# A PROCESS MODEL FOR THE FORMALISATION OF QUALITY ATTRIBUTES OF SERVICE-BASED SOFTWARE SYSTEMS

***Manoj Lall[1*], John A. Van Der Poll[2], and Lucas M. Venter[3]***

[1]Department of Computer Science, Tshwane University of Technology, South Africa
[2]Graduate School of Business Leadership, University of South Africa, South Africa
[3]Research Directorate, North-West University, South Africa

Email: LallM@tut.ac.za[1*](corresponding author), vdpolja@unisa.ac.za[2], Lucas.Venter@nwu.ac.za[3]

DOI: https://doi.org/10.22452/mjcs.vol32no4.3

***ABSTRACT***

*The benefits offered by software architectures include providing an improved understanding of high-level relationships amongst systems components and facilitating in making principled decisions between alternatives. A vast majority of architectural decisions are focused on realising functional requirements and often ignore the non-functional requirements (quality attributes) altogether or include them as an afterthought. This has a negative impact on the overall acceptance of the software system by its intended users. Moreover, these architectures are specified using diagrammatic notations such as boxes and lines as connectors between them. As a result, such semi-formal representations lack precision in their definitions and may lead to misinterpretations of the architecture resulting in errors being introduced at later stages of development. Although several Architectural Description Languages (ADLs) exist for the formalisation of systems architectures, few are designed for formalising non-functional requirements. Since ADLs concentrate on component-level representations and not on the system as a whole, their support of non-functional requirements is limited. Additionally, as each ADL normally supports one quality attribute, multiple ADLs are required to cater for various quality attributes. This articles aims to address the challenges of inadequacies of specifying quality attributes in software systems in a manner that is both formal and applicable to a wider range of quality attributes. To achieve this goal, this paper present a process model that assists in formalising the non-functional requirements of a service-based software system and eliminates the need for multiple ADLs. It demonstrates the practicability of the proposed model by applying it to one quality attribute, namely "availability", using the general purpose **Z** specification language.*

***Keywords**: **Architectural description language, availability, formal specification, non-functional requirements, quality attributes, UML modelling, Web services, Z***

## 1.0 INTRODUCTION

Satisfying the users' requirements is arguably the main objective of a software system. This often entails making careful architectural decisions. While a vast majority of architectural decisions are focused on realising the functional requirements, the non-functional requirements (NFRs) are often neglected until the end of the development cycle, or are ignored altogether. Reasons cited for this are that they are hard to define and represent precisely [1, 2]. Amongst the benefits offered by software architectures include – encourage understanding of high-level relationships amongst systems components; facilitate making principled choices amongst design alternatives; and promote analysis and description of high-level properties of the system. However, they are often specified informally or semi formally using notations such as boxes and lines as connectors between them [3]. The usefulness of such semi-formal representations is limited due to its lack of precise definitions. These imprecisions may lead to misinterpretations of the architecture resulting in errors being introduced at later stages of software development. The ADLs aim to overcome these limitations [4, 5].

In essence, ADLs provide for a precise notation with a well-defined semantics for describing software architectures represented in terms of components, connectors and their configurations [4, 3]. ADLs support architectural visualisation and model structure and behaviour of software systems. Additionally, ADLs may also be used during a system's analysis and validation stages [6]. Since, ADLs concentrate on component-level representations and not on the systems architecture as a whole, their support of non-functional requirements – which are global constraints that must be satisfied by the whole system – are limited [7]. This point serves as the impetus for this paper.

The purpose of this research is to fill this gap left by ADLs in specifying formally the quality attributes beyond component level. We achieve this by proposing a process model for specifying formally non-functional requirements of software systems. As the proposed approach is not confined to just component levels, it is suitable

for formalising quality attribute goals spanning the whole system. The formalisation of the quality attributes are important as it subsequently leads to reduction in the cost of correcting errors arising from imprecision inherent in specifications written in an informal or a semi-formal notation. In addition, formalisms can enable the benefits of formal reasoning about the system before its actual implementation.

## 2.0 BACKGROUND

Several ADLs have been proposed, each with their unique characteristics and capabilities in support of certain architectural facets and following a particular approach to the specification and evaluation of an architecture. Examples of well-known ADLs include C2 [8], Darwin [9], MetaH [10], Rapide [11], and Wright [12]. However, for a comprehensive analysis of quality attributes, multiple ADLs are required as each ADL normally analyses only one attribute [13, 3]. Additionally, ADLs focus on component-level representations and not on the system as a whole. Therefore, their support for non-functional requirements, i.e. constraints that must be satisfied by the whole system, are inadequate [7].

As mentioned earlier, several benefits may be derived by using a specific ADL, however constraints such as being applicable to only a particular operational domain or only being suitable for a particular analysis techniques (e.g. testing, model and consistence checking) have contributed to their slow adoption by practitioners [14]. A further factor contributing to the slow adoption of ADLs in practice is the lack of understanding of the benefits of formalism offered by the use of ADLs. In contrast, the Unified Modelling Language (UML) has become the de facto standard for modelling system architectures. Being suitable to a wide range of operational domains, the UML is often seen as a successor to the existing ADLs [15-17].

