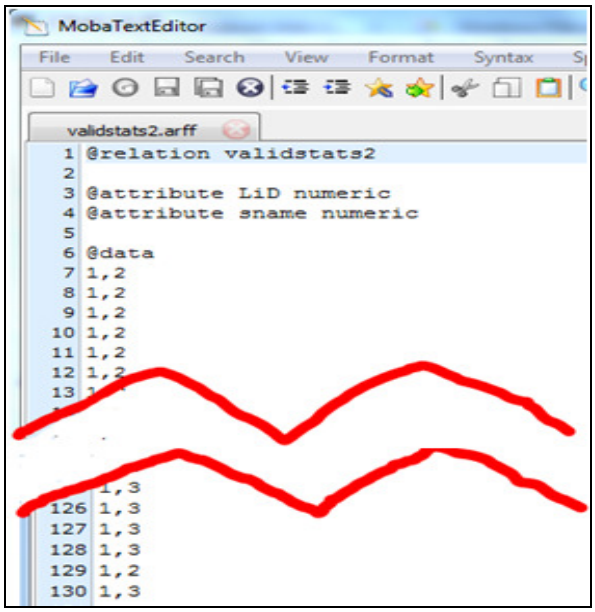


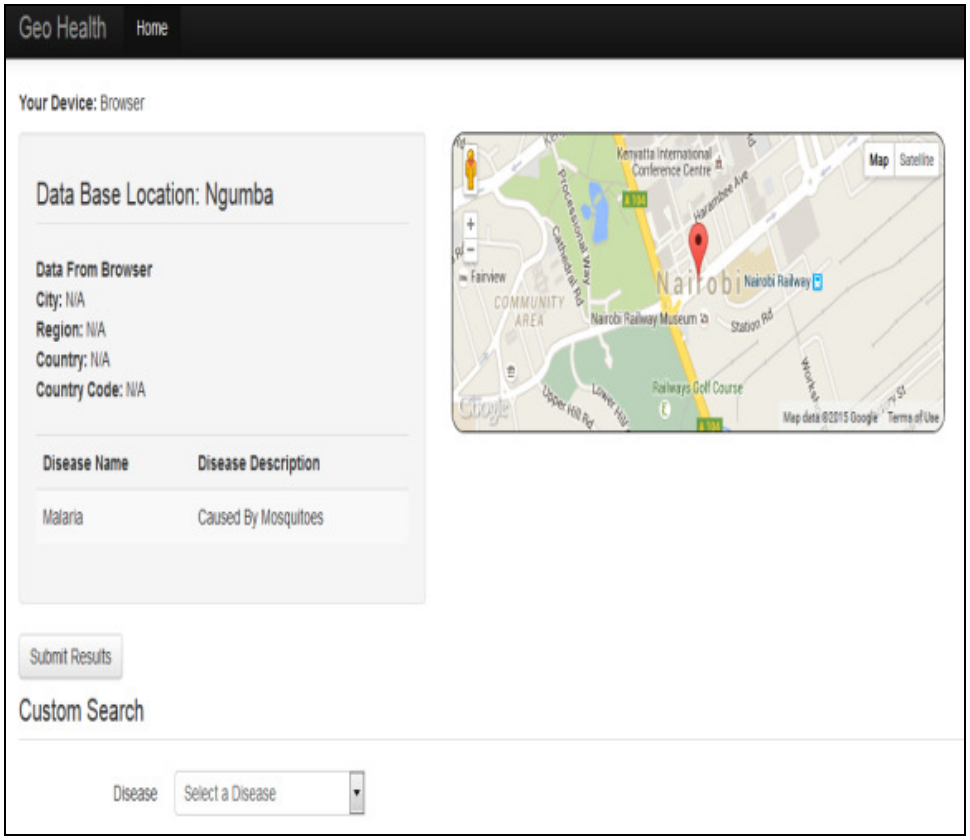
Figure 5.4 Verifying output of the validation framework

5.1.3. Changing Adaptation Rules

The learning component was tested in a controlled environment by repeatedly searching for a disease (i.e. *Malaria*) that is different from the one suggested by the predefined adaptation rules (i.e. *Common Cold*). Sporadic searches for diseases such as *influenza* and *skin cancer* were also included. Clustering generally works on frequencies of the decision logs. As a result an increase in the number of logs searching for Malaria for the region results in the adaptation rules changing. This can be seen in Figure 5.5 where an increase in the logs suggesting *Malaria* (1,3) instead of *Common Cold* (1,2) resulted in a change in the adaptation rules and subsequently the framework started suggesting M*alaria*.



(a) Change in the validation logs



(c) Adapted System

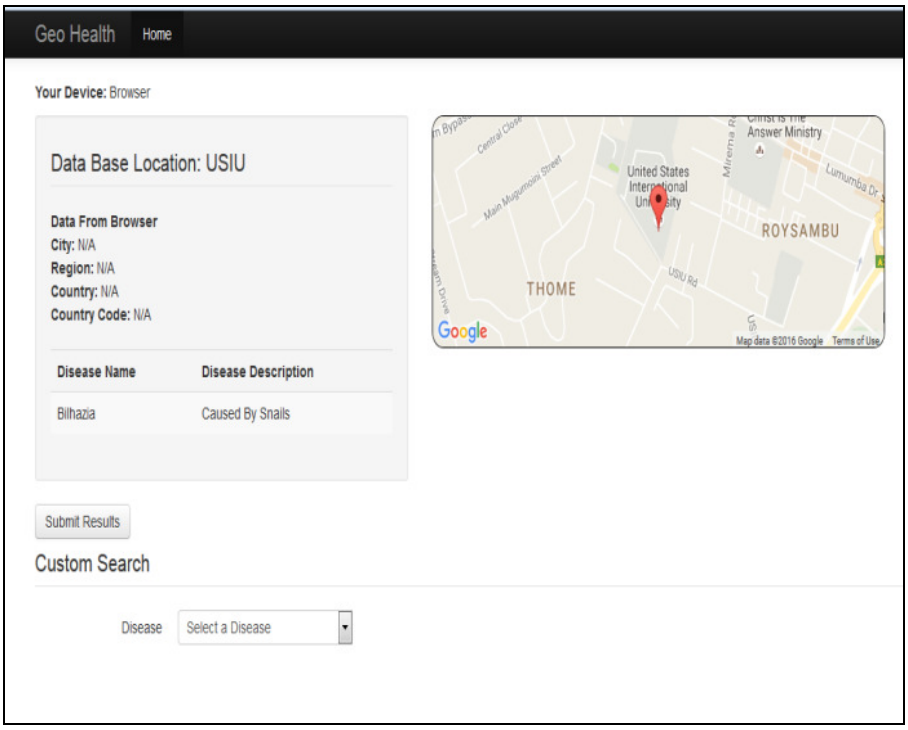
Figure 5.5 How validation statistics influence adaptation rules

The results demonstrated that the application was not only able to self-adapt in response to external environment triggers, but also to self-validate through a learning process and subsequently update the adaptation rules. The services adapt in response to environmental triggers to provide results that are relevant to the user without the users intervention.

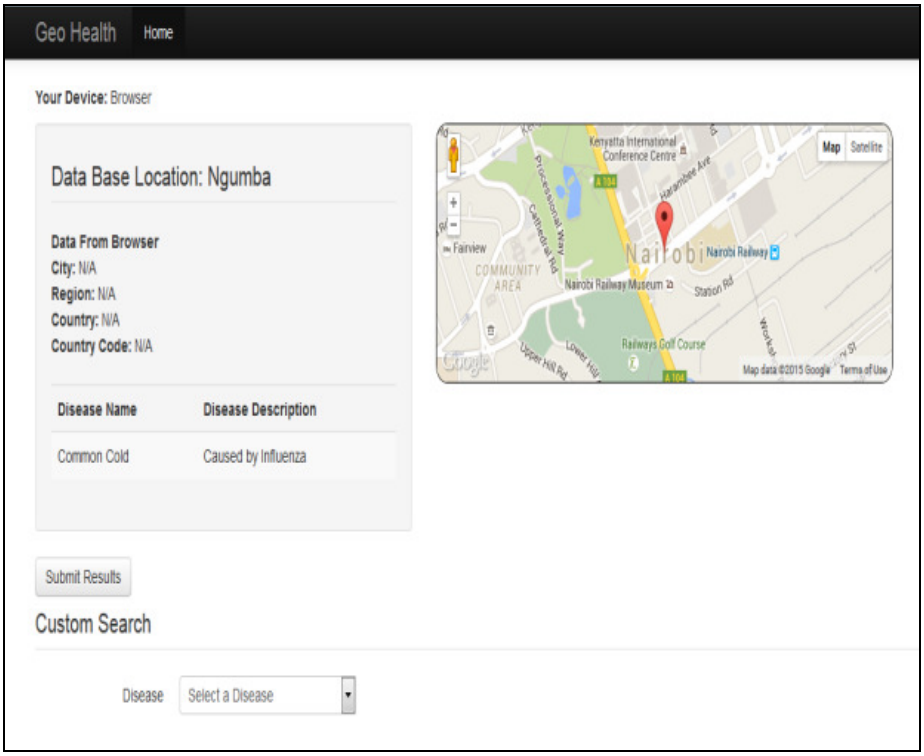
5.4 Experiment 2 - Managing Overlapping and Conflicting Change Triggers

The aim of this experiment was to assess how the framework dealt with overlapping and conflicting change triggers. The experiment explored the nature and significance of adaptation conflicts that might arise as a result of competing service provider reputation and service cost goals. The experiment used the geo-location data shown in Figure 5.2. Service cost and reputation often present conflicting goals given that a service whose source has a high reputation is likely to cost more than one whose source has a low reputation. An assumption was made that affluent geographical locations favour a high reputation over cost and while marginalized locations prefer low cost services. When the system runs, it begins by detecting the user's geo-location and then, based on existing adaptation rules, retrieves the service cost and the reputation required. To ensure that the system was able to display this information to the user before retrieving the service, the service attributes such as the reputation and cost were described in a service registry. An independent web service was used to retrieve the information. The framework was exposed to both conflicting and non-conflicting triggers as part of the evaluation. For example, to simulate non-conflicting triggers, a business environment trigger (geo-location) and a runtime environment trigger (device type) were selected.

Figure 5.6 shows how the prototype behaved when accessed from different locations. The application was able to retrieve different disease information for different geographic locations. The application uses a GPS service to look up the client's location, which is then passed on to a disease mapping service to retrieve information on prevalent diseases in the location. Geo-location acted as the external environment trigger, affecting the choice of information retrieved.



(a) Prototype in Location A



(b) Prototype in location B

Figure 5.6 Adaptation based on geo location trigger

From the results it can be observed that the user application was able to adapt based on business environment triggers. A non-competing runtime environment trigger was then introduced into application by adding a sensor that detects the device used to access the service-oriented system. The framework detects device types by querying the device operating system and the system resources such as the processor type and memory available for the Java Virtual Machine (JVM) to use. For this experiment the Java *Runtime* class, which allows the application to interface with the environment in which the application is running, was used. The class defines several methods for accessing system properties about the operating system on which the JVM is running. The specific properties accessed for this experiment were the number of processors `(Runtime.getRuntime().availableProcessors())` and the memory available `(Runtime.getRuntime().freeMemory()/(1024*1024))` for the JVM. These resources are evaluated against predetermined threshold values after which the appropriate workflow is selected. Figure 5.7 shows the JVM details obtained for two devices with different system specifications used for this test.

Figure 5.7 System properties returned from different devices

These properties act as an indicator of resource constrains in the system i.e. memory and CPU performance. For example if a processor with less than four cores or memory less than 52 MB of memory is detected, the framework assumes that the user device has limited resources. However, if the opposite occurs the framework assumes the user has adequate resources to access the application. To address the varied resource requirements the application reconfigures itself to make use a resource-sensitive workflow. The framework was able to reconfigure the application to use a workflow pattern that uses less memory when a mobile device was detected and to bring up the appropriate health information associated with the geographic location. Figure 5.6 shows that the device detected was also displayed for verification. Because users in specific geographic locations have similar lifestyles, they tend to own similar devices hence the choice of workflow pattern is closely associated with the location. Such triggers do not conflict or compete.

To simulate competing change triggers, nine services were created and "tagged" with various values for cost and reputation. The system was seen to adapt dynamically to suggest a service

with a high reputation <*rep=3*> when accessed from an affluent location as shown in Figure 5.8. Further the user could accept or reject this decision as shown in the Figure.

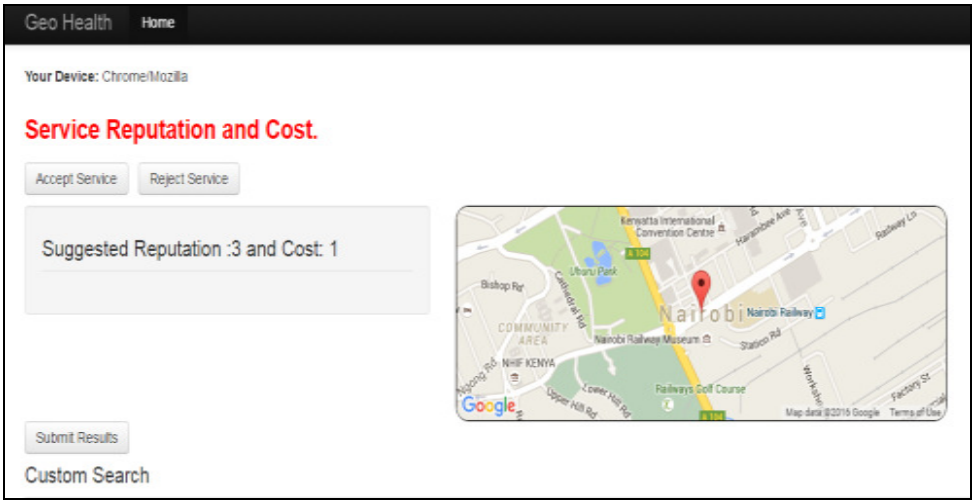


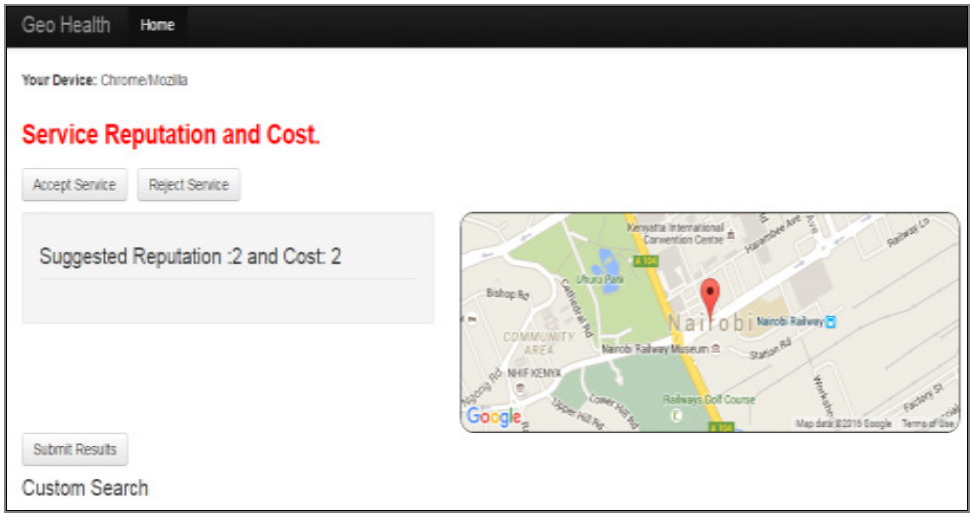
Figure 5.8 Accepts/Rejects options of suggested service

5.3.1. Trigger Negotiation

When the framework encounters conflicting goals, a negotiation process is initiated to try and find an acceptable compromise. This is achieved by optimizing one goal at the expense of another incrementally until an acceptable tradeoff is arrived at. In this case, the final outcome might be a slightly cheaper service with a lower, but acceptable provider reputation. If the user rejects the trade-off decision, the negotiation is repeated to find a new compromise. However, given the conflicting nature of these qualities and varied consumer expectations, it is impossible to create adaptation rules that can identify a service that satisfies all consumer requirements. However, through a process of validation and machine learning, suitable adaptation rules can be constructed to find the closest match as demonstrated in *Experiment 1*. If the user cannot find an acceptable service the required information is not retrieved, and the application does not run. The final decision (acceptable service or no service) is logged as part of the framework's validation statistics. The validator uses the logs as part of it's training data to inform future adaptation rules using an EM clustering algorithm.

To gauge how well the framework has learnt, the suggested service was repeatedly rejected for the negotiated one. Figure 5.9 (a) shows that after some time the system started displaying the negotiated value as the default value from the adaptation rules, indicating that the adaptation rules had changed. The web service output in Figure 5.9 (b) confirms that the cluster formed for

location 4 (`Lid=3.5516`) is shifting from the default value of 3 for reputation towards 2 (`Rep=2.5316`). The system is able to learn the acceptable compromise and change the adaptation rules accordingly.



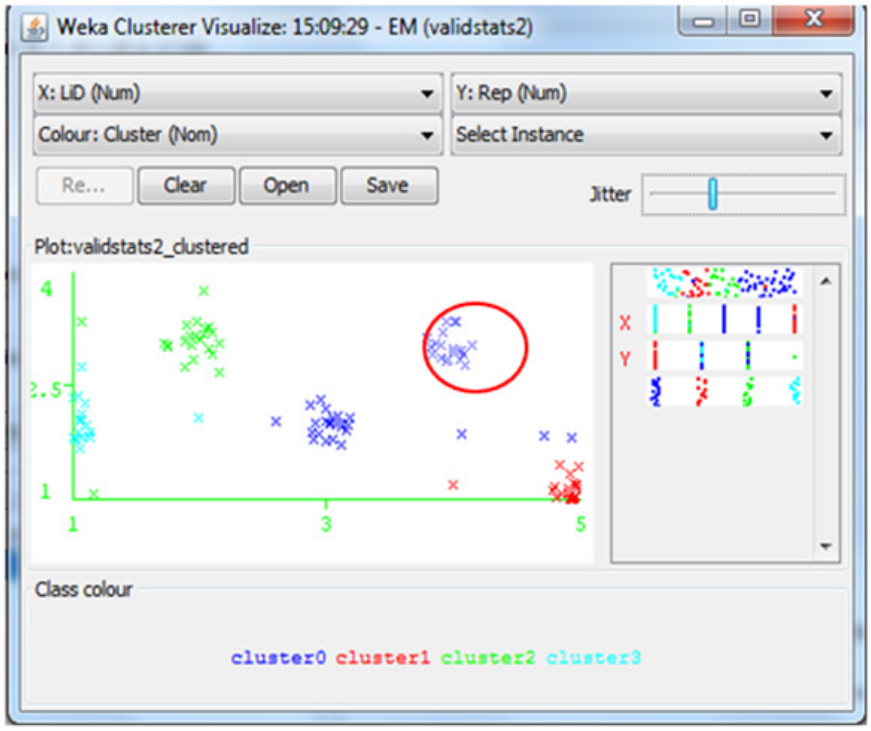
(a) Resultant new adaptation decision

```
EM
==

Number of clusters selected by cross validation: 4


            Cluster
Attribute        0      1      2      3
             (0.37)   (0.2)  (0.19)  (0.24)
=========================================
LiD
  mean       3.5516 4.7936      2 1.4343
  std. dev.  0.6479 0.8086 1.4174 0.8687

Rep
  mean       2.5316      1      3 2.0256
  std. dev.  0.5515  0.003 0.7606 0.1824
```

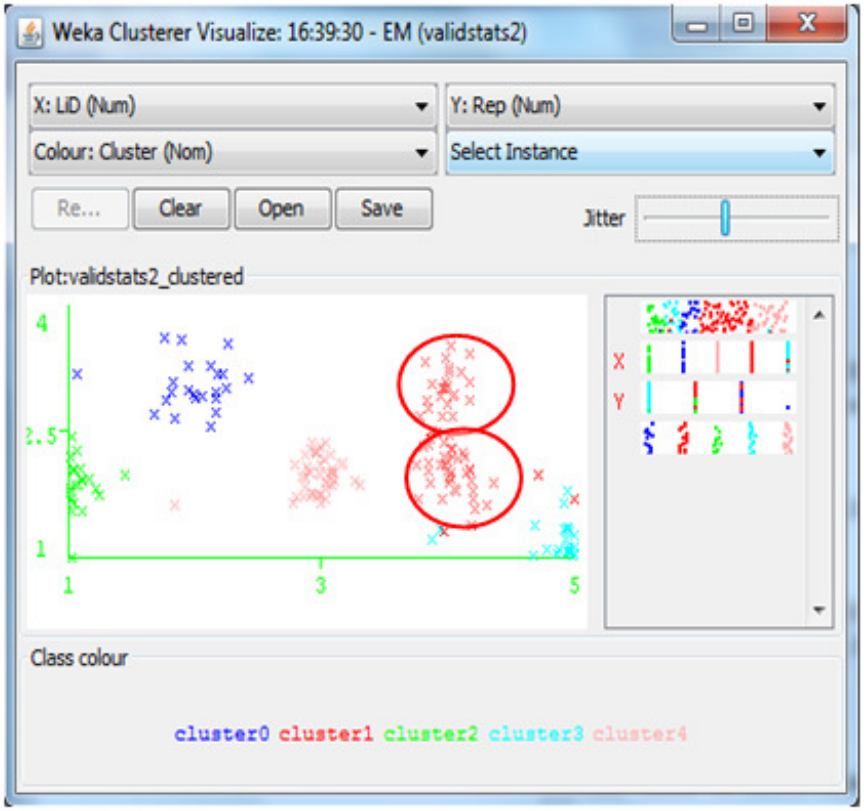
(b) New clusters from validation service

Figure 5.9 New rules generated from negotiated decisions

A visualization of the validation statistics via *Weka* show how the clusters changed following negotiation based on user feedback. Figure 5.10 (a) shows the clusters that informed the previous adaptation rules. Following repeated negotiation the formation of a new cluster with a higher density than the old one eventually gave way to new adaptation rules, as shown in Figure 5.10 (b).



(a) Cluster based on previous rules



(b) New higher density cluster formation

Figure 5.10 Evaluating the validation output using *Weka*

During this time the geo-location sensor was still active and users were still able to accept or reject the adaptation decision made based on the location trigger. The framework is therefore able to handle overlapping triggers and support a negotiation process where there are conflicting triggers. Creating adaptation rules for conflicting triggers is not an easy task without factoring in how users arrive at a compromise. The use of machine learning to observe and learn how users arrive at a compromise is therefore critical where adaptation triggers are conflicting. Self-validation therefore presents a novel solution to the problem of multiple conflicting adaptation goals.

5.5 Experiment 3 – Assessing Framework Accuracy

The aim of this experiment was to assess how fast and accurately validation helped to adapt the content and the quality of health information to changing user needs or demands. As part of the experiment, data on diseases relevant to four geographic locations, Mombasa, Kisumu, Machakos, and Meru was used in the experiment. The data was drawn from a Kenya national survey on health (Kenya National Bureau of Statistics, 2015).

5.4.1. Measuring System Stability

This experiment simulated seven users attempting to access the system from the four different geographic locations. Once validation has occurred, a decision on whether the adaptation rules are still accurate or need to be up dated is arrived at after which the rules can either be maintained or changed. This decision is informed by feedback from the user, which is in the form of acceptance or rejection of the decision based on adaptation rules as explained in *Experiment 1*. The feedback then acts as a trigger that adapts the adaptation rules. This trigger indicates that the systems decision is inadequate. If the triggers occur frequently the adaptation rules keep changing and the system is unstable during this time because the user keeps rejecting the adaptation decision. The goal of the learning component is to understand the user's current needs and change the adaptation rules accordingly to reach a stable state. A stable state is denoted by a reduction in the number of changes that call for modification of the adaptation rules in a given time interval. This is used to determine the accuracy of the validation process.

In order to measure stability the frequency with which the user accepted decision changed within a given time period is determined. Frequent changes would indicate that the system is

unstable and therefore the validation component is not working as effectively as it is required to. The goal of the learning component is to understand the users' current needs and change the adaptation rules accordingly to reach a stable state. A stable state is denoted by a reduction in the number of changes that call for modification of the adaptation rules in a given time interval. This is used to determine the accuracy of the validation process.

5.4.2. Accuracy of the Validation Framework

From the validation logs populated by the system the requirement change frequency ($\Sigma\Delta/[t_1,t_2]$) for a given time interval ($[t_1,t_2]=\{x|t_1<x<t_2\}$) for different locations is calculated. Thirty time intervals (five days for each interval) are picked and used to chart the system behavior as shown in Figure 5.11. The results show that in *Location A* and *Location B* the system started out fairly unstable with the user's frequently rejecting adaptation decisions. However, with time the system is able to learn and the number of changes reduces significantly. In *Location C* the system started out fairly stable but at the 11^{th} time interval users started rejecting adaptation decisions. However, the system is seen to start learning and eventually stabilizes. In *Location D* the system remains fairly stable throughout.

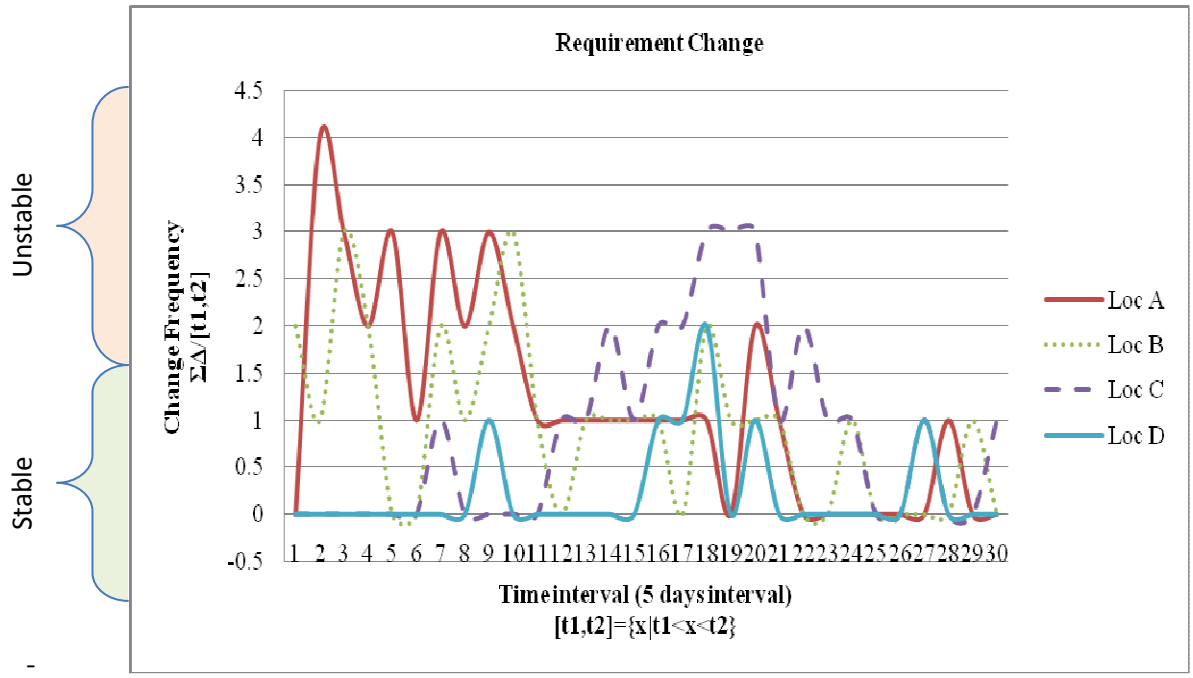


Figure 5.11 Framework Accuracy

The systems stability can be used to gauge the accuracy of the self-validating framework. Because of the accuracy of the deep learning algorithm, the system is satisfactorily able to reach a

stable state. If the system however is unable to reach a stable state, then this is an indication that the learning algorithm is not learning as effectively as it should. An alternative learning algorithm can then be selected. The framework can work with a variety of learning algorithms as described in the experiments. Because the validation component has been implemented using an independent web service, any learning algorithm can be developed as a web service and plugged into the framework.

5.6 Experiment 4 – Improving Framework Accuracy

Experiment 2 provided a practical verification of the framework's accuracy and flexibility. The experiment used an EM clustering algorithm to mine logs on the final user accepted adaptation decision and identify patterns that indicate the changing user requirements. These patterns were used to update the existing adaptation rules and inform future adaptation decisions. However, results from the experiment also revealed three problematic scenarios that are typical of natural data, i.e. the existence of no clusters, similar clusters, or several different clusters. Such scenarios made it difficult for the clustering machine learning algorithm to arrive at a decision. To address this, the clustering algorithm in *Experiment 2* was adjusted to handle the three scenarios in the following manner:

- *Several different clusters exist* – This occurs when a fraction of the population is interested in one disease and another fraction in a second disease. The cluster density was factored such that the highest cluster density was selected to inform the adaptation rules.
- *Similar clusters* exist - If the cluster density is the same then other factors need to be factored in to identify the predominant cluster, such as the existing adaptation rules or the time stamp of the logs. This occurs when the users preferences are changing and their new needs are about to overtake the old needs. It is a period of uncertainty. To handle this uncertainty the existing adaptation rules were made to prevail until such a time when a predominant cluster emerged.
- *No clusters exist*– This will occur because no specific disease is recurrently searched for. For such a scenario the existing adaptation rules were used to provide the default disease for the region. Default diseases were selected based on the results of an earlier government funded survey described in section 5.2.

Where any of these scenarios existed, the accuracy of the framework became questionable. There was need for an alternative way to address the problem in a bid to improve on the accuracy of self-validation. This experiment describes how *deep learning* was used to address the problem and to improve the accuracy the validation process.

5.5.1. Neural Network Training

A key concept underlying Deep Learning methods is the use of distributed representations of the data, in which a large number of possible configurations of the abstract features of the input data are feasible. The training set has therefore to be carefully selected to allow for a compact representation of each sample, leading to a richer generalization. Because deep learning calls for several input features, both the environmental and business triggers described earlier were combined as the input features for the neural network. These features are Memory Capacity, Processor Speed, Device type, and Geo-location.

A multi-layer ANN architecture is made up of an input layer, one or more intermediate or hidden layers, and an output layer. Additionally several neurons with several inputs (x_n) and weights (w_n) exist. The action of selecting the correct weights is the process that results in the adaptation rules in the case study. The supervised learning approach is used, where the output for a set of inputs is given to train the network. Different supervised learning approaches, or learning rules, also exist for supervised algorithms such as error-correction, Boltzmann, Hebbian, and competitive approach. The Error-correction approach is used here, which focuses on the fact that the output perceived in the training process do not always correspond to the desired output. This error is known as the total mean square error (MSE) and is obtained as shown in *equation 5.1*. It is computed using all training patterns of the calculated and target outputs (Islam et al., 2010).

$$MSE = \frac{1}{2}\sum_{j=1}^{m}\sum_{i=1}^{k}\left(T_{ij} - O_{ij}\right)^2 \qquad (5.1)$$

Where m is the number of examples in the training set, k is the number of output units, T_{ij} is the target output value of the i^{th} output unit for the j^{th} training example, and O^{ij} is the actual real-valued output of the i^{th} output unit for the j^{th} training example.

The back-propagation algorithm uses an iterative gradient technique to minimize the MSE between the calculated output and the target output. The main idea is to initially move forward to compute the error, and then backward the error updating the weights from the output layer to the input. The method calculates the error, modifying the weights to progressively reduce the error. Initially the weights are set to random small numbers and progressively updated during training based on the calculated error. This is the basis of the back propagation learning algorithm. Figure 5.12 gives an example of the error rate values obtained during training of the neural network using the training dataset.

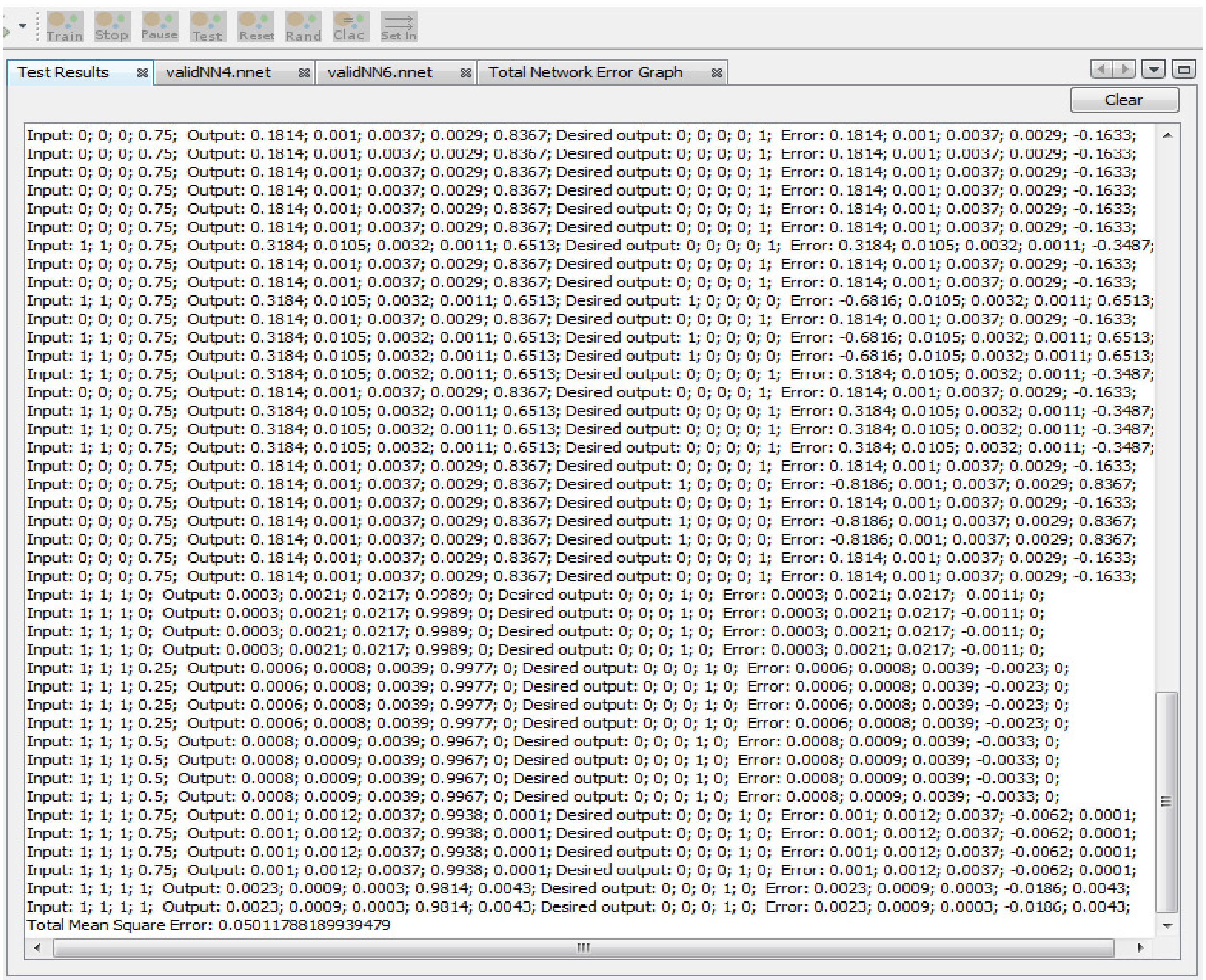


Figure 5.12 Total Mean Square Error Rates

Because a neural network is made up of layers and nodes that describe its architecture, it is a good idea to train several networks to ensure that a network with good generalization is found. The objective of the training is to find a network with the smallest *error* and *regularization* terms. The error term evaluates how a neural network fits the data set. It depends on adaptive parameters such as biases and weights. On the other hand, the regularization term is used to prevent *overfitting* (large difference between training and test error), by controlling the effective complexity of the neural network. To avoid overfitting Piotrowski *et al.*, (Piotrowski *et al.*, 2013) recommend keeping the ANN architecture relatively simple, as complex models are much more prone to overfitting. This is because the error on the training set is driven to a very small value, but when new data is presented to the network the error is large. The network has memorized the training examples, but it has not learned to generalize to new situations. The training set is kept relatively small and tests conducted to show the neural networks ability to generalize.