The UML makes use of the Object-oriented paradigm for visualising, specifying, constructing and documenting the artefacts of software systems [18]. The UML is defined in terms of its meta-models, consisting of an abstract syntax in a UML class diagram, a static semantics in the Object-Constraint Language (OCL), and a dynamic semantics, mainly in English [19, 20]. Although the UML utilises numerous diagrams to model a software system, they are mostly focused at modelling the functional aspects and do not provide for a standardised way to model non-functional requirements [21]. The non-functional requirements are incorporated into the UML model by means of extensions such as constraints, tagged values, and stereotypes [22, 23]. Despite the fact that the graphical notations of UML render the analysis and the design of systems easier, the models suffer from a weak semantics for some of its key concepts, such as associations, and specialisation – thereby making it difficult to establish the accuracy and completeness of its artefacts [24, 25]. The OCL is an attempt to formalise the semantics of a UML specification. Although OCL borrows some basic notations from mathematical set theory and logic, it is predominantly a textual language and therefore lacks a complete formal semantics [26, 27]. Therefore, the OCL might not give the same advantage, such as rigor in their reasoning and proofs which several ADLs offer.

In summary, the ADLs allow for a formal treatment of system architecture but are limited to a particular architectural style and operational domain, whereas the UML supports multiple architectural views and is suitable for modelling in a wide range of application domains. However, both prove inadequate in specifying quality attributes of software systems in a formal manner that could be used in a wide range of application domains. In essence, what is required is more formalism for a wider range of operational domains. This is what our work aims to achieve. In our proposed process model, the wide range of operational domains that it can be applied to is achieved by using UML and the formalism is enhanced by making use of **Z** notations. We use Z since it has a proven track record for the formal specification of systems. It provides for precision, clarity and rigour. The behavioural and structural aspects of the system is captured using UML constructs and thereafter mapped on to **Z** notations to realise the formalism.

**Z** [28, 29] is a well-known notation used in formaly specifying software. **Z** is based on first-order logic and a strongly typed fragment of Zermelo-Fraenkel set theory [30, 31]. Its extensible toolkit of mathematical notations together with its schema notation for specifying structures in the system, and for structuring the specification itself has enabled it to be used in specifying various types of systems [32, 33]. For instance, **Z** has been successfully deployed in several application domains including safety critical systems, security systems, and other general purpose systems. The use of mathematical notation for specifying a system offers benefits such as precision, clarity and rigor in its reasoning and proofs, leading to the early detection of problems in the systems requirements [34]. Since our work falls in the domain of software *specification* and owing to its simplicity, we use **Z** instead of more comprehensive development environments like Event-B/Rodin [35].

## 3.0 RELATED WORK

A review of related literature suggests there is a notably lack of support for incorporating non-functional requirements into software systems. Some researchers such as Vestal [10] and Shaw et al., [36] have incorporated support for NFRs into ADLs. However their support is limited to a few NFRs. For example, MetaH provides support for scheduleability, reliability and security whereas UniCon is limited to modelling scheduleability only [4]. Other researchers such as Khaled et al.,[37] have addressed this problem by proposing an extension to the Wright ADL. Another approach that is independent of a particular ADL is as proposed in NoFun [38] and Process NFL [39]. In NoFun the authors have developed an ad-hoc notation for describing components and connectors. Whereas, ProcessNFL is a language designed to consider specific Non-functional characteristics like the correlations and conflicts. However, both languages lack the precision in their semantics to support formal reasoning. Modeling QoS in service-based software system specifically have been addressed by researchers such as Ying et al.,[40] and Wang et al.,[41]. However, their work concentrated on modelling selection and composition issues respectively. Ran [42] have focused on the discoverability issues of Web services. To the best of our knowledge, our model is novel in the sense that it is neither confined to any particular quality attribute nor does it require knowledge of various ADL extensions. We achieve this generality by making use of UML, a general-purpose modelling language, and **Z**, a general purpose formal notation.

The rest of this article is structured as follows. In Section 4, we present the *P*rocess *M*odel for the *For*malisation of *N*on-functional *R*equirements (PMForNR) of Service based software systems. The PMForNR model as applied to the non-functional requirement "availability" is exemplified in Section 5. The formal specification is developed using the **Z**'s Established Strategy as presented in Potter et al., [43]. Our main contributions and directions for future work appear in Section 6.

## 4.0 THE PROCESS MODEL FOR THE FORMALISATION OF NON-FUNCTIONAL REQUIREMENTS

In the domain of software engineering, a process model is an interconnected sequence of activities, transformations and events that represent strategies for accomplishing software development [44]. It acts as a template for the creation of other instances of a process of the same type categorised together into a model. In this article we have used "availability" as an instance of the quality attribute on which our proposed PMForNR process model has been exemplified; it can be used for the formalisation of other quality attributes as well. The main building blocks of PMForNR process model are shown in Fig. 1.

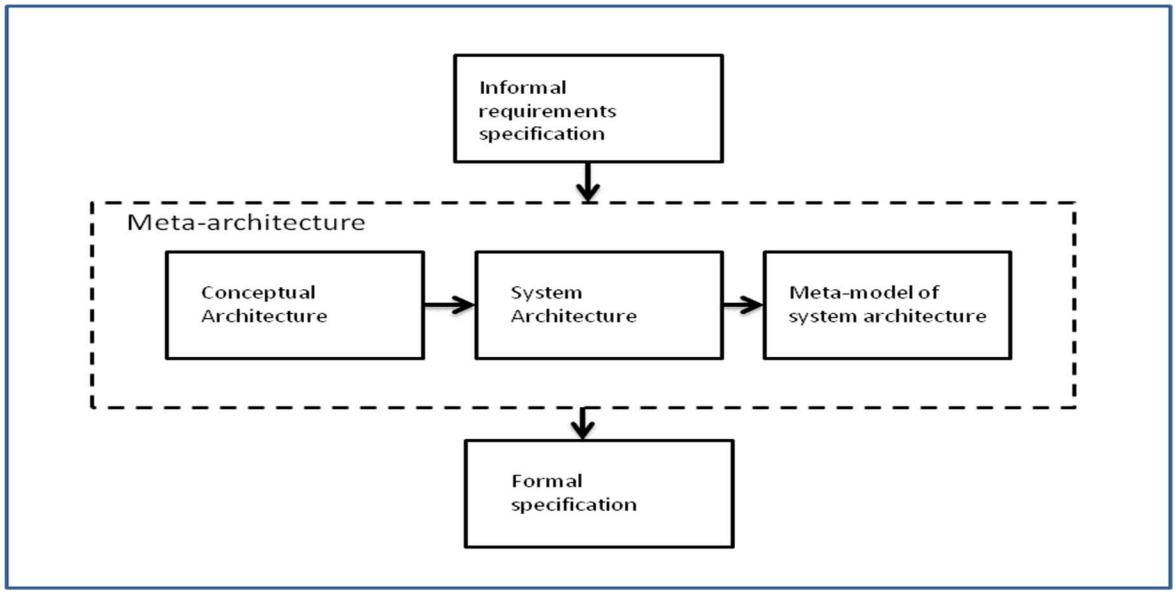


Fig.1. The PMForNR process model

The PMForNR process of formalising a non-functional requirement specified in a natural language is described below (refer Fig. 1):

- Step 1: A non-functional requirement of the system is specified in a natural language.
- Step 2: A conceptual architecture representing entities that support the non-functional requirement in question, is identified. These entities could be platform independent technologies, paradigms, and mechanisms that support the non-functional requirements. We view a conceptual architecture essentially as a block diagram representing the entities in a visual language.
- Step 3: A systems architecture depicting the structure and the interactions of the entities of the conceptual architecture is developed. In essence, the systems architecture models the structure and the behaviour of the entities in the conceptual architecture in support of the non-functional requirement.
- Step 4: Meta-models of the structure and the behaviour of the entities of the system architecture are defined. We have used UML diagrams for this purpose. The resulting UML artefacts are semi-formal representations of the non-functional requirements specification.
- Step 5: The semi-formal meta-models are subsequently mapped to a formal representation. We make use of the method described in Shroff and France [45] to map the UML representations to **Z.**

In the following section, we demonstrate the feasibility of PMForNR in formalising the non-functional requirements. We select "availability" for this purpose, since support for this non-functional requirement has the potential of extending the domain of applications where Web services could be used. For example, applications that require Web services to be readily available include those belonging to Web service co-ordination infrastructures and those belonging to mission-critical systems, such as air traffic control systems.

## 5.0 THE "AVAILABILITY" CASE STUDY

Following the steps defined in Section 4, the first step requires that a natural language representation of the non-functional requirements be specified. The ***availability*** *of a Web service* may be defined as the proportion of time that a service is in a functioning condition [46]. It should be noted that non-functional requirements can rarely be said to be absolutely satisfied. Their implementation is considered to be satisfied only if it is within acceptable limits [47]. As an example, depending on the requirements specification, it would not be unreasonable to say that a service is considered to be available if its availability is approximately 99%.

### 5.1 The Conceptual Architecture

Step 2 in the PMForNR requires the identification of entities that can augment the non-functional requirements. These can be seen as technical components that support quality attributes (in this case, availability). An important factor impacting on the availability of Web services is the availability of the infrastructure required to access the Web service. We make use of the Cloud. A cyber infrastructure, such as the Cloud, is not in a position to provide quality assurance on its own but requires supporting entities such as human beings or (software) agents to act in a suitable manner. The use of Agent technology in conjunction with the Cloud is utilised to augment the availability of the service-based applications. Fundamentally, the Cloud provides appropriate supporting protocols and mechanisms to facilitate optimal functioning of the cyber component and the agent paradigm makes provision for robustness and fault tolerance of the interactions. Other important components of the conceptual model in support of "availability" are the security of its entities and the messages sent between the entities to achieve the required functionalities. Trust relationships between the communicating parties are vital for supporting the availability of services in service-based applications and therefore form part of our conceptual architecture. Although, the above-mentioned entities play a crucial role in supporting the availability requirement of the services, the availability of the Web service function itself is supported by using *communities* of Web services. A community in the context of Web services may be defined as a collection of Web services that perform similar functions [48]. Once a Web service in a community is unavailable for a particular request, it is replaced by another from that community, thus improving the availability of the Web service. Fig. 2 represents a conceptual architecture for augmenting the availability of service-based applications.