The process of training the dataset called for modifying several parameters such as the learning rate, momentum, randomizing the weights, as well as increasing the size of the training dataset to arrive at an acceptable error rate. When training an ANN initial weights are identified and updated as the input values move from one layer to another. The final output may not always be the expected output resulting in an error. Since the nature of the error space cannot be known a prioi, neural network analysis often requires a large number of individual runs to determine the best solution. Most learning rules have built-in mathematical terms to assist in this process which control the 'speed' (learning rate) and the ability (momentum) of the learning. The speed of learning is the size of the steps that the weights are changed by. This determines the speed by which a solution is arrived at. But the solution may not always be the best solution i.e. it may have an error. Momentum helps the network to arrive at solutions that are the best by editing the step sizes such that they are not always a constant value.

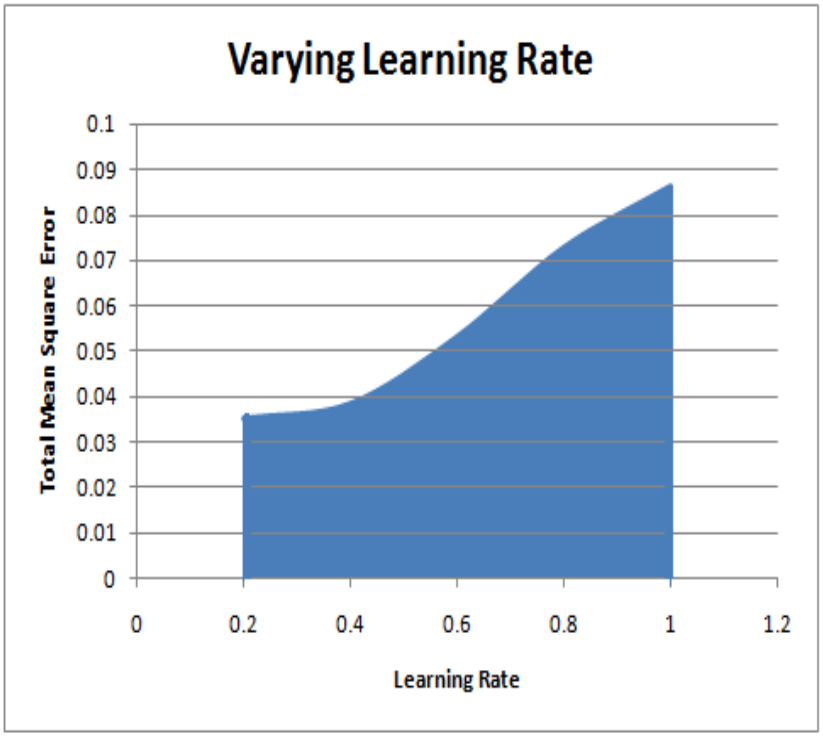
Momentum simply adds a fraction m of the previous weight update to the current one. When using back propagation with momentum in a network with n different weights w_1, w_2, ..., w_n, the i^{th} correction for weight w_k is given by:

$$\Delta w_k(i) = -\gamma \frac{\partial E}{\partial w_k} + \alpha \Delta w_k(i-1) \qquad (5.1)$$

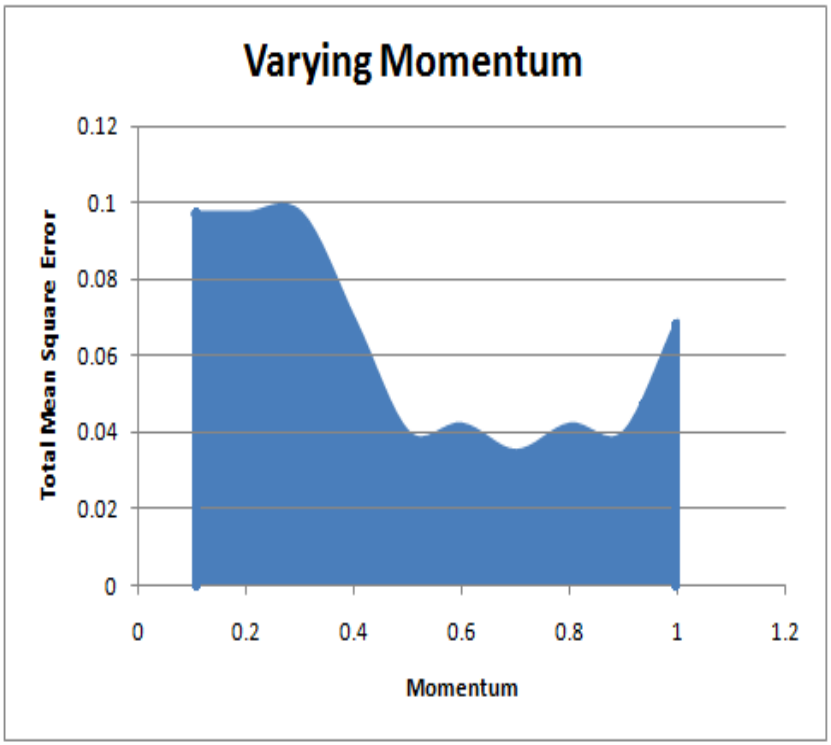
Where Υ and α are the learning and momentum rate respectively. The error function E is determined at the output. Momentum is a value between 0 and 1 that is randomly determined multiplied to the previous weight to form the new weight.

If both the momentum and learning rate are kept at large values, then you might miss the best solution (minimum) with a huge step. A small value of momentum on the other hand can arrive at a solution that is not the best (local minimum) because the step size remains relatively the same. It also slows down the training of the system. The ideal solution lies in keeping a larger momentum term and smaller learning rate. Once a neural network is 'trained' to a satisfactory level it may be used as an analytical tool on other data.

For this experiment the learning rate and momentums are adjusted during training. When the learning rate was increased, the total means square error also increased as shown in Figure 5.13 (a). This is because increasing the learning rate also increases network instability, with weight values oscillating erratically as they converge on a solution. The momentum rate prevents settling into a local minimum by skipping through it. As the momentum rate was varied the error increased when the rate was approaching extreme values i.e. 0.0 or 1.0 as shown in Figure 5.13 (b). This is because as the momentum rate approaches the maximum of 1.0 the training becomes unstable and thus may not achieve local minima, or if it does, it takes an inordinate amount of training time. On the other hand as it approaches 0.0, the momentum is not considered and the network is more likely to settle into a local minimum. Therefore a relatively low learning rate and a moderate momentum rate were ideal for the study.



(a) ANN training by varying learning rate



(b) ANN training by varying momentum rate

Figure 5.13 Training by varying learning and momentum rates

Typically, each back propagation training session starts with different initial weights and biases, and different divisions of data - training, validation, and test sets. By varying these parameters it was observed that these different conditions can lead to very different solutions for the same problem. Weights that resulted in small error values were identified and used for the architecture. Reed and Marks recommend using small initial weights to avoid immediate saturation of the activation function (Reed & Marks, 1999). Further an investigation on how varying the number of hidden layers and Nodes affected the total mean square error was conducted. As mentioned earlier to avoid *overfitting* a few hidden layers were used. Training was conducted using less than three layers and it was observed that the error grew as the number of layers increased as shown in Figure 5.14. For the data set used there was no significant difference in the error when one or two hidden layers were used as well as when the number of nodes in each layer was increased. However when the nodes are too few the network can result in *underfitting,* when the model is not able to obtain a sufficiently low error value on the training set. On the other hand using too many neurons in the hidden layers can result in *overfitting* and an increase in the time taken to train the network. Nodes that were close to the number of input and output parameters were therefore selected.

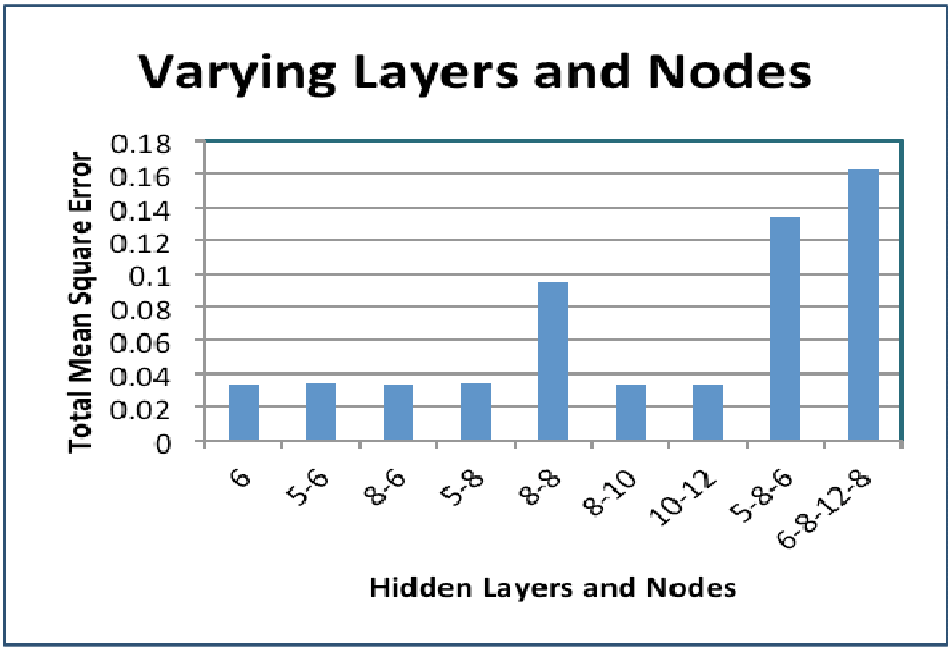


Figure 5.14 Training by varying the hidden layers and nodes

It can be deduced from this result that for the dataset a prediction system can be optimized with a low factor learning rate (e.g. 0.2) and a moderate momentum (e.g. 0.5). Further the minimum

number of hidden layers for a deep neural network, two, was sufficient for the experiment with 5-6 nodes per layer. A screen shot of the ANN Architecture arrived at after the training process using *Neuroph Studio* process is depicted in Figure 5.15.

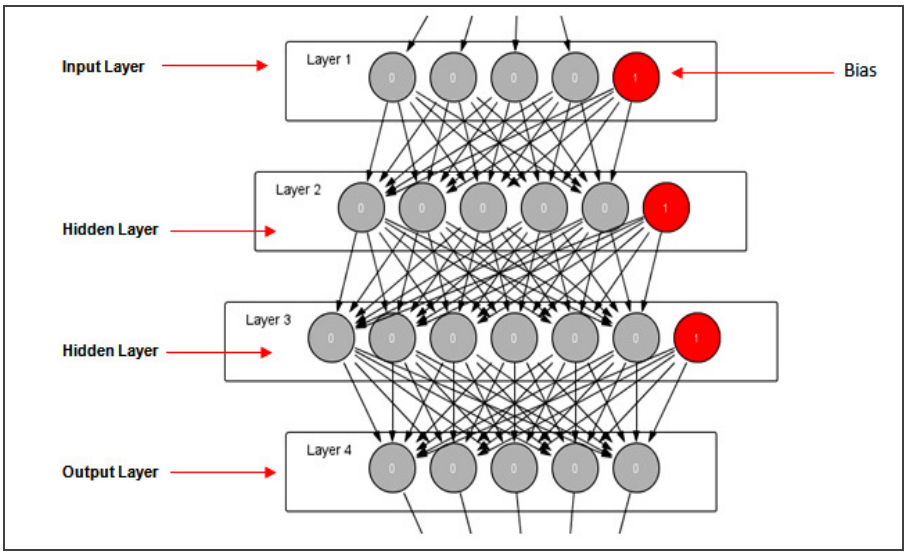


Figure 5.15 ANN Architecture

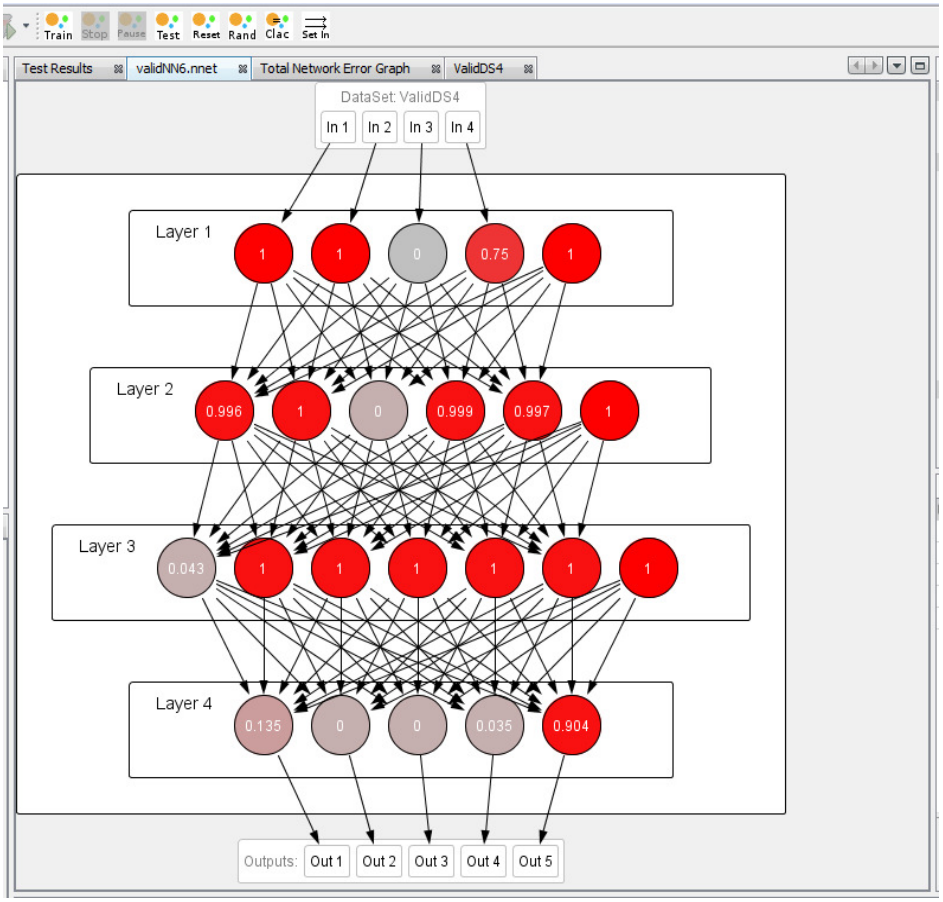
When the neural network architecture was tested it produced comparatively more accurate results than the previous experiment. These are discussed next.

5.5.2. Neural Network Accuracy

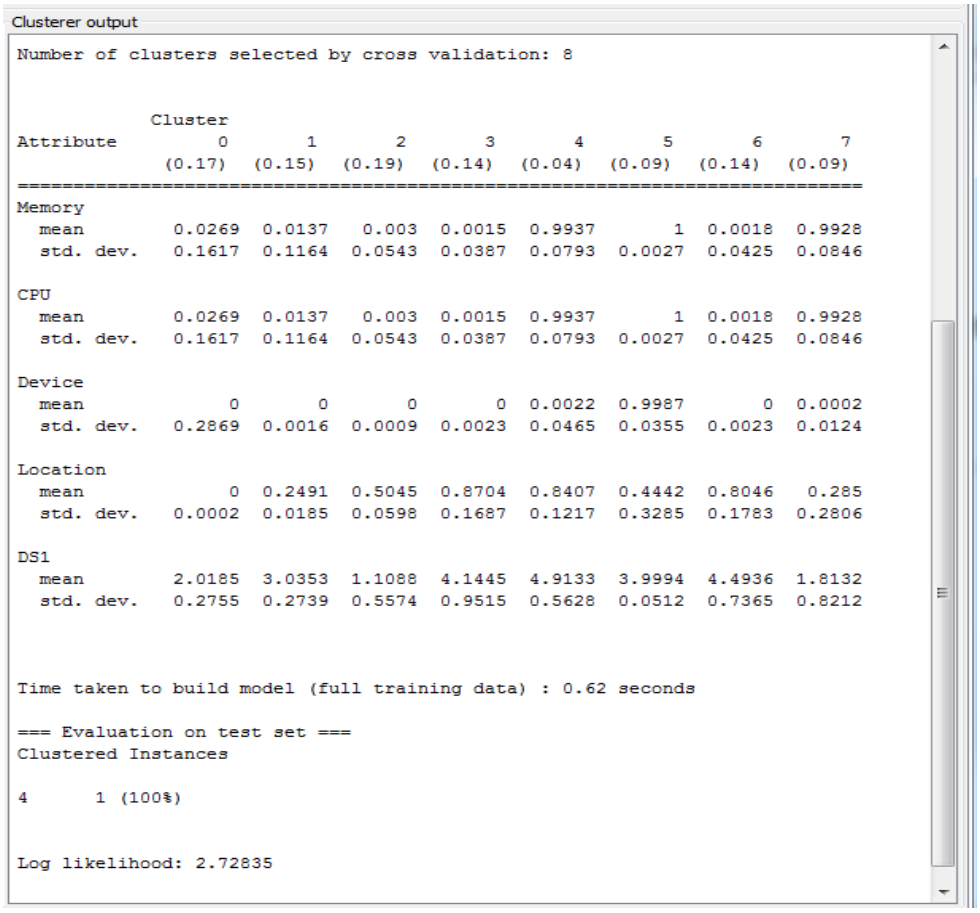
To gauge the accuracy of the algorithm three tests were conducted: (i) a test for data that was well represented in the training set; (ii) a test for data whose approximate values only were in the training set; and (iii) a test for data that was not in the training set nor were its approximate values. A comparative analysis of both the deep learning algorithm and the EM clustering algorithm was performed and the results described next.

When tested with input that was *well represented* in the training set, as expected the A*NN* gave accurate results as shown Figure 5.16 (a). For example, a user system with the input parameters as high memory (1), fast processor (1), a mobile device (0), and location 0.75 (symbolizing geo-coordinates), gave the output as the disease labeled 5. This is because the most significant output is the fifth output from the figure. Several occurrences of the exact input were in the training set. The same data was run through EM clustering algorithm. The results, show that for the same test

data 1 1 0 0.75 (described earlier) with expected output as disease 5, the results were also given accurately as belonging to cluster 4 whose features are described as `0.9937 0.9937 0.002 0.8407 4.9133 (i.e. ≈ input: 1 1 0 0.8, ≈output:5)`. Such a cluster is strongly represented as shown in Figure 5.16 (b).