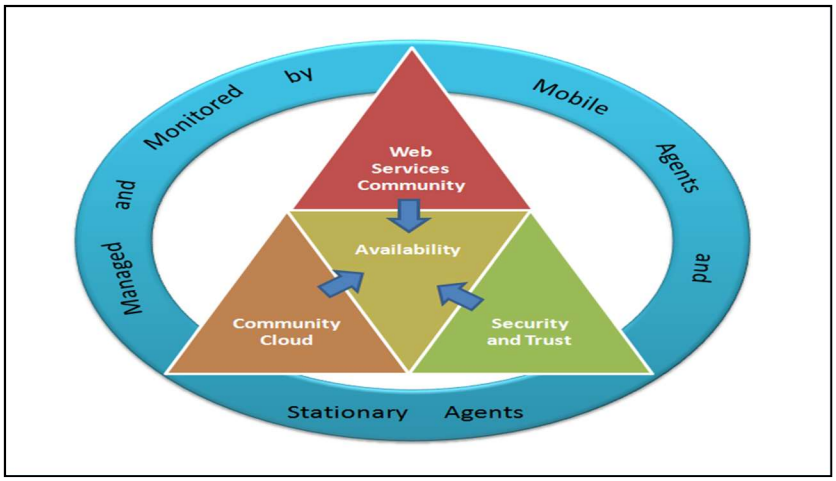


Fig. 2. The Conceptual architecture for augmenting availability

The entities in the conceptual architecture and their interactions are detailed in the Systems architecture presented in Section 5.2. The next step in the formalisation of the non-functional requirement using the model is to establish the systems architecture.

### 5.2 Systems Architecture

In our systems architecture, the support for availability of Web services-based applications is approached from two directions. On the one hand, infrastructural support for messages transmission between the requester and the provider is considered. The main reason for considering this factor is that even though a Web service may be in a position to process a client's request, it would be of little use if the request itself was delayed or could not reach the Web service. As the Web services are located on different hosts distributed widely across the Internet, promoting the "reachability" of these Web services amounts to an improvement in the infrastructure facilities, i.e. in the connectivity between these hosts, and the hardware/middleware on which these Web services are positioned and operate. To enhance the reachability of Web services, our systems architecture makes use of virtualisation techniques in conjunction with the idea of community-cloud. Virtualisation ensures improved infrastructural and middleware availability [49], whereas, the community-cloud allows for an improved utilisation of underutilised resources in the community. It must be noted that the members of a community-cloud are predominantly those forming the Web services community, and those interested in providing infrastructural and/or middleware support, i.e. offering Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) [49, 50]. Fig. 3 shows the link between the Community of Web services and the community-cloud supported by the virtualisation of the physical resources in our PMForNR.

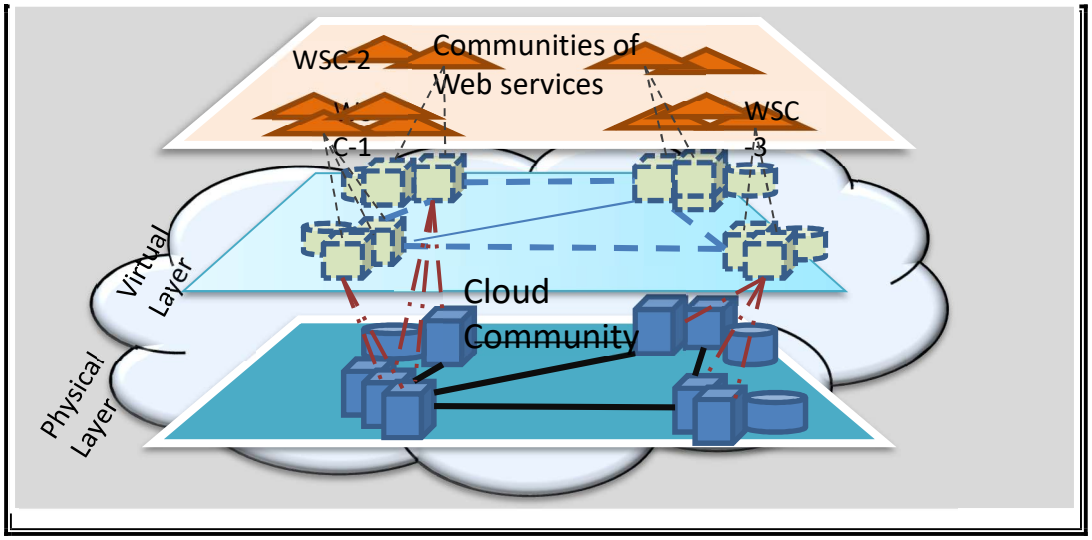


Fig. 3. A layered view of the PMForNR systems architecture

The second aspect in the approach to augmenting the availability of the Web services is with respect to the improvement is the "readiness" to process the request from a client. This is influenced by mechanisms for enhancing the possibility of finding the Web service in a state such that it can act on the request immediately. An enhancement to the readiness of a Web service to process a request from a client is achieved by increasing the amount of time the Web service spends in a state that will enable it to respond to an incoming request without having to wait for completing a previous request or for particular resources. This can be achieved by pre-processing the requests before they are handed over to the Web services. Examples of pre-processing activates that could be performed on incoming messages are scheduling and assigning of priorities, aimed at reducing the response time and improving on the throughput rate. In PMForNF systems architecture, the responsibilities of pre-processing are assigned to stationary agents. In certain extreme cases, a Web service may be in a state from which it cannot recover on its own due to major problems caused by one or more external entities, or due to some internal problems. In situations such as these, where a Web service does not respond at all, or is very slow in responding, the PMForNR system architecture relies on the communities of Web services to provide a replacement Web service.

Although it has been argued that an improvement in the "reachability" of the services and their "readiness" to respond to requests from clients can improve the overall availability of a Web service-based application; they are of little use if a Web service does not react to the requests owing to a lack of trust between the clients and the Web service. Issues related to security and trust in the context of agents and Web services are important concepts and hence form an integral part of our systems architecture. The systems architecture is depicted in Fig. 4.

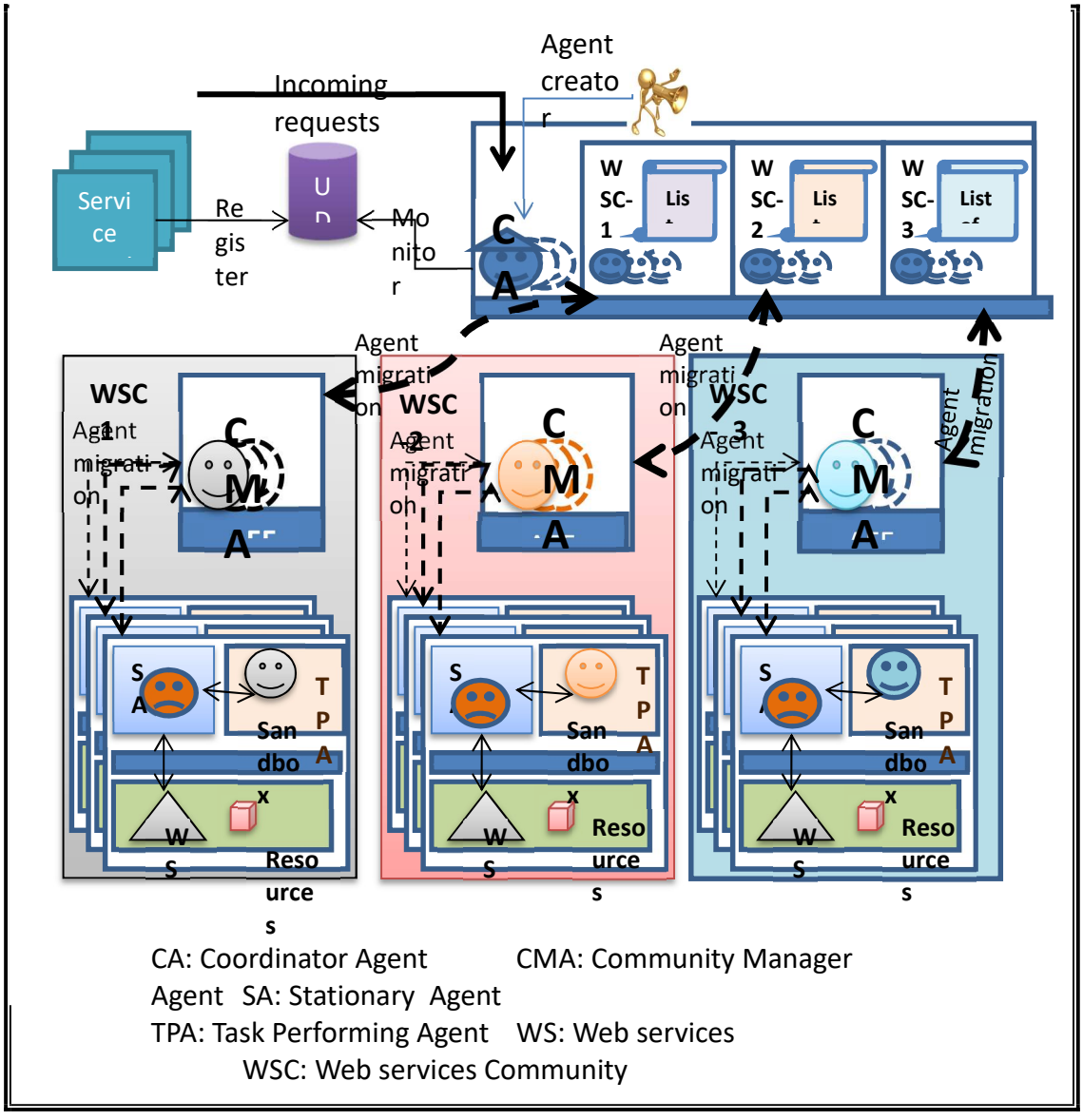


Fig. 4. Systems architecture in support of availability [51]

**The Coordinator Agent (CA):** A key responsibility of CA is to create communities of Web services. The community members, though separated geographically, are brought together in a list containing their Uniform Resource Identifiers (URIs). For all practical purposes, each list represents a community and are managed by a Community Management Agent (CMA). The CMAs are clones of CA. The CA is also tasked with scrutinising and decomposing incoming requests such that each unit of request can be satisfied by a particular community. In addition, the CA is also responsible for collecting and prioritising all the messages that the CMAs bring with them on their return from the various communities.