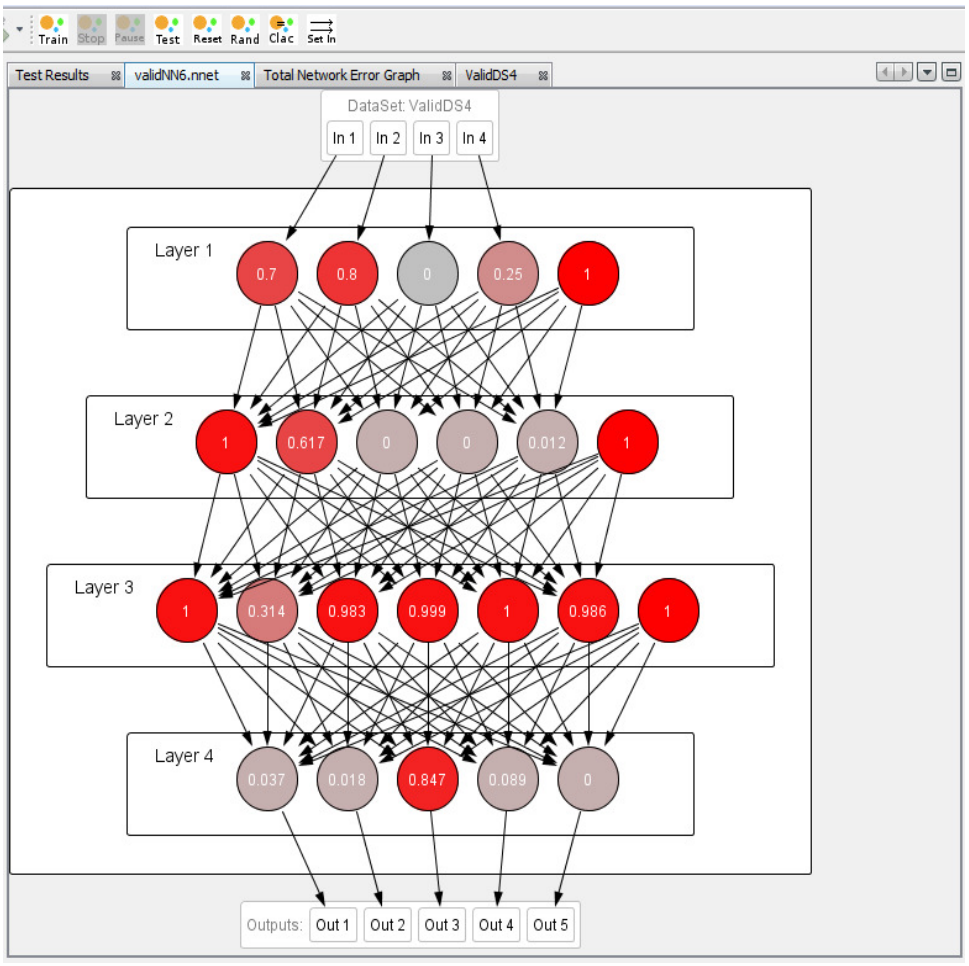
(a) ANN Classification Output



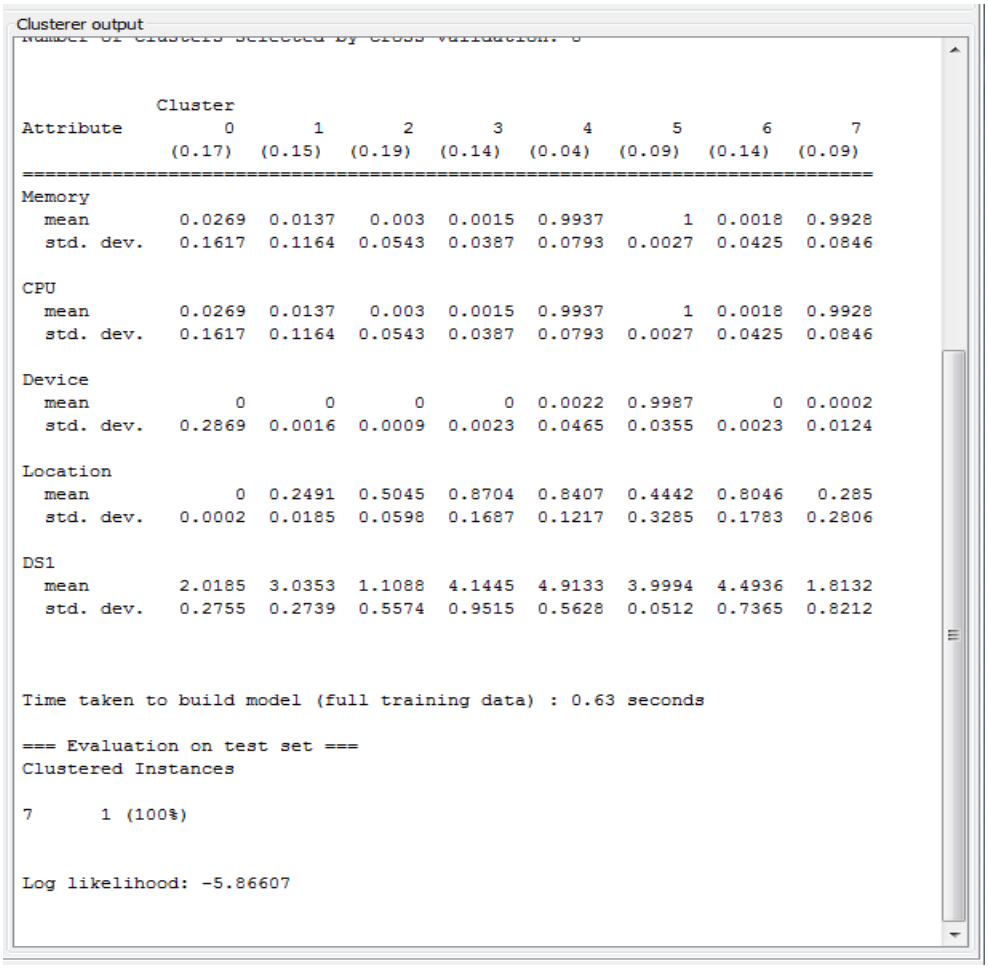
(b) EM Clustering Output

Figure 5.16 Exact Data in Training Set

Further the *ANN* was tested using *approximate values* in the training set. For example, the training set had `1 1 0 0.25` and a test data of `0.7 0.8 0 0.25` was used instead, which was not present in the training set. Based on the training set, the expected output was the disease labeled 3. The results shown in Figure 5.17 (a) demonstrate that the A*NN* once again gave accurate results indicating it was able to generalize. The most significant output is the third output. However when the same test was run on the EM clustering algorithm, the results were given inaccurately as belonging to cluster 7 whose features are described as `0.9928 0.9928 0.0002 0.285 1.8132 (i.e. ≈ input: 1 1 0 0.3, ≈ output: 2)`. The results are inaccurately returned as belonging to the disease labeled 2 in Figure 5.17 (b).



(a) ANN Classification Output



(b) EM Clustering Output

Figure 5.17 Approximate Data in Training Set

A visualizing of the data on WEKA reveals the reason why. The training set has two clusters as shown in Figure 5.18. The smaller cluster could be as a result of emerging behavior, for instance a disease epidemic that results in a change of user needs as explained in the first experiment. EM clustering algorithm identifies all existing clusters accurately, but does not sift through the output to identify what is significant.

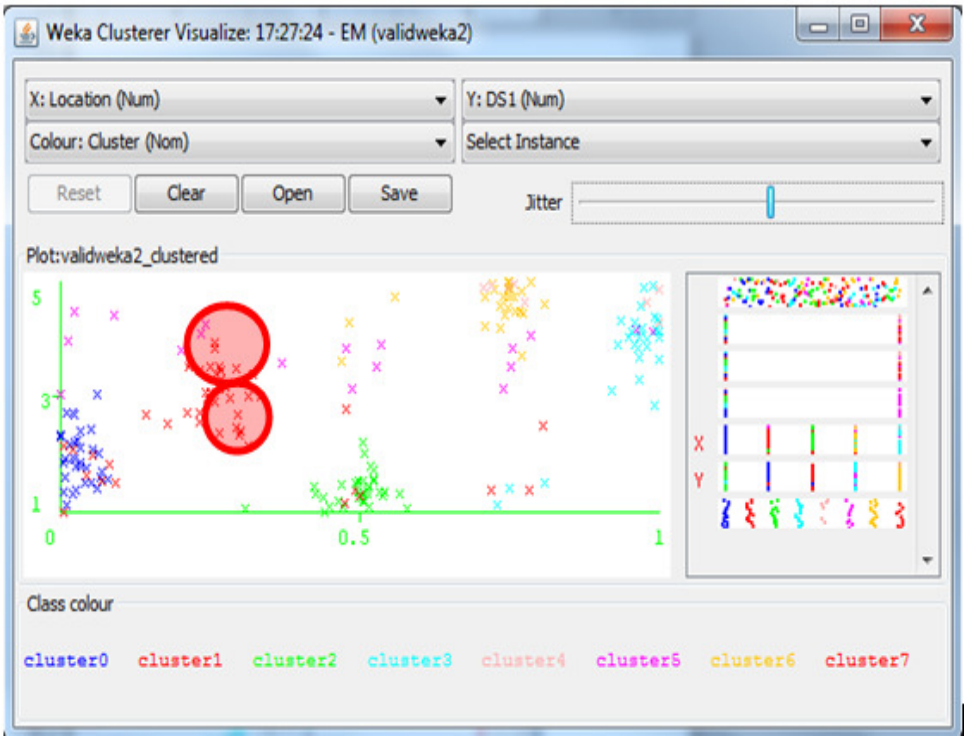
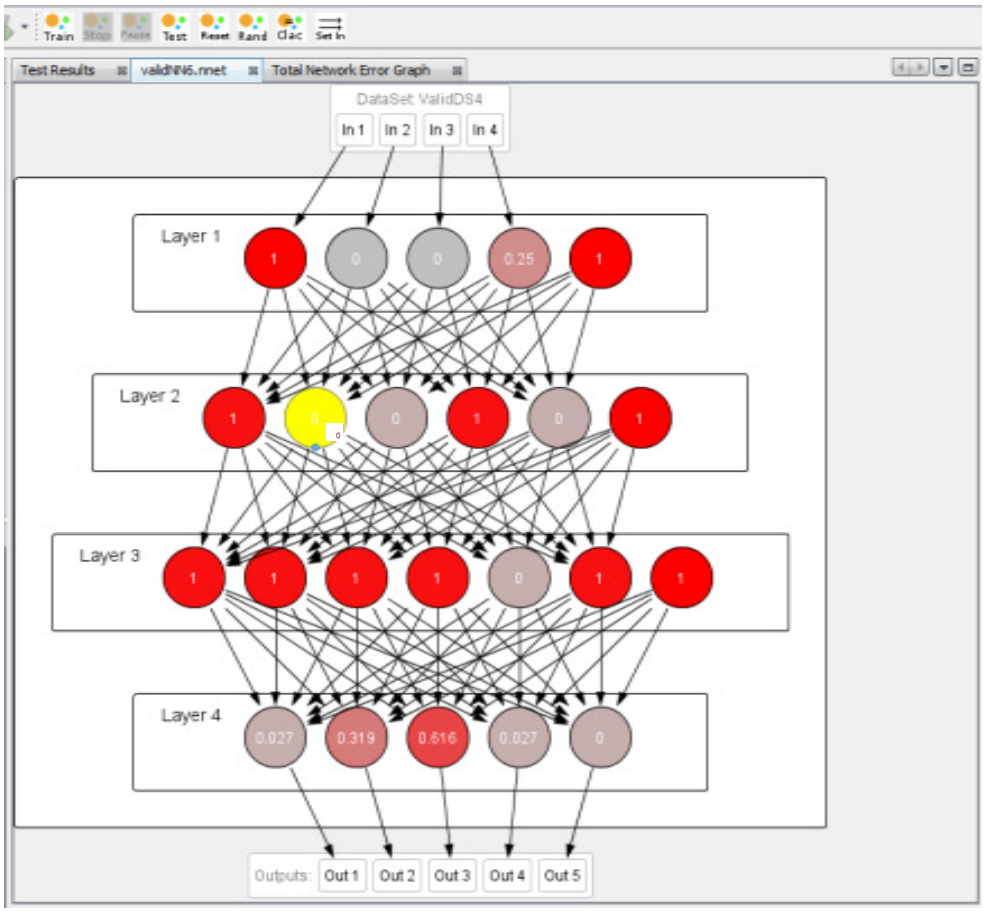
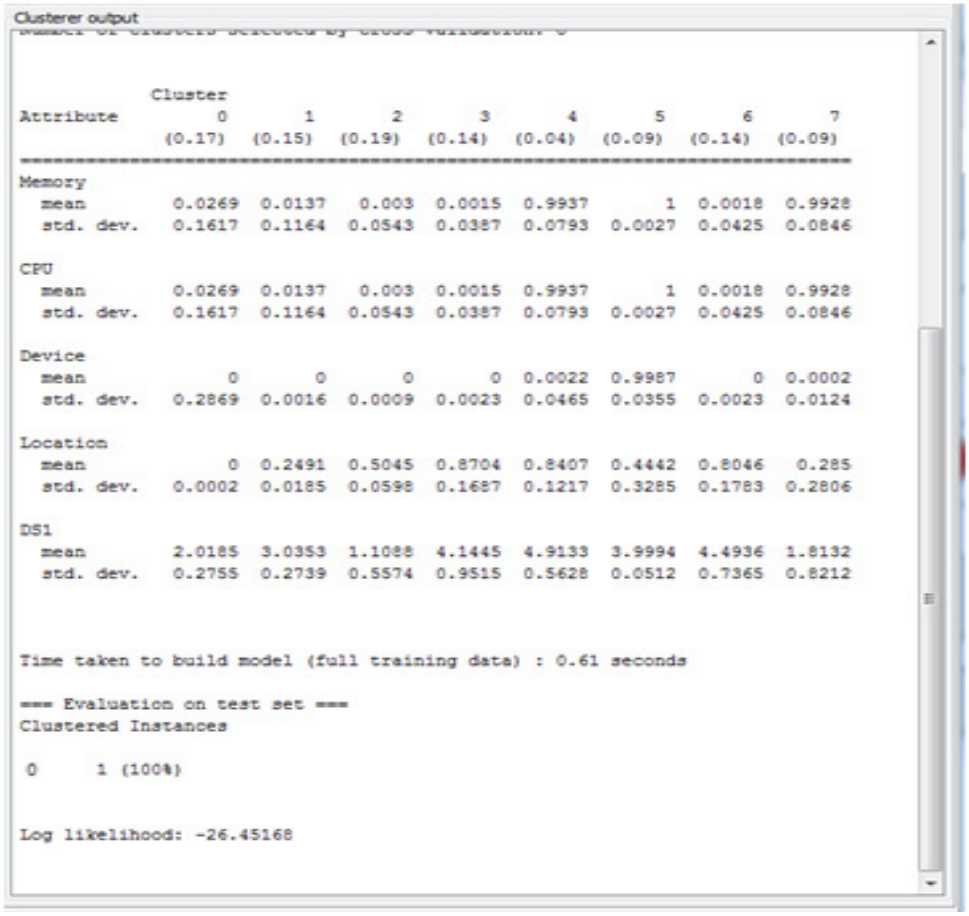


Figure 5.18 Visualization of emerging clusters

Another notable result was obtained when data that was *not represented* in the training set, or its approximate values, was tested. From the results shown in Figure 5.19(a), it was observed that the A*NN* was still able to detect that geo-location was the driving feature and correctly gave the output as 3. However, the EM clustering algorithm inaccurately predicted the output as the disease labeled 2 (Cluster 0), as depicted in Figure 5.19(b).



(a) ANN Classification Output



(b) EM Clustering Output

Figure 5.19 No Data in Training Set

A summary of these results is given in Table 5.1. It is clear that when data is not well represented in the training set the EM clustering algorithm is unable to learn through generalization. Deep learning however has the ability to generalize and can therefore be used to improve on self-validation. The fact that the neural network is able to generalize shows that it is not *overfitting*. One method for improving network generalization is to use a large network, however the larger the network, the more likely it is to over fit the data. If you use a small enough network, it will not have enough power to over fit the data

Table 5.1. Testing the learning algorithms

Test Description	Input Parameters				Output Parameters						Expected Output
	Memory Size	Processor Speed	Device Type	Geo-location	ANN Output (*Disease marked by significant output*)					EM Output (*Disease in selected cluster*)	
					Output 1	*Output 2*	*Output 3*	*Output 4*	*Output 5*		
Exact Data in Training set	1	1	0	0.75	0.135	0	0	0.035	**0.904**	4.9133 (≈5)	5
Approximate Data in Training set	0.7	0.8	0	0.25	0.037	0.018	**0.847**	0.089	0	1.8312 (≈2)	3
No Data in Training set	1	0	0	0.25	0.027	0.319	**0.616**	0.027	0	2.0185 (≈2)	3

5.7 Experiment 5 –Mitigating System Resource Demands

Although the results obtained in the previous experiment were encouraging, it was observed that deep learning exerted significant constraints on the system resources. The system used to run the experiments was a Pentium Dual-Core CPU with a 2.3 GHz processor and 4 GB RAM. Given that it was recommended as a method of validating *dynamic* adaptation, there was need to establish how the chosen algorithm affected performance and subsequently find a way of addressing this constraint on resources. The performance of deep learning was compared to EM clustering in terms of the execution time of each algorithm.

5.6.1. EM Clustering Performance

Clustering is an important means of data mining and of algorithms that separate data of similar nature. While there are different types of clustering algorithms, Jung et al., (Jung et al., 2014) show that the EM algorithm is often used to provide functions more effectively. In their work they compare the speed and accuracy of K-means and EM clustering methods and show that EM clustering had higher speeds. The K-means algorithm was slightly more accurate but it was more time-consuming than EM.

In this research EM Clustering on overage takes 0.62 seconds to learn using the full dataset. This can be seen in the screen shot in Table 5.2 obtained from the tests conducted in the previous experiment.