**Community Manager Agents (CMAs)**: After a CMA is created it migrates with a copy of the list (which acts as a local registry of the community) to a host that is strategically placed within the community. Upon arriving at its selected destination, the CMA creates Task Performing Agents (TPAs). TPAs are clones of the CMA responsible for interacting with stationary agents in order to invoke services offered by the Web service. The stationary agents act as proxies for the Web services. On completion of the tasks at the Web server, TPAs return to the host where they were created and deliver the results to the CMA. The CMA collects the results from the TPA and delivers it to the CA. The results are assembled by the CA (if multiple communities were involved) and returned to the client. The sequence of interactions between the various components of the systems architecture is depicted in Fig. 5.

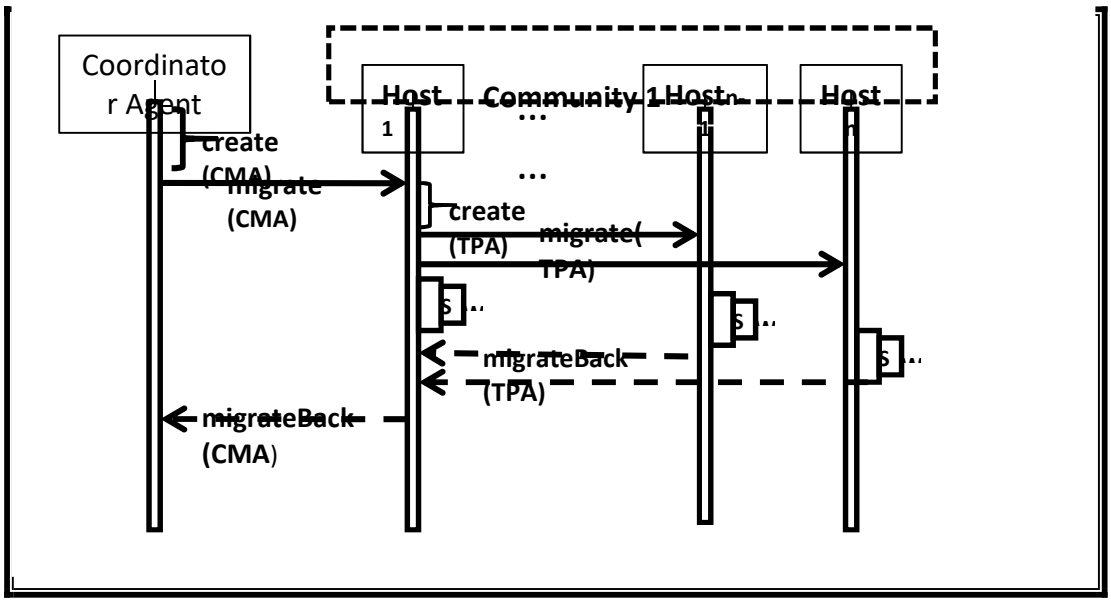


Fig. 5. Interaction sequence amongst the community entities [51].

For agents and Web services to function effectively, they make use of various kinds of resources owned by different providers. The CMAs, in conjunction with the stationary agents (SAs), are also responsible for maintaining an account of the resources used in performing a task and the providers of those resources. Such information is useful in making payments to the resource providers; encouraging them to provide high QoS; and serving as a means to attract other service providers to the community.

**Stationary Agent (SA)**: On arrival at the Web services host, the TPA is placed in a secure environment to prevent it from executing any security attack against the host or other entities on the host. The SA then performs a credential check on the agent, in order to establish its trustworthiness. Besides dealing with the security and trust issues, the stationary agent is responsible for managing the resources that may be required by the Web services.

The next step in the PMForNR is the creation of meta-models, presented in Section 5.3. These meta-models are UML artefacts modelling the structure and behaviour of the entities represented in the systems architecture. These UML models are essentially a semi-formal representation of the systems architecture.

### 5.3. The UML Models

As mentioned in Section 5.1, the main components of the Conceptual architecture are the Community Cloud, the Web service community, the Agent system, the Gateway, and the Security and Trust subsystem. The Community Cloud provides the hardware platform on which the Web services and the agents operate; the Web services community caters for the functional requirements, and the Gateway enables the ACL-SOAP-ACL conversions. The Security and Trust system enhances confidence in the communicating parties. The Agent system fulfilled a very important function in this architecture – it caters for the non-functional requirements by making use of certain attributes of the Community Cloud and the Security and Trust system.