Table 5.2. EM Clustering Performance

Clusters / Attribute		0 (0.17)	1 (0.15)	2 (0.19)	3 (0.14)	4 (0.04)	5 (0.09)	6 (0.14)	7 (0.09)
Memory	Mean	0.0269	0.0137	0.003	0.0015	0.9937	1	0.0018	0.9928
	Std. dev.	0.1617	0.1164	0.0543	0.0387	0.0793	0.0027	0.0425	0.0846
CPU	Mean	0.0269	0.0137	0.003	0.0015	0.9937	1	0.0018	0.9928
	Std. dev.	0.1617	0.1164	0.0543	0.0387	0.0793	0.0027	0.0425	0.0846
Device	Mean	0	0	0	0	0.0022	0.9987	0	0.0002
	Std. dev.	0.2869	0.0016	0.0009	0.0023	0.0465	0.0355	0.0023	0.0124
Location	Mean	0	0.2491	0.5045	0.8704	0.8407	0.4442	0.8046	0.285
	Std. dev.	0.0002	0.0185	0.0598	0.1687	0.1217	0.3285	0.1783	0.2806
DSI	Mean	2.0185	3.0353	1.1088	4.1445	4.9133	3.9994	4.4936	1.8132
	Std. dev.	0.2755	0.2739	0.9515	0.9515	0.5628	0.0512	0.7365	0.8212
No of clusters selected by cross validation: **8**									
Time taken to build model (full training data): **0.62 seconds**									

5.6.2. Deep learning Performance

Deep learning was observed to takes 50 times more to train the same dataset. The time taken to train a neural network depends on the number of iterations and the number of hidden layers. Two hidden layers were used for the network because the previous experiments carried out gave the most accurate results with two hidden layers. Further the more the number of hidden layers the more the system resources consumed. To test performance the *Netbeans* profiler was used to estimate the neural networks training time. To train the Neural Network 500 iterations were used as shown in Figure 5.20. The neural network was observed to learn after about 150 iterations on average.

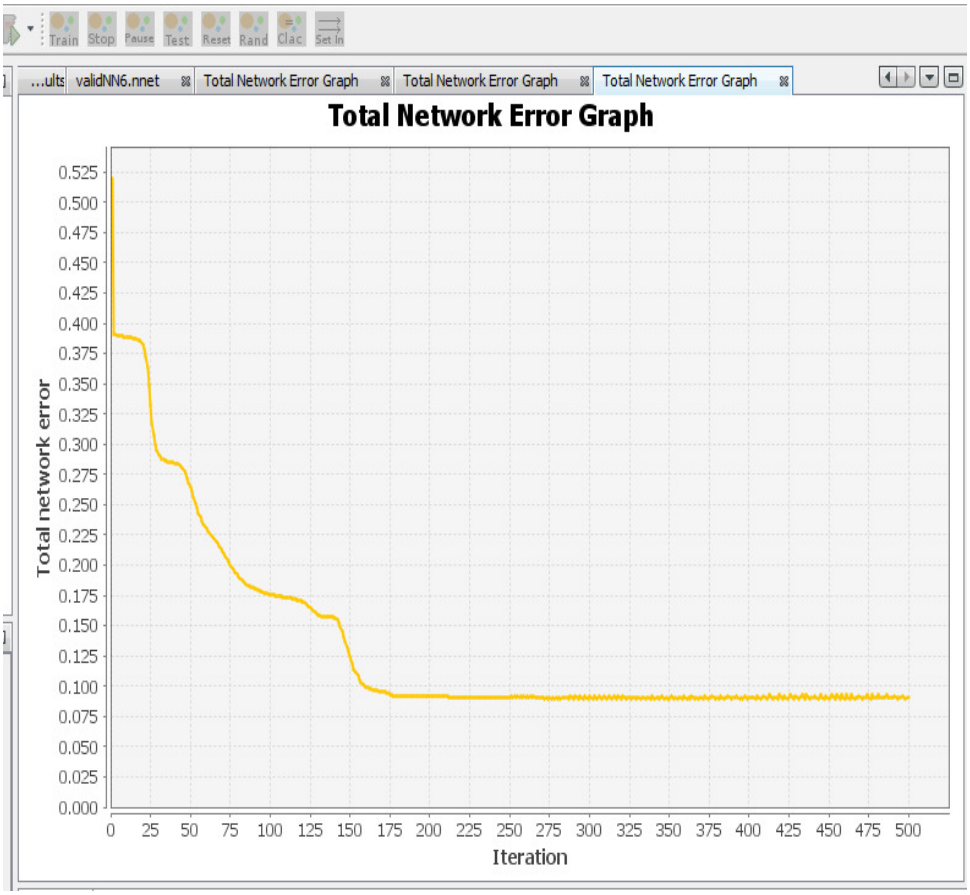


Figure 5.20 Deep learning total network error graph

Netbeans profiler gave the time taken for 500 iterations as 101,865ms on average as shown in Figure 5.21. This gives the average time for 150 iterations as 30559.5ms (30.56s), which is approximately 50 times longer than EM Clustering.

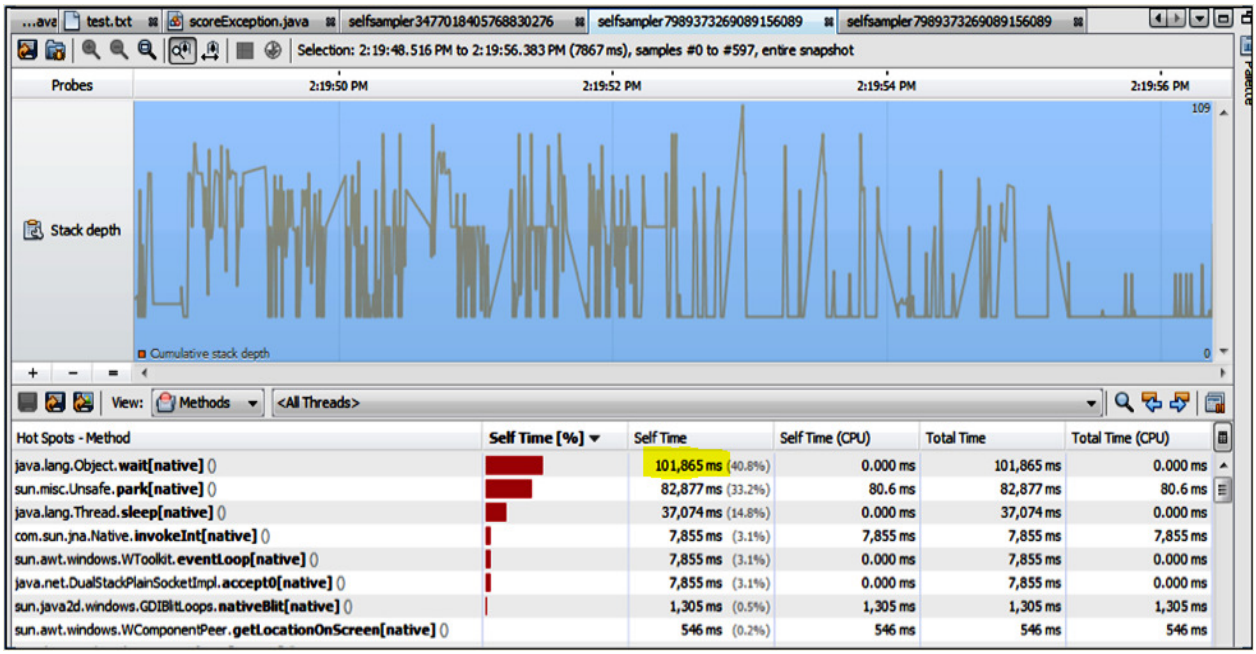


Figure 5.21 Deep learning performance

To address the performance challenge presented by deep learning, deep learning is used off-line rather than at runtime as was the case with EM clustering. Deep learning would therefore compliment the use of EM clustering. This would be achieved by running deep learning on validation logs periodically to validate the results of EM clustering and then using the results to update the validation rules where discrepancies were detected. This way the framework would benefit from the accuracy of deep learning without compromising performance at runtime.

5.8 Summary

Because information access still remains a big challenge in most countries, the provision of a solution that enables users to access only relevant information significantly eases the cost and time burden experienced. In this chapter a health care case study that was used to test the prototype was described. The choice of the case study was motivated by the fact that it presents multiple triggers that can be used to test dynamic adaptation as well as evaluate the self-validating framework proposed. If the decision model falls short in some way, the system should evolve to update the decision rules seamlessly. This scenario enabled the testing of the self-adaptation, validation and evolution process.

To evaluate the framework five experiments were conducted. The objective of the first experiment was to show the frameworks ability to self-validate following an adaptation decision through the use of machine learning algorithms. Because adaptation is often as a result of multiple triggers that sometimes conflict, the second experiment set out to evaluate the frameworks ability to adapt and self-validate in the presence of multiple conflicting triggers. Self-validation could trigger further adaptation if the identified adaptation decision is not acceptable to the user. This iterative behavior could continue indefinitely if the system is not able to accurately learn resulting in an unstable system. The third experiment therefore set out to gauge how well the framework was able to learn based on the stability of the resultant system. Because of limitations experienced when analyzing natural data with clustering algorithms, a fourth experiment was set up to investigate the accuracy of the developed framework. A comparative analysis of EM clustering and deep learning algorithms was conducted to show how the later improved the accuracy of the self-validation process. Finally it was observed that deep learning also placed heavy constraints on system recourses. The objective of the last experiment was to determine the extent to which deep learning constrained the system and identify a solution to the challenge.

Results from the experiment showed that the dynamic adaptation of service oriented systems can be significantly enhanced by incorporating validation in the process. Given that user requirements are dynamic it is difficult to statically determine them and create adaptation rules. Further it is difficult to comprehensively identify all the triggers of adaptation beforehand. Adaptation rules therefore need to be developed at runtime based on changing user requirements. This can only be achieved through a self-validating process where the self-adapting system learns on the fly. Machine learning algorithms can be used to implement the self-learning process.

It is worth noting however that the approach was tested using a single case study. The use of several case studies and a variety of user devices can further validate the framework. Additionally only two machine learning techniques were used. The framework can be improved on by performing further evaluation using other learning algorithms. A longitudinal survey that makes use of more data and a larger geographic location would also have helped in gauging the stability and accuracy of the framework.

The proposed framework can be used in any other scenario where the application used needs to change its behavior depending on external or internal environment triggers. With many

governments providing services to citizen online, such a solution will address the challenge presented by poor infrastructure and limited technology access among low income earners. The same approach can be used for marketing consumer products or in the development of early warning systems. The ability to self-adapt, self-validate, and self-evolve gives systems a longer life than would otherwise be the case, hence saving users the cost and hustle associated with frequently having to search for updates.

Chapter 6

6. Conclusion

This chapter provides concluding remarks to the research described in this thesis by summarizing the findings of the research, making recommendations, and describing the lessons learnt. It starts by reviewing the achievements of this research and comparing them to the research objectives described in section 1.3. From the results obtained while evaluating the framework, the objectives are re-examined with a view of determining if they were fulfilled. Based on these results, recommendations are made on the future direction that the work can take in order to improve on the results. The chapter concludes by discussing limitations of the work and the lessons learnt while conducting the research.

6.1 Objectives Revisited

In section 3.1 the research objectives of this study were laid out. In this section the self-validating runtime adaptation framework is examined to establish how it satisfies these objectives. The objectives were drawn from the challenges identified from a review of current work in the runtime adaptation of service oriented systems. Each research objective is discussed against the results obtained, highlighting the key contributions to the objective.

- The first objective was to develop a framework that can *support runtime validation in self-adaptive service-oriented systems in order to improve adaptation accuracy and relevance.* The review of the current adaptation approaches in dynamic service oriented systems also revealed that current adaptation approaches provide little empirical evidence of their effectiveness. It is possible that after the system adapts the user finds that the final adaptation state is not acceptable. This implies that the adaptation decision was not effective. Further an adaptation decision that is acceptable today will cease to be acceptable in future as the user requirements are not static but keep changing. It is therefore important for an adaptation technique to provide a mechanism for evaluating its effectiveness and subsequently changing to reflect the new user requirements.

The prototype framework showed how this can be achieved though a self-validation process where the user behavior is observed and logged. If for example the user selects different health information from that which is provided by the adaptation rules or a cheaper service than the one recommended by the adaptation rules then their final choice is logged. These logs are used to inform future adaptation rules through a machine learning process, which occurs at runtime. The *Enhanced Maximization* Clustering algorithm is used for machine learning because it performs better than other clustering algorithms as explained in the literature. Subsequently accuracy of the adaptation process is improved through the validation process on the fly.

- The second objective was to develop a runtime framework that can *support multiple, conflicting change triggers.* A review of current adaptation approaches in dynamic service oriented systems revealed that most of the work provides localized solutions to specific industrial problems with specific triggers. For instance a network of surveillance camera could switch from one camera to an alternative camera when they detect failure of a particular camera, (Weyns et al., 2016). Such a system relies only on failure as the trigger for adaptation. Further analysis however showed that applications are often exposed to multiple adaptation triggers that sometimes compete or conflict. For example switching to an alternative camera could be dependent on other factors such as power utilization or the object under surveillance.

  The health care support system case study used in this research presented multiple triggers, some of which conflict. It showed how such a system is influence by external triggers such as the geographic location and internal triggers such as the device type and service quality. While triggers such as the location and the device type may be non-competing, other triggers such as the service reputation and cost can compete hence calling for a compromise. The prototype implemented showed how the compromise can only be arrived at through a validation and negotiation process. The validation process detects the need for a compromise and the negotiation process arrives at this compromise. The validation process then keeps a log to inform future adaptation decisions about this consumer behavior.

- The third objective was to develop a runtime framework that *supported self-learning and has the ability to evaluate the accuracy of the validation process by determining the systems stability.* While the first two objectives made dynamic adaptation more generic and accurate, the addition of a validation component was likely to introduce challenges that needed to be addressed. One of these challenges includes the recurrent need to adapt again if the adaptation decision is not acceptable to the user. This recurrent adaptation results in an unstable system and could be used to measure the accuracy of the validation process. A stable system is an indication that the validation process is accurate because the adaptation process does not immediately trigger another adaptation process.

  The experiments conducted analyzed the logs kept by the system and the results obtained showed how the system stability varied in different locations. Through the validation process the system stability was either maintained or improved on. In some few cases however the system was not able to attain stability, an indication that the validation process was not as accurate. The results further revealed some inadequacies in the selected machine learning algorithm which meant that an alternative algorithm would be required in such cases. As a result the research used a classification machine learning algorithm which proved to be more accurate than the clustering algorithms used for the case study. It is worth mentioning that no single learning algorithm is the best for all scenarios as observed from the literature reviewed. By ensuring that validation is conducted by an independent service, any learning algorithm can be plugged on to the framework to suit the scenario in question as demonstrated by this research.

- The fourth objective was to develop a framework that can *mitigate the negative impacts of self-validation on the systems performance and resources.* The additions of a validation component also pose a second challenge by exerting additional resource requirements on the application as well as degrading the systems performance. This can counter the benefits of validating. This research therefore investigated some of the negative impacts of validation on system resources and how can they be mitigated.

  In this research the resource requirements of the prototype developed was further compounded by the choice of deep learning as the machine learning classification algorithm. The experiments conducted showed how the algorithm took 50 times longer to

perform self validation and used more memory than the clustering algorithm. To address this challenge the research recommends the use of online validation only where the learning algorithm does not exert a strain on the system resources while offline validation can be used with the more resource demanding learning algorithms. Further, the framework recommends the use of online validation in order to reap the benefits of dynamic adaptation while offline validation is conducted periodically to compliment the online approach without degrading the systems performance.

6.2 Future Work

In this section the recommendations for future work are made. It takes into consideration the prototype developed and the results obtained with a view of enhancing the framework for further research. The self-validating dynamic adaptation framework can be enhanced by making use of current and alternative adaptation, validation, and Negotiation strategies. In addition the performance of any implementation can be improved on by making use of current state of the art tools as well as testing the framework on a variety of real life case studies. These recommendations are discussed next:

Identify the appropriate validation strategy based on the selected adaptation strategies and trigger specifications. – This research revealed that different adaptation models exists such as static versus dynamic, predictive versus reactive, and embedded versus pluggable. Further these approaches are influenced by different types of adaptation triggers and the application context. Each of these could in turn call for a validation strategy such as formal methods, model based or machine learning. For instance a static or design time adaptation strategy may call for formal methods or model based validation strategy which was not examined in this research. Further work is required to make recommendations on the validation strategy ideal for specific adaptation strategies. Additionally the adaptation strategy may be dependent of the type of sensors or triggers used. Further work is required to analyze this relationship with a view of making the framework more generic.

Improving the accuracy of the self validating framework – From the results obtained it was observed that different machine learning algorithms have different accuracy measures for the same problem. The literature reviewed also suggests that no single machine learning algorithm is ideal for all cases. Further work should analyze different machine learning algorithms to

determine which algorithms provide more accurate results for specific cases. This can be achieved by implementing the framework on a variety of case studies as well as exposing it to different machine learning algorithms. In cases where competing adaptation triggers exist integrating current service negotiation approaches would also improve on the accuracy of the framework.

Measuring and addressing system stability at runtime – In this research the need for recurrent adaptation was used as an indicator of system stability. The results are used to determine the effectiveness of the validation approach in order to improve on it by selecting alternative machine learning algorithms. This was however done statically but future work in this area could look at how to perform this on the fly. Checking system stability at runtime would also present additional challenges on performance and resource requirements which would need to be addressed.

Improving the performance of the self validating framework – Although this research looked at how the performance of the framework can be improved on by conducting part of the validation process offline, there are other strategies that can be employed to improve on performance. One strategy could lie on the choice of tools used to implement the framework such the development environment, libraries, web servers and the programming language selected. Further work is needed to analyze how the choice of these tools influences the systems performance. Performance can also be improved by testing the prototype developed on different hardware or operating system platforms.

The use of a self validating framework in risk aversion – The frameworks ability to predict adaptation requirements before adaptation takes place means that it can be use to provide early warnings in places where the adaptation decision is risky. This can be achieved by integrating a forecasting component in the framework. At present the framework can only provide information that affects the current adaptation decision. Future work could look at how predictive models can be used by analyzing validation logs and avert future risks. This research did not also investigate different ways of analyzing the validation logs based on the specific case study. For example, how far back should validation logs be mined to inform the current or future adaptation decision? Additionally different validation thresholds exist for different cases. For instance the number of logs that can be used to determine a disease outbreak is different from the number of logs that can be used to determine consumer preference for a product in the market. Such analysis would further inform the use of the framework for risk aversion.

Conducting a longitudinal experiment with the framework to test its ability to provide early warnings – In order to test the framework on disease outbreaks, experiments should be conducted over a long period of time. This is because disease outbreaks can occur several years apart. Due to the limited time available for this research, such an experiment could not be conducted. As a result simulations were used to test the framework where users simulated a sudden interest in certain diseases. User behavior however is a product of many factors such as their social environment, cultural background, literacy levels and economic situation among others. Such behavior is not always predictable and therefore not always easy to simulate.

Adding features to the prototype to enhance the utility – The utility of the framework could be improved by providing a feature to search for diseases presence in different geo-locations. This would be particularly useful for users who would like to visit a location and take precautionary measure to avoid infection and hence curb the spread of disease.

6.3 Reflection

During the course of this research a number of design and implementation choices were made that invariably affected the results and evaluation of the approach described in this thesis. This section reflects on the process and provides a list of the limitations encountered as well as the lessons learnt.

6.3.1 Approach limitations

Although this research provided one way of improving the effectiveness of dynamic adaptation of service oriented systems, the solution presented also countered this effectiveness in some ways. For instance the addition of validation mechanisms degraded the quality of service of the adaptation process by increasing performance time and memory requirements. Further the type of machine learning algorithm selected also determined the accuracy of the validation component. Attempts to optimize the validation component were discussed in the research although they do not completely eliminate the challenge presented. The use of static validation partially mitigated the performance challenge and resource requirement challenge.

Another limitation to this approach comes from the choice of validation strategy used. While the machine learning strategy gives the framework the ability to learn, the behavior of self adaptive systems during design and implementation is also critical for validation and can be

achieved by other validation techniques such as formal methods and model based approaches that were not factored in the proposed framework.

Because the development of a machine learning algorithm was outside the scope of this work, it is also worth noting that the validation component is only as accurate as the machine learning tools used. The use of alternative tools such *Deeplearning4J*, alternative platforms such *R Platform*, and alternative libraries such as *scikit-learn* may have presented more accurate results but a cross analysis was not performed in this work.

Finally the nature of the case study required a number of experiments to be simulated rather than the use of real data. This is because obtaining the data would have called for a longitudinal survey, which could not have been achieved within the time allocated for this thesis, such as the simulation of disease outbreaks. Additionally consumer behavior in the field is not always the same as their behavior in a controlled environment. For example many users wanted to experiment with the application rather than mimic their actual behavior in the field. As a result some of the user behavioral limitations were not addressed in the research.

6.3.2 Lessons Learnt

Any experiment conducted on performance should make comparisons based on different hardware and software Platforms. This research was primarily conducted on a single device and hence was only able to comment on performance across different software platforms such as the machine learning tools used. Further while the *NetBeans Profiler* was used to measure performance, other specialized and more accurate tools such as *JProfiler* or *Visual VM* and methods such as the inbuilt *nanoTime()* could have complimented the approach.

Another notable lesson was the need to engage an epidemiologist when determining the control or validation threshold values for the case study on disease outbreaks. While the research made use of frequencies to determine a potential outbreak, there are more specific methods for determining outbreaks such as measures of *Incidence proportion*, *Incidence rates*, and *Prevalence*. Further there is need to analyze consumer behavior when developing a consumer-centred solution. Cultural, social, personal, and psychological factors influence consumer behavior and should be factored in when designing a solution to predict this behavior.

6.4 Final Remarks

While the past two decades have seen a lot of research conducted in the dynamic adaptation of service oriented systems, very little research has focused on evaluating the effectiveness of these approaches. Recent attempts have proposed verification approaches which focus on software specification or model-based validation approaches. This ultimately means that the adaptation solution presented will become obsolete the moment user requirements change.

This research set out to address this challenge by proposing dynamic adaptation solutions that can self-validate and self-evolve to address changing user requirements. It achieves this by incorporating a learning component using machine learning clustering and classification techniques. The solution proposed is evaluated using a health care support system. Experiments are conducted that reveal the strengths and the weaknesses of the approach and solutions provided to address the weaknesses. The results showed that the proposed self-validating framework can be used to significantly improve the effectiveness of dynamic adaptation of service oriented systems.

From the findings recommendations are made to improve on the work by making use alternative machine learning techniques as well as testing the application on other case studies. Future work in this area should also investigate the effectiveness of the self-validation framework at runtime as well as draw expertise from other disciplines such as epidemiology or social sciences.

Appendix A

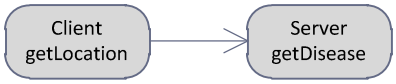
A. System Reconfiguration

This appendix shows how adaptation techniques based on workflow reconfiguration and the selection of alternative services were implemented and tested. Workflow reconfiguration was based on the fact that different workflow pattern have different performance constraints. The selection of alternative services was based on the fact that services from a provider can fail calling for an alternative provider to be selected.

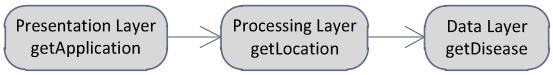
A.1 Workflow Reconfiguration

Workflow reconfiguration called for orchestration of the service oriented system based on work flow patterns chosen to represent different architectural styles as illustrated in Figure A.1.

Work Flow 1 - Client Server Architecture



Work Flow 2 – Message Bus Architecture



Work Flow 3 – Layered Architecture

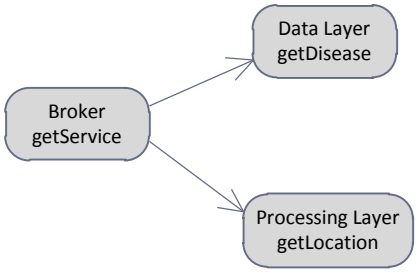
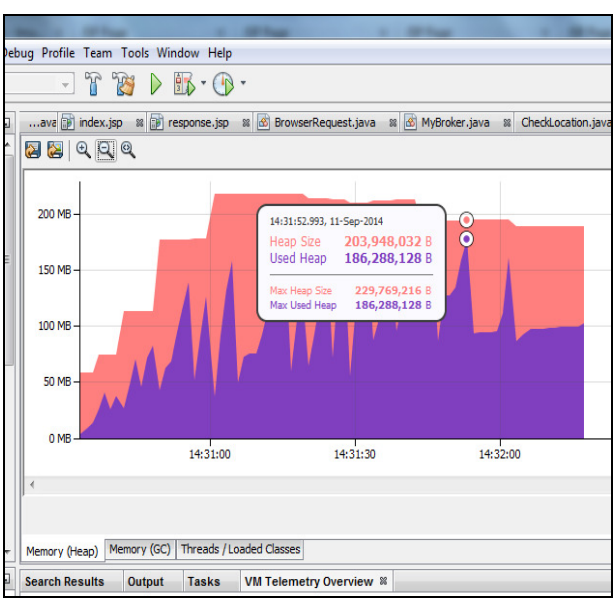


Figure A.1 Sample workflow patterns used

When the SOA system is orchestrated in the different ways depicted above some quality aspects of the system change as illustrated in Table A.1. Figure A.2 (a) shows how we used the

Netbeans profiler to determine the systems performance for different workflow patterns. We kept a log of the adaptation decision for validation purposes, Figure A.2 (b). The symbols WR, NG, AS, NA represent the adaptation techniques used for example WR represents Workflow Re-composition, and AS represents Alternative Service. The default workflow pattern works well in an environment where resources are not constrained. However when the application detects a mobile device adaptation occurs where a workflow pattern that works well in a resource constrained environment is selected. The logs then indicate that WR was the adaptation technique used for example where the data reads 4 3 1 WR. The user can also reject the service suggested based on the attributes such as cost and reputation. When this happens, the adaptation process results in an alternative service being retrieved. The limitation here is that the logs only store the final adaptation decision. For example the log - *(4 3 3 AS)*, the work flow reconfiguration process that occurred before is not logged.




```
@relation 'Validation and AR2'

@attribute LID numeric
@attribute PERFORMANCE numeric
@attribute COST numeric
@attribute ARULE {WR,NG,AS,NA}

@data
4,3,1,WR
4,3,1,WR
4,3,1,WR
4,2,1,NG
4,3,1,WR
4,3,3,AS
4,2,2,NG
4,2,2,NG
```

(a) Performance based on Workflow Patterns *(b) Workflow Re-composition Decision logs*

Figure A.2 Evaluating performance of different workflow patterns

A.2 selecting alternative Service Providers

A service failure trigger was simulated by ensuring the service selected fails. We do this by making the selected service throw an exception at runtime. The monitoring component detects the failure through the use of a service sensor. The adaptation component then picks the alternative service described in the adaptation rules.

To test six virtual machines were used whereby each hosted the service to mimic different service providers. Some of the services were made to fail by throwing runtime exceptions. The adaptation engine begins by detecting if a called service has thrown an exception. If the exception is detected by the monitoring component then the adaptation component responds by providing an alternative service. If the alternative service fails the process is repeated until a successful service is found. Figure A.3 illustrates part of the distributed systems architecture highlighting the different service providers.

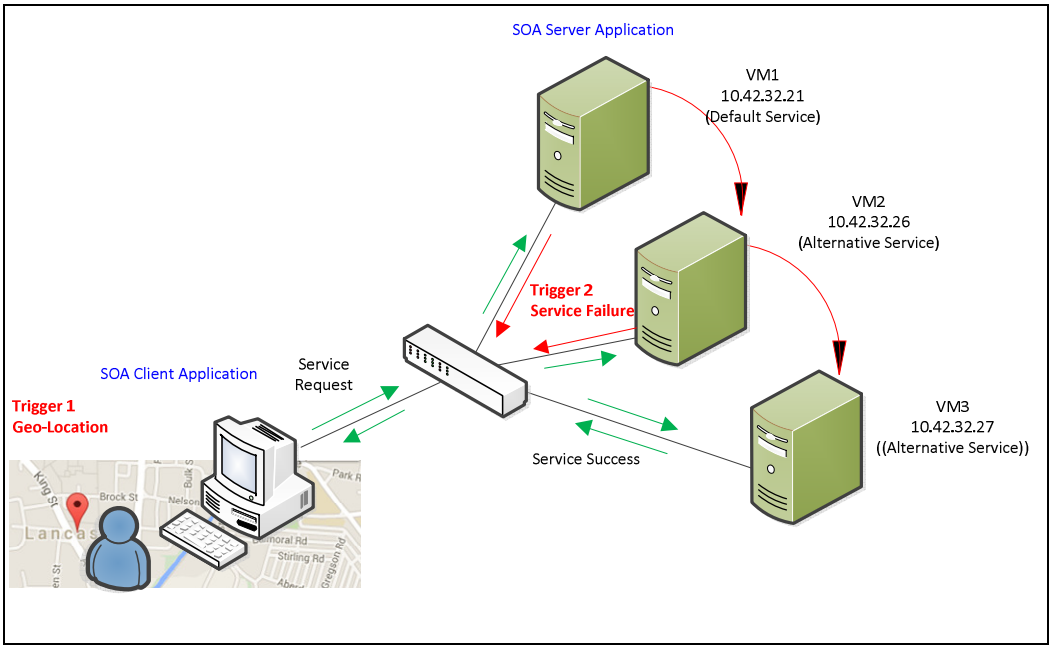


Figure A.3 Multiple triggers on a distributed system

A.3 Binding Provider Interface

If all the service providers available fail then a message is returned to the user that no service providers are available. To implement the *BindingProvider* interface is used which provides access to the protocol binding and associated context objects for request and response message processing. The code snippet in Listing A.1 shows how this was done.

```
import javax.xml.ws.BindingProvider;

public static String getLocation(java.lang.String coordinates) {
        String r = "N";
System.out.println("in location service with coord: " +    coordinates);
com.geohealthf.ws.CheckLocation_Service service = new
com.geohealthf.ws.CheckLocation_Service();
com.geohealthf.ws.CheckLocation port = service.getCheckLocationPort();
BindingProvider bp = (BindingProvider) port;


bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
"http://10.42.32.21:8080/HealthServer/CheckLocation?wsdl");
        r = port.getLocation(coordinates);

        if (r != null) {//(!r.equals("")) {
            System.out.println("VM1");
            return r;
        } else {

bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
"http://10.42.32.26:8080/HealthServer/CheckLocation?wsdl");
            r = port.getLocation(coordinates);
            if (r != null) {
                System.out.println("VM2");
                return r;
            } else {

bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
"http://10.42.32.27:8080/HealthServer/CheckLocation?wsdl");
                r = port.getLocation(coordinates);
                if (r != null) {
                    System.out.println("VM3");
                    return r;
                            }}}}}}
System.out.println("No service providers found!!");
return r;
    }
}
```

Listing A.1 Binding Provider Interface Implementation

Appendix B

B. Using the Neural Network

A multi-layer NN architecture is made up of an input layer, one or more intermediate or hidden layers, and an output layer. Each layer is made up of nodes called neurons with no connections between neurons in the same layer, but the neurons in adjacent layers are all interconnected. In order to calculate the output of a neuron, based on the input several functions exist. In order to calculate the output of a neuron, based on the input, Neuroph Studio uses the sigmoid function. This function can easily identify both possibilities without ambiguity.

B.1 Normalizing the Data

In machine learning a classification problem is one that calls for assigning a predefined class to a given input. There are several algorithms that can be used to solve a classification problem. Multilayer neural networks, using the back propagation algorithm have proven that they can solve such problems with high accuracy. In order to train the neural network using this algorithm the data set has to be normalized for the values range from 0 to 1. We used a feature scaling formula for this.

Figure B.1 shows a sample of the training dataset designed to achieve this in our case study as captured in Neuroph Studio. Because deep learning calls for several input features we combined both our environmental and business triggers described earlier as the input features for the neural network. These features are Memory Capacity, Processor Speed, Device, and Geo-location, labeled Input 1-5 in the screen shot.

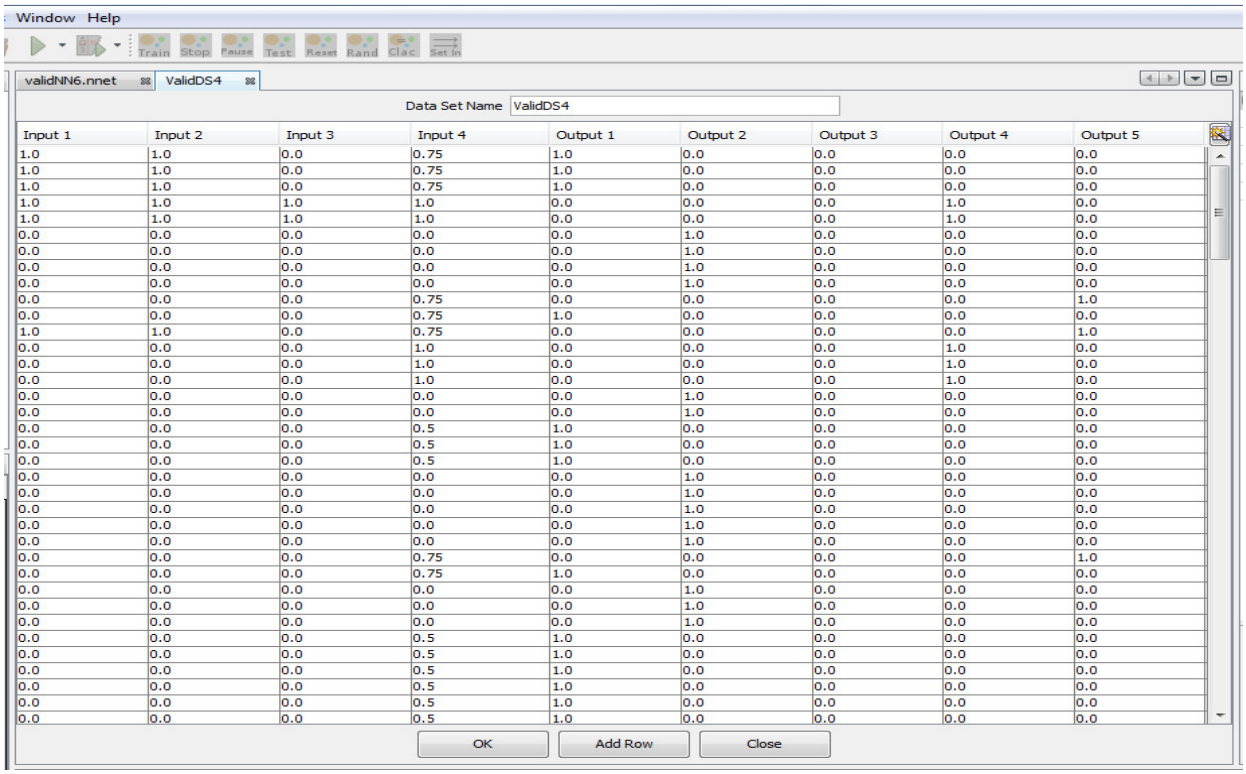


Figure B. 1 Normalized Input Variables

Appendix C

C. Automating the validation component

C.1 EM Clustering Implementation

The machine learning algorithms used for this study was EM Clustering which was implemented using Java Libraries imported from WEKA, open sourced under the General Public License (GPL). It was designed as a web services to support interoperable machine-to-machine interaction over a network using SOAP messages. A snippet of the code used to implement this algorithm is illustrated in Listing C.1

```
//-------------------- Run EM Clustering Library---

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.ejb.Stateless;
import weka.core.Instances;

import weka.clusterers.DensityBasedClusterer;
import weka.clusterers.EM;
import weka.clusterers.ClusterEvaluation;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

@WebService(serviceName = "valid")
@Stateless()
public class valid {

@WebMethod(operationName = "execute")
public String execute(@WebParam(name = "input")
final String input, @WebParam(name = "slid") String slid)throws Exception{

ClusterEvaluation eval;
Instances data;
String[] options;
DensityBasedClusterer cl;
data = new Instances(new BufferedReader(new
FileReader(input)));
StringBuffer result;
result = new StringBuffer();
result.append("Weka - Demo\n==========\n\n" + "\n\n--> normal\n");
options = new String[2];
options[0] = "-t";
options[1] = input;

result.append(ClusterEvaluation.evaluateClusterer(new
EM(), options));
```