The static view of the agent system represented by a Class diagram is depicted in Fig. 6. Agent is declared as an abstract class and Community Management Agent and Coordinator Agent are specialisations of this (Agent) class. Coordinator Agent creates the Community Management Agent which in turn creates the Task Performing Agents. The Task Performing Agents migrate to the Stationary Agents that are co-located with the Web services. The communication between the Stationary Agent and the Web services takes place via the Gateway.

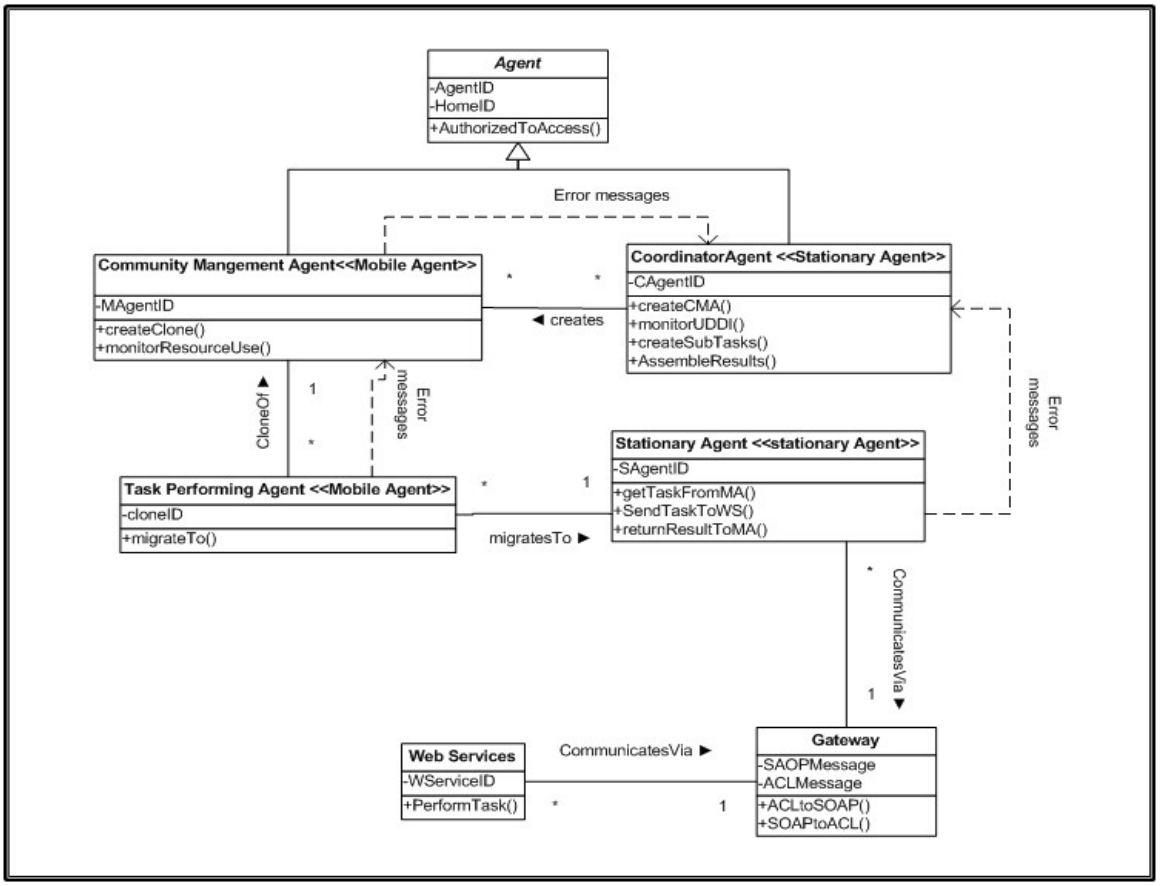


Fig. 6. A Class diagram of the PMForNR systems architecture (Synthesised by the authors)

The behaviour of the Agent system is depicted in Fig. 7. The Coordinator Agent monitors the Universal Description, Discovery and Integration (UDDI) or other similar structures for relevant Web services. It then creates the Communities of Web services by collecting relevant information from the UDDI. The Coordinator Agent then creates the Community Management Agent and provides details of the Web services in that community. The Community Management Agent thereafter creates the Task Performing Agent and provides information to enable it to migrate and interact with the Stationary Agent which is co-located with the Web services. On successful completion of the task, the agents migrate back and present the relevant information to their creators.