Listing C.1 Cluster Extraction

In order to identify significant clusters where two or more clusters existed the cluster density was obtained using the code snippet indicated in Listing C.2 (a). The number of clusters formed was also extracted using the code snippet provided in Listing C.2 (b).

```
// --------------------Evaluating Cluster density-------
cl = new EM();
eval = new ClusterEvaluation();
eval.setClusterer(cl);
eval.crossValidateModel(cl, data, 10, data.getRandomNumberGenerator(1));
```

(a)

```
//-----------------Getting Number of Clusters-------
try {   result.append("\n--> density (CV)" + "\n\n# of clusters: "
                    + eval.getNumClusters() + "\n\n\n");
        } catch (Exception e) {
            e.printStackTrace(); }
```

(b)

Listing C.2 Cluster descriptions

The role of the implemented web service was to receive the users geographic location from the client web service, read the adaptation rules from an XML file, mine past user accepted adaptation decisions and extract clusters from them, validate the adaptation rules read using the clusters and then execute the adaptation decision by calling the relevant service. Listing C.3 shows a snippet of the code used to read the Adaptation Rules.

```
//--------------------- Read Adaptation Rules.xml---
DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
Document doc = docBuilder.parse(new File("ATRules.xml"));

doc.getDocumentElement().normalize();
doc.getDocumentElement().getNodeName());

NodeList listOfrules = doc.getElementsByTagName("DRule");
int totalrules = listOfrules.getLength();
            //Read Rules
            for (int s = 0; s < listOfrules.getLength(); s++) {
                Node firstRuleNode = listOfrules.item(s);
                if (firstRuleNode.getNodeType() == Node.ELEMENT_NODE) {
                 Element firstRuleElement = (Element) firstRuleNode;
                    //-------
                 NodeList StypeList = firstRuleElement.getElementsByTagName("lid");
                 Element StypeElement = (Element) StypeList.item(0);
                 NodeList textSTList = StypeElement.getChildNodes();
                           + ((Node) textSTList.item(0)).getNodeValue().trim());
                 lidr[s] = ((Node) textSTList.item(0)).getNodeValue().trim();
                 NodeList AS1TechList =
firstRuleElement.getElementsByTagName("did");
                 Element AS1TechElement = (Element) AS1TechList.item(0);
                 NodeList textAS1TList = AS1TechElement.getChildNodes();
                           + ((Node) textAS1TList.item(0)).getNodeValue().trim());
                 didr[s] = ((Node) textAS1TList.item(0)).getNodeValue().trim();
```

Listing C.3 Reading Adaptation Rules

C.2 Visualizing decision making and the learning process

We then extracted 46 results from 46 consecutive runs of the engine and analyzed the data using Weka 3.6.11, a java data mining software. Weka is an open source tool that makes use of machine learning algorithms such as classification, regression, clustering, association rules, and visualization for knowledge analysis. To illustrate how our validation component learns a decision trees are used to visualize the analysis process as shown in figure C.1.

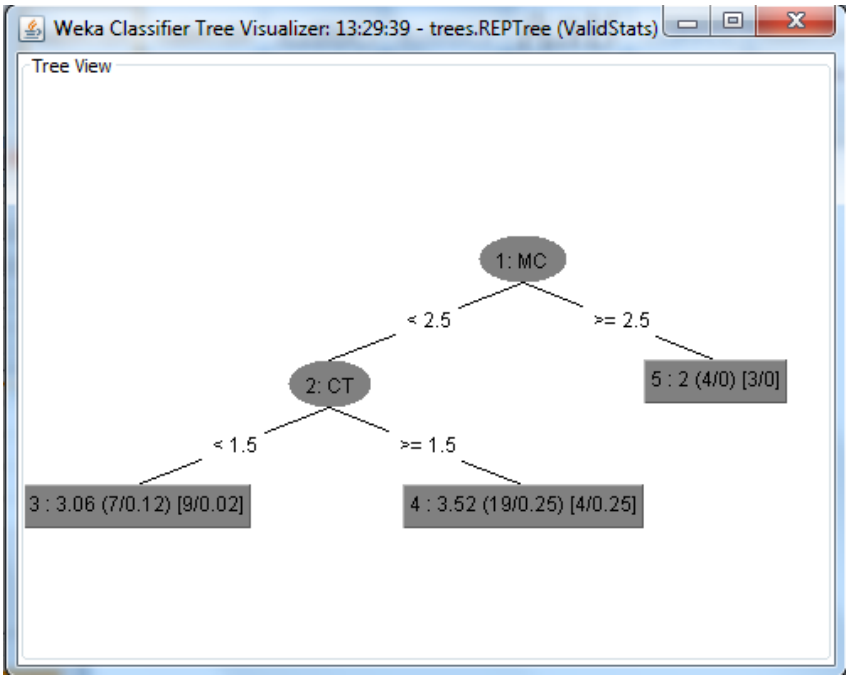
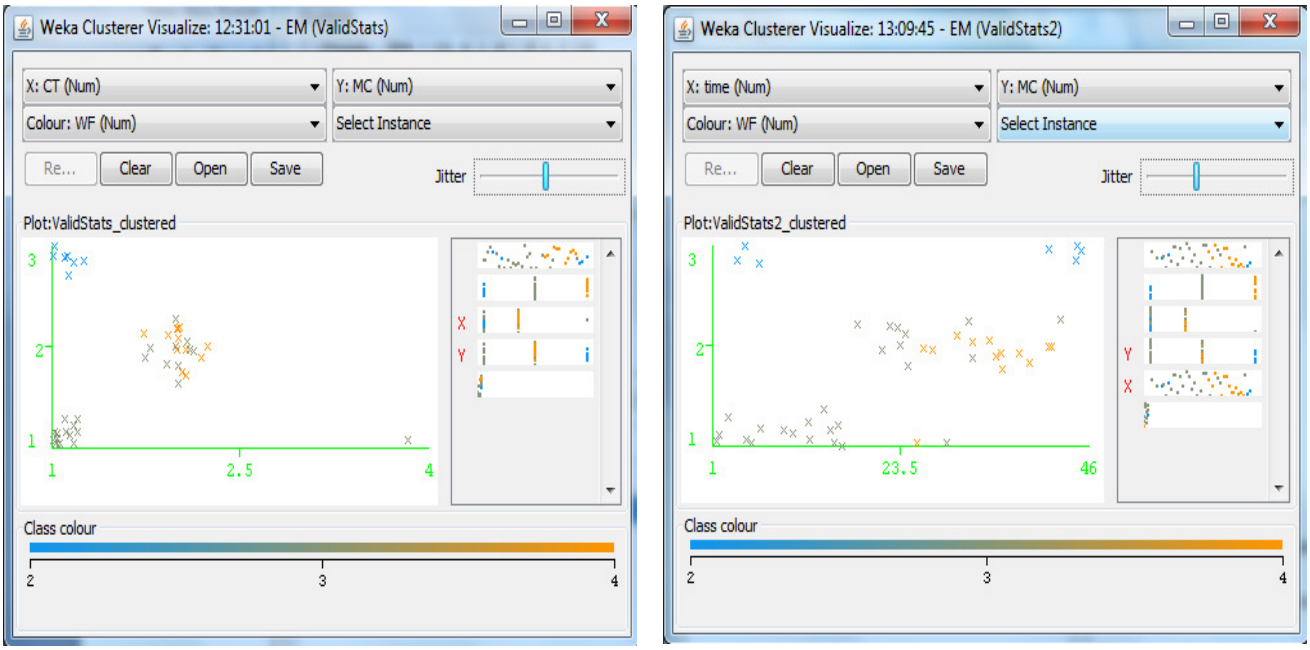


Figure C. 1 visualization of decision making process using Decision Trees

The figure illustrates the decision tree that Weka generated from our simulated data through classification. This decision is based on the validation statistics of the accepted adaptation decision and not necessarily the proposed adaptation rules. The cluster diagrams in Figure A.5 can be used to visualize this. From the results we can be able to see how learning takes place with the decision process changing. For instance you can tell that when the memory capacity (MC) trigger is high (3) and CPU time (CT) trigger is low (1) then the accepted decision was work flow 2 (blue). This was generally acceptable with no deviations. However when the memory capacity trigger and CPU time trigger was moderate (2) the decision was sometimes work flow 3 (grey) and sometimes work flow 4 (orange), Figure C.2 (a). The decision changes with time with the popularity of WF3 disappearing as WF4 becomes the accepted decision as shown in Figure C.2 (b).



(a) (b)

Figure C. 2 visualization of the learning process using clustering

Appendix D

D. SMART Aggregate values for different adaptation Techniques

The aim of the decision phase of the proposed framework is to identifying a suitable adaptation technique. This is done by the adaptation manager, which makes use of predefined, rules (based on experiments) as well as historical adaptation behavior. Re-composition is the act of selecting the appropriate adaptation technique that will be responsible for performing re-composition. As mentioned earlier this process is often compounded by the presence of multiple change triggers. I make use of simple multi-attribute rating technique (SMART), a tool that is widely used for decision-making to identify a suitable adaptation technique. It chooses between several alternatives of adaptation techniques by referring to various adaptation triggers they address. These methods use cardinal weight input information. For example, values can be assigned to some of the adaptation techniques surveyed. This is shown in Table D.1. Suppose a system user has a strong preference to system quality and would also prefer a system that can be customized for his business. The results shown in the table are analyzed for weight desired by the arbitrary system user.

A closer examination of how the techniques perform against quality reveals how the tool arrives at a decision as depicted in Figure D.1. The choice of adaptation technique(s) depends on the nature of the change. The adaptation manager should be in a position to make a trade off on the choice of adaptation technique based on this. It is this decision analysis component that is the key to our integrative approach towards generic runtime adaptation.

Table D. 1 SMART Aggregate Values

	Weight	Zeng et al [8]	Swaminathan [9]	Cubo et al [10]	Huang et al [11]	Dustdar et al [12]	Autili et al [13]	Cardellini et al [14]	Hirschfeld et al[15]	Lorenzoli et al [16]	Maller et al [17]	He et al [18]	Mateescu et al [19]	Zeng et al [20]	Robinson et al [21]	Tanaka et al [22]	Valetto et al [23]	Siljee et al [24]	Orriens et al [25]	Wang et al [27]	Sliwa et al [34]	Zirpins et al [33]	Aoumeur et al [35]	Lins et al [36]	Ivanovic et al [43]
Runtime Env.	0	0	0	0	0	0	50	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	100
Service failure	50	0	50	0	0	50	50	50	0	50	0	50	0	50	100	0	50	100	0	0	0	0	0	0	100
Business Env.	10	50	50	100	50	50	50	50	100	0	50	0	50	100	50	100	50	100	100	100	100	100	100	100	100
Predictive	0	0	0	0	0	0	0	0	0	0	0	0	0	100	50	100	50	50	50	50	50	50	50	50	100
Static	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	100	0
Dynamic	20	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Enbedded	0	100	0	100	100	100	100	0	100	0	0	100	0	100	0	100	100	100	100	100	100	100	100	100	100
Pluggable	20	0	0	0	0	0	0	1	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
Aggregate benefits		25	50	30	25	50	50	50.2	30	45.2	25.2	45	25.2	55	75.2	30	50	80	30	30	30	30	30	30	80

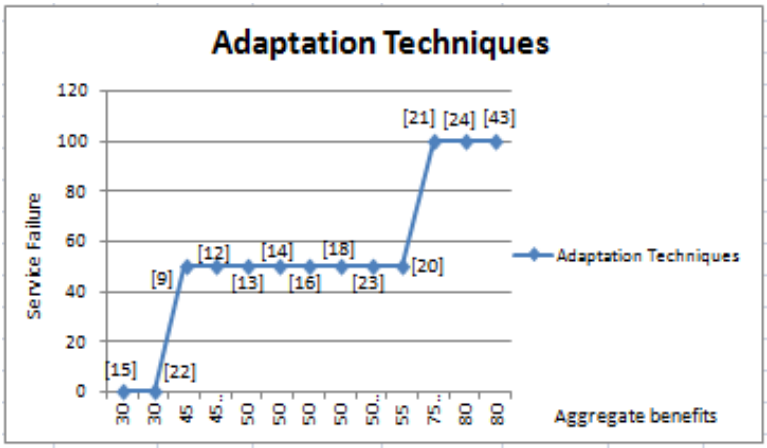


Figure D.3 SMART Aggregate Values for a given Attribute

Once the adaptation manager makes a decision using SMART, a log is added showing the triggers received and the decision made. If the decision made is not sufficient (perhaps due to new user requirements) then this is detected because the system immediately triggers another request for adaptation. Because the log entry only shows the initial triggers and the final decision, a mismatch will be detected between the log entry and the SMART decision. If this occurrence becomes the norm (reaches a stipulated value) then the SMART rules must change and this is done by the validation component. This is how the validation component influences the reconfiguration decision.

The proposed integrated framework will address the limitations identified in existing adaptation techniques in order to provide better runtime adaptation support for service-oriented systems. This framework does not replace existing adaptation techniques, but rather finds a way of adapting them to different system requirements. As a result any new adaptation techniques can simply be plugged into the system for adaptation.

References

Alfares, H. K., & Duffua, S.O. 2008. Assigning Cardinal Weights in Multi-Criteria Decision Making Based on Ordinal Ranking. Journal of Multi-Criteria Decision Analysis.

Alti, A., & Lakehal, A., & Laborie, S., and Roose, P. (2016). Autonomic Semantic-Based Context-Aware Platform for Mobile Applications in Pervasive Environments. Future Internet. 8. 48. 10.3390/fi8040048.

Alpaydin E. (2004). Introduction to Machine Learning. The MIT Press.

Andre, F., Daubert, E., and Gauvrit, G. (2010). Towards A Generic Context-Aware Framework For Self-Adaptation Of Service oriented Architectures. Intl. Conf. on Internet and Web Applications and Services, ICIW, vol. 0, 309–314.

Arcaini, P., Riccobene, E., Scandurra, P. (2015). Modeling And Analyzing MAPE-K Feedback Loops For Self-Adaptation. In: Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS '15, IEEE Press, USA.

Armando A. et al. (2012) The AVANTSSAR Platform for the Automated Validation of Trust and Security of Service-Oriented Architectures. In: Flanagan C., König B. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2012. Lecture Notes in Computer Science, vol 7214. Springer, Berlin, Heidelberg

Autili, M., Berardinelli, L., Cortellessa, V., Marco, A., Ruscio, D., Inverardi, P., and Tivoli, M. (2007). A Development Process For Self-Adapting Service Oriented Applications. ICSOC Lecture Notes in Computer Science, Volume 4749, Springer.

Ayodele, T. O. (2010). New Advances in Machine Learning., Published: February 1, 2010 under CC BY-NC-SA 3.0 license, ISBN 978-953-307-034-6.

Baresi, L., Heckel, R., Thöne, S., Varró, D. (2003). Modeling And Validation Of Service-Oriented Architectures: Application Vs. Style. In: Proc. ESEC/FSE and ACM SIGSOFT, 68–77, ACM Press.

Bartolini, N. Bongiovanni, G. C. and Silvestri, S. (2009). Self-* Through Self-Learning: Overload Control For Distributed Web Systems. Computer Networks, vol. 53, no. 5.

Bengio, Y. (2009). Learning Deep Architectures for AI. Foundations and Trends in Machine Learning, vol. 2, no. 1, 1-127.

Bouguettaya, A., Benatallah, B., Blake, M. B., Casati, F., Chen, L., Dong, H., Dustdar, S., Erradi, A., Georgakopoulos, D., Huhns, M. N., Liu, X., Leymann, F., Mistry, S., Medjahed, B., Malik, Z., Neiat, A. G., Nepal, S., Ouzzani, M., Papazoglou, M. P., Singh, M.P., Sheng, Q. Z., Wang, H., Wang, Y., and Yu, Q. (2017). A service computing manifesto: the next 10 years. Commun. ACM, 60, pp. 64-72,

CAK. (2015). Kenya's Mobile Penetration: Quaterly Report, http://www.ca.go.ke/index.php/what-we-do/94-news/366-kenya-s-mobile-penetration-hits-88-per-cent, Accessed 22 January, 2018.

Calandra, R., Raiko, T., Deisenroth, M. P., Pouzols F. M. (*2012*) Learning Deep Belief Networks From Non-Stationary Streams. In Artificial Neural Networks and Machine Learning–ICANN. Springer, Berlin Heidelberg, 379–386.

Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., and Tamburrelli. G. (2011). Dynamic Qos Management And Optimization In Service-Based Systems. IEEE Transactions on SoftwareEngineering.

Canal, C., Murillo, J. M., and Poizat, P. (2008). Software Adaptation. J. UCS 14(13), pp 2107-2109

Canfora G., and Penta, M., Di. (2006). SOA: Testing And Self-Cheking. In Proceedings of the International Workshop on Web Services Modeling and Testing 3–12, Palermo, Italy.

Cardellini, V., Casalicchio, E., Grassi, V., Presti F. L., and Mirandola, R. (2009). Towards Self Adaptation For Dependable Service Oriented Systems. ICSOC Lecture Notes in Computer Science vol. 5835, Springer, 24-48.

Cardozo, N. Christophe, L. De Roover, C. and De Meuter, W. (2014). Run-Time Validation Of Behavioral Adaptations. in International Workshop on Context-Oriented Programming , USA.

Cheng, B. H. C., De Lemos, R. et al. (2009). Software Engineering for Self-Adaptive Systems: A Research Road Map. Springer-Verlag.

Cubo, J., Canal, C., and Pimentel, E. (2008) Support Context Awareness In Adaptive Service Composition, Proceedings of the Workshops of the Conference on Software Engineering and Data Bases, Volume 2, No. 5.

Cugola, G., Ghezzi, C., Sales Pinto, L., and Tamburrelli, G. (2012). Adaptive Service-Oriented Mobile Applications: A Declarative Approach?. Proceedings of the 10th international conference on Service-Oriented Computing, 607-614.

Cybenko, G. (1989). Approximation by superpositions of sigmoidal functions. Math. Control, Signals, Syst., vol. 2, no. 4, 303-314.

Deitel H. M., and Deitel, P. J. (2011). Java how to Program (9th ed.). Prentice Hall PTR, Upper Saddle River, NJ, USA.

Di Nitto, E., Ghezzi, C., Metzger, A., Papazoglou, M., Pohl, K. (2008). A journey to highly dynamic, self-adaptive service-based applications. Journal of Automated Software Engineering Volume 15 Issue 3-4

Duncan, A., and Hölzle, U. (1999). Load-Time adaptation: Efficient And Non-Intrusive Language Extension For Virtual Machines. Technical Report TRCS99-09.

Dustdar, S., Goeschka, K., Truong, H., Zdun, U. (2009). Self-Adaptation Techniques For Complex Service-Oriented Systems. Fifth International Conference on Next Generation Web Services Practices, IEEE.

Erl, T. (2008). SOA Principles Of Service Design. Prentice Hall.

Eze, T., Anthony, R. J., Walshaw, C., and Soper, A. (2011). The Challenge Of Validation For Autonomic And Self-Managing Systems. In Proc. 7th Intl. Conf. AAS, 128-133.

Fiadeiro, J., Lopes, A., & Abreu, J. (2012). A formal model for service-oriented interactions. Science of Computer Programming, 77(5), 577-608.

Filieri, A. et al. Software Engineering Meets Control Theory. Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS, 2015), [Acceptance rate 16/55, 29%].

Fleurey, F., Dehlen, V., Bencomo, N., Morin, B., and Jézéquel. J.-M. (2008). Modeling and Validating Dynamic Adaptation. In 3rd International Workshop on Models@Runtime (MODELS'08), France.

Fredericks, E. M., Ramirez, A. J., and Cheng, B. H. C. (2013). Towards Run-Time Testing of Dynamic Adaptive Systems. In Software Engineering for Adaptive and Self-Managing Systems (SEAMS), ICSE, 169–174.

Fu, X. Shi, W. Akkerman, A. Karamcheti, V. (2001). CANS: Composable, adaptive network services infrastructure. In Proceedings of the USENIX Symposium on Internet Technologies and Systems. Sixth International Conference on Adaptive and Self-Adaptive Systems and Applications, Venice.

Garlan, D., Schmerl, B., and Chang, J., (2001). Using Gauges for Architecture-Based Monitoring and Adaptation. Proc. Working Conference on Complex and Dynamic System Architecture. Brisbane, Australia.

Gediga, G., Hamborg, K., Düntsch, I. (2001). Evaluation of software systems. Encyclopedia of Computer Science and Technology 45, pp. 166-192.

Gilani, W., Scheler, F. Lohmann, D., Spinczyk, O., and Preikschat, W. S. (2007). Unification of Static and Dynamic AOP for Evolution in Embedded Software Systems. Software Composition pp 216-234

GoK, (2015), Kenya Demographic and Health Survey, Kenya Open Data, Retrieved from https://opendata.go.ke, updated Apr 14, 2015

Golm, M., and Kleinöder, J. (1997). MetaJava – A Platform For Adaptable Operating-Systems Mechanisms. ECOOP Workshop on Object-Orientation and Operating Systems, Jyväskylä , Finland.

Gomaa H., and Hashimoto, K., (2014). Model-Based Run-Time Software Adaptation For Distributed Hierarchical Service Coordination. Proc. Huang, A., and Steenkiste, P. (2004). Building self configuring services using service specific knowledge. Proceedings of the IEEE International Symposium on High Performance Distributed Computing, IEEE Computer Society: Washington DC, USA.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H., (2009) The WEKA data mining software: an update. SIGKDD Explor. Newsl. 11, 1

He, Q., Yan, J., Jin, H., and Yang, Y. (2008). Adaptation Of Web Service Composition Based On Workflow Patterns, Proceedings of the 6th International Conference on Service-Oriented Computing, Sydney, Australia.

Hielscher, J. Kazhamiakin, R. Metzger A. and Pistore, M. (2008). A Framework For Proactive Self-Adaptation Of Service-Based Applications Based On Online Testing. Proc. First European Conf. Towards a Service-Based Internet.

Hirschfeld, R., and Kawamura, K. (2004). Dynamic Service Adaptation, Proceedings of Distributed Computing Systems Workshops, 290 – 297.

Hoffert, J., Mack, D., and Schmidt, D. C. (2009). Using Machine Learning To Maintain Pub/Sub System Qos In Dynamic Environments. Proceedings of the 8th International Workshop on Adaptive and Reflective Middleware.

Hornik, K., Stinchcombe, M., and White, H. (1989) Multilayer feedforward networks are universal approximators. Neural Networks, 359-368.

Huang, A., and Steenkiste, P. (2004) Building self-configuring services using service specific knowledge. Proceedings of the IEEE International Symposium on High Performance Distributed Computing, IEEE Computer Society: Washington DC, USA, pp.45-54

Huber, N., Brosig, F., & Kounev, S. (2011). Model-based self-adaptive resource allocation in virtualized environments. In Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (pp. 90-99). ACM

Hussein, M., and Gomaa, H. (2011). An Architecture-Based Dynamic Adaptation Model and Framework for Adaptive Software Systems. 9th IEEE/ACS International Conference.

IBM. (2006). An architectural blueprint for autonomic computing. Tech. rep., IBM

Ivanovic, D., Carro, M., and Hermenegildo, M. (2010). Towards Data-Aware QoS-driven Adaptation for Service Orchestrations. IEEE International Conference on Web Services, 107-114.

Janert P. K, (2013) Feedback Control for Computer Systems: Introducing Control Theory to Enterprise Programmers. O'Reilly Media

Jung, Y. G., Kang, M. S., & Heo, J. (2014). Clustering performance comparison using K-means and expectation maximization algorithms. Biotechnology, Biotechnological Equipment, 28(sup1), S44–S48. http://doi.org/10.1080/13102818.2014.949045

Jureta, I. J., Faulkner, S., and Thiran, P. (2007). Dynamic Requirements Specification For Adaptable And Open Service Systems,. Requirements Engineering Conference

Kell, S. (2008). A Survey of Practical Software Adaptation Techniques. Journal of Universal Computer Science, vol. 14, no. 13, 2110-2157.

Keller, R., and Hölzle, U. (1998). Binary Component Adaptation. ECCOP '98 Proceedings of the 12th European Conference on Object-Oriented Programming, Springer-Verlag, London, UK.

Kim, H., Song, H. Y. 2012. Formulating Human Mobility Model in a Form of Continuous Time Markov Chain. 3rd Intl. Conf. Ambient Systems, Networks & Technologies.

King, T. M., Ramirez, A. E., Cruz, R. and Clarke. P. J. (2007). An Integrated Self-Testing Framework For Autonomic Computing Systems. Journal of Computers, 2(9): 37-249.

KNBS, 2017, Access To Computer Service By County And District, https://www.knbs.or.ke/download/access-to-computer-service-by-county-and-district/, Updated 2017, Accessed 10th January 2018.

La, H. J., Kim, S. D. (2011). Static and dynamic adaptations for service-based systems, Journal of In Information and Software Technology, Volume 53, Issue 12, 2011, Pages 1275-1296, ISSN 0950-5849.

Le, Q. V. (2015). A Tutorial on Deep Learning: Nonlinear Classifiers and The Back propagation Algorithm, http://robotics.stanford.edu/~quocle/tutorial1:pdf, accessed on 11th August 2016.

Li, G., Liao, L., Song, D., Wang, J., Sun, F., and Liang, G. (2013). A Self-healing Framework for QoS-Aware Web Service Composition via Case-Based Reasoning.. In Proceedings of APWeb.

Lei, Z., and Georganas, N., (2001). Context-Based Media Adaptation In Pervasive Computing. Proceeding of Canadian Conference on Electrical and Computer Engineering, Toronto, Canada.Conference on Service-Oriented Computing and Applications, 80-87.

Lemos, R., de et al. (2013). Software Engineering For Self-Adaptive Systems: A Second Research Roadmap. Lecture Notes In Computer Science, Vol. 7475. Springer-Verlag, 1-32.

Lins, F. A. A., dos Santos, J. C., and Rosa, N.S. (2007). Improving Transparent Adaptability In Web Service Composition. IEEE International Sliwa, L., Gleba, K., And Amanowicz, M. (2010). Adaptation Framework For Web Services Provisionin Tactical Environment. Within the project supported by Polish R&D funds,

Lorenzoli, D., Mussino, M. P., Sichel, A., and Tosi, D. (2006). A SOA Based Self-Adaptive Personal Mobility Manager. IEEE International Conference on Services Computing, 479 – 486.

Mateescu, R., Poizat, P., and Salaün, G. (2008). Adaptation Of Service Protocols Using Process Algebra And On-The-Fly Reduction Techniques. Proceedings of the 6th International Conference on Service-Oriented Computing: Sydney, Australia,.

Maurer, M., Brandic, I., Emeakaroha, V., Dustdar, S. (2010). Towards knowledge management in self-adaptable clouds. Fourth International Workshop of Software Engineering for Adaptive Service-Oriented Systems (SEASS '10), Miami, Florida, USA.

Morin, B., Barais, O., Jézéquel, J.-M., Fleurey F., and Solberg, A. (2009). Models@Run.Time To Support Dynamic Adaptation. IEEE Computer, vol. 42, no. 10, 44-51.

Motahari-Nezhad, H., Bartolini, C., Graupner, S., and Spence, S. (2012). Adaptive Case Management In The Social Enterprise. in Service-Oriented Computing. Springer.

Moyano, F., Baudry, B., Lopez, J. (2013). Towards Trust-Aware and Self-Adaptive Systems. In Proc. of IFIPTM.

Muller J, Bakkum DJ, Hierlemann A. Sub-millisecond closed-loop feedback stimulation between arbitrary sets of individual neurons. Front Neural Circuits. 2013;6 [PMC free article] [PubMed]

Najafabadi, M. M., Villanustre, F, Khoshgoftaar, T. M, Seliya N, Wald, R, Muharemagic, E. (2015). Deep learning applications and challenges in big data analytics. Journal of Big Data;2(1):1–21.

Nallur, V., and Bahsoon, R. (2009). Self-Optimizing Architecture For Ensuring Quality Attributes In The Cloud, In Proc. 8th Working IEEE/IFIP Conference on Software Architecture: Cambridge, UK.

Namuye, S., Mutanu, L., Chege, G., Macharia, J. (2014). Leveraging health through the enhancement of information access using Mobile and service oriented technology. In proceedings of the IST-Africa Conference.

Newman, P., and Kotonya, G. (2012). Managing Resource Contention in Embedded Service-Oriented Systems with Dynamic Orchestration. In Proceedings of 10th International Conference on Service Oriented Computing.

Orriens, B., and Yang, J. (2006). A Rule Driven Approach For Developing Adaptive Service Oriented Business Collaboration. Proceedings of the IEEE International Conference on Services Computing,Chicago, Illinois,.

Pathan, K. J., Reiff-Marganiec, S., Shaikh, A. A., and Channa, N., (2011). Reaching Activities by Places in the Context-Aware Environments Using Software Sensors. Journal of Emerging Trends in Computing and Information Sciences, VOL. 2, NO. 12, ISSN 2079-8407

Papazoglou, M. P., Traverso, P., Distar, S., and Leymann, F. (2007). Service Oriented Computing: State Of The Art And Research Challenges. Published by the IEEE Computer Society, Vol. 40, 38-45.

Pardalos, P.M., Georgiev, P. G., Papajorgji, P., Neugaard, B. (2013). Systems Analysis Tools for Better Health Care Delivery. Springer Science & Business Media.

Parraa, C., Blancb, X., Clevea, A., and Duchien, L. (2011). Unifying Design And Runtime Software Adaptation Using Aspect Models. Journal Science of Computer Programming archive, Holland.

Platz M, Rapp J, Groessler M, Niehaus E, Babu A, Soman B (2014). Mathematical Modelling of GIS Tailored GUI Design with the Application of Spatial Fuzzy Logic. Universität Koblenz-Landau

Psaier H., Juszczyk L., Skopik F., Schall D., Dustdar S. (2011) Runtime Behavior Monitoring and Self-Adaptation in Service-Oriented Systems. In: Dustdar S., Schall D., Skopik F., Juszczyk L., Psaier H. (eds) Socially Enhanced Services Computing. Springer, Vienna

Piotrowski, A. P., and Jarosław, J. N., (2013) A comparison of methods to avoid overfitting in neural networks training in the case of catchment runoff modelling. Journal of Hydrology, pp 97-111,

Reed, R.D., and Marks R. J., (1999) Neural Smithing, MIT Press, Cambridge, Mass,

Robinson, D., and Kotonya, G. (2009). A Runtime Quality Architecture For Service-Oriented Systems. Proceedings of the 7th International Joint Conference on Service-Oriented Computing, Stolkholm, Sweden.

Robinson, D., Kotonya, G.: A Runtime Quality Architecture for Service-Oriented Systems. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 468–482. Springer, Heidelberg (2008)

Romero, D., Hermosillo, G., Taherkordi, A., Nzekwa, R., Rouvoy, R., and Eliassen, F. (2013). The DigiHome Service□Oriented Platform. Software: Practice and Experience, Wiley-Blackwell.

Roohi, F. (2013). Artificial Neural Network Approach to Clustering. The International Journal Of Engineering And Science (Ijes), Volume 2, Issue 3, 33-38 Issn: 2319 – 1813 Isbn: 2319 – 1805

Rosen, M.,Lublinsky, B., Smith, K. T., and Balcer, M. J. (2008). Applied SOA: Service-Oriented Architecture And Design Strategies. Wiley Publishing, Inc.

Salehie, M., and Tahvildari, L. (2009) Self-adaptive software: Landscape and research challenges. ACM Trans. Auton. Adapt. Syst. , New York, NY, USA, pp. 14

Schumann, J., Gupta, P., Jacklin, S. (2005). Toward Verification And Validation Of Adaptive Aircraft Controllers. In: Proc. IEEE Aerospace Conference, IEEE Press.

Sevarac, Z. (2014). Neural Network Based Brain-Computer Interfaces, Brain Awareness Week 2015 Conference, University of Belgrade - School of Electrical Engineering.

Shetty, S.D., Vadivel, S., Vaghella, S. (2010). Weka Based Desktop Data Mining as Web Service.

World Academy of Science, Engineering and Technology

Siljee, J., Bosloper, I., Nijhuis J., and Hammer, D. (2005) DySOA: Making Service Systems Self-Adaptive. ICSOC Lecture Notes in Computer Science, Volume 3826/2005, 255-268, Amsterdam, The Netherlands.

Skałkowski, K., SłOta, R., KróL, D,. and Kitowski, J. (2013). Qos-Based Storage Resources Provisioning For Grid Applications. Published in Future Generation Computer Systems Journal, Volume 29.

Śliwa, J., Gleba, K., Amanowicz, M. (2010). Adaptation Framework foR web services provision in tactical environment. Military Communications and Information Systems Conference, Wrocław, Poland, pp. 52-67

Sommerville, I. (2016). Software Enginnering,. Pearson Education Limited, 10th Edition.

Swaminathan, R. K. (2008). Self Configuring And Self Healing Web Services In Complex Software Systems, CECS IT project report, Part of the Special Projects Group: University of Waterloo, Waterloo.

Taiani, F., Killijan, M., Fabre, J. (2009) COSMOPEN: dynamic reverse engineering on a budget. How cheap observation techniques can be used to reconstruct complex multi-level behaviour, Journal of Software: Practice and Experience

Tamura, G., Villegas, N., M"uller, H. et al. (2013). Towards Practical Runtime Verification and Validation of Self-Adaptive Software Systems. LNCS, vol. 7475 108-132, Springer.

Tanaka, M., and Ishida, T. (2008). Predicting And Learning Executability Of Composite Web Services. Proceedings of the 6th International Conference on Service-Oriented Computing, Sydney, Australia.

Tarasyuk, A., Troubitsyna, E., Laibinis, L. (2012). Formal Modelling and Verification of Service-Oriented Systems in Probabilistic Event-B. International Conference on Integrated Formal Methods. Springer Berlin Heidelberg, 2012.

Tartler, R., Lohmanna, D., Schelera F., and Spinczyka, O. (2010). An Integrated Approach For Static And Dynamic Adaptation Of System Software. Knowledge-Based Systems, Volume 23, Issue 7, 704-720.

Taylor, R. N., Medvidovic, N., and Oreizy, P. (2009). Architectural Styles For Runtime Software Adaptation. European Conference on Software Architecture, Joint Working IEEE/IFIP Conference European Conference on Software Architecture: 171 – 180, Cambridge,.

Tosic , V., Ma, W., Pagurek, B., Esfandiari, B. (2004). Web Service Offerings Infrastructure (WSOI) - A Management Infrastructure. In Proc. of NOMS (IEEE/IFIP Network Operations And Management Symposium), Seoul, South Korea.

Valetto, G., and Kaiser, G., (2002). A Case Study In Software Adaptation. Proceedings of the 25th International Conference on Software Engineering, Workshop on Self-healing systems, 73-78.

Wang, H., Zhou, X., Zhou, X., Liu, W., Li, W., Bouguettaya, A. (2010). Adaptive Service Composition Based On Reinforcement Learning. in Proc. ICSOC, 92-107.

Wang, Q., Quan, L. Ying, F. (2004). Online Testing Of Web-Based Applications. In: Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC), 166–169.

Weyns, D. (2012). Towards an Integrated Approach for Validating Qualities of Self-Adaptive Systems. Workshop on Dynamic Analysis.

Weyns, D., and Iftikhar, U., (2016). Model-Based Simulation At Runtime For Self-Adaptive Systems. 13th international conference on Autonomic Computing Germany.

Zeng, L., Benatallah, B., Lei, H., Ngu, A., Flaxer, D., and Chang, H. (2003). Flexible Composition Of Enterprise Web Services. Electronic Markets, Volume 13, Number 2, 141-152.

Zeng, L., Lingenfelder, C., Lei, H., and Chang, H. (2008). Event-Driven quality of service prediction", Proceedings of the 6th International Conference on Service-Oriented Computing: Sydney, Australia,.

Zhang, J., Goldsby, H., and Cheng. B. H. C. (2009). Modular Verification Of Dynamically Adaptive Systems. In Proc. 8th International Conference on Aspect Oriented Software Development, 161-172.

Zhou, G., Sohn, K., Lee, H. (2012) Online Incremental Feature Learning With De-noising Auto encoders. In: International Conference on Artificial Intelligence and Statistics. JMLR.org. 1453–1461