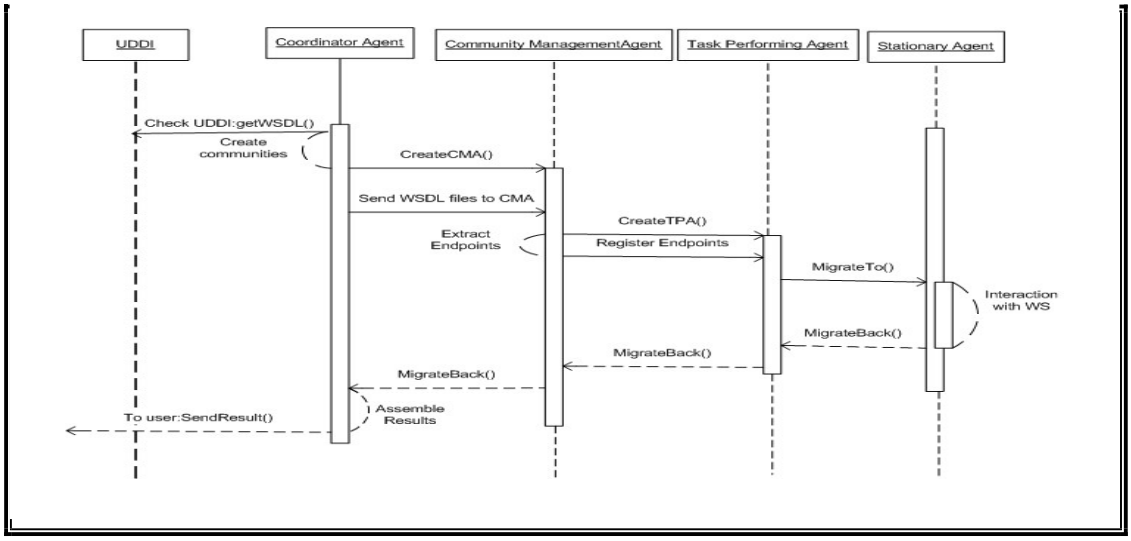


Fig. 7. Sequence diagram of agent interactions in the PMForNR systems architecture (Synthesised by the authors)

The final step in modelling the non-functional requirements is to map the meta-models of the meta-architecture to a formal notation. In Section 5.4, we present the formal specification using **Z**. It must be noted that although the formal specification could be presented in greater depth, the focus of this work is to instead demonstrate the feasibility of such a model. Hence the specification is conceptual.

### 5.4 Formal Specification

In the PMForNR systems architecture, presented in Section 5.2, both the mobile and the stationary constructs are modelled as specialisations of an agent. In its basic form, an agent has a unique identity, is created at a particular host and is created to perform a certain task on behalf of its user. To complete its task, the mobile agent follows an itinerary and it may require the use of certain resources at a particular host. The host on which it is created is designated as the agent's home. The agent's itinerary may be predefined, or may be created and updated as the agent migrates. In the PMForNR systems architecture, the CMAs follow a predefined itinerary (to migrate from the CA to a host in the Web service community) and the TPAs create their itinerary dynamically.

In the following sub-sections we highlight the steps in formalising the PMForNR systems architecture by following the Established Strategy for presenting a **Z** specification [43]. A more comprehensive specification appears in Lall et al.,[46].

#### 5.4.1 Given Sets and Global Variables

It is customary in **Z** to first define the basic types of the specification. In our specification *IDENTITY* represents all possible mobile and stationary agents; *NODE* is a set of all nodes in the system, while *RESOURCE* is the set of all resources that may be needed for the execution of the agents and the Web services. *TASK* represents the set of all tasks that may need to be performed by the agent system and the Web services, while *REPORT* is the set of all messages that may be returned to the user of the system. *RESULT* is the outcome of all tasks assigned to each Web service community. Subsequently the basic types are:

$$[IDENTITY, NODE, RESOURCE, TASK, RESULT, REPORT]$$

$$BOOLEAN ::= true \mid false$$

In the following Axiomatic definition, the host on which entities such as mobile agent, stationary agent or the Web service are located, has a number of *resources* that needs to be available (*resAvail*) for these entities to perform their allocated tasks.

$$resources, resAvail : \mathbb{P}\ RESOURCE$$
$$identities : \mathbb{P}\ IDENTITY$$
$$userTasks : \mathbb{P}\ TASK$$

The following functions are defined – *createMAChild, createSAChild, authorisedAccess* and *trusted.* The *createMAChild* and *createSAChild* are functions from an agent to either a mobile or a stationary agent created by that agent. These functions are partial as not all agents are required to create a child (mobile or stationary) agent. When a mobile agent arrives at the Web services host, the stationary agent representing the Web services, needs to know whether the mobile agent can be allowed access to the services available at that host. Similarly, the mobile agent visiting a Web service host needs to establish if the stationary agent at that host can be trusted. The functions *trusted* and *authorisedAccess* provide the answers. Schemas for *Mobile_Agent* and *Stationary_Agent* are defined later in this article.

$$createMAChild : Agent \nrightarrow Mobile\_Agent$$
$$createSAChild : Agent \nrightarrow Stationary\_Agent$$
$$trusted : Mobile\_Agent \rightarrow BOOLEAN$$
$$authorisedAccess : Mobile\_Agent \times Stationary\_Agent \rightarrow BOOLEAN$$

The functions *createMAChild* and *createSAChild* are partial injective functions as no mobile or stationary agent can have more than one agent as parent and not all agents create child agents.

**Example 1**

Suppose *m1* and *m2* are mobile agents and *s1* and *s2* are stationary agents, then $authorisedAccess(m1, s1) \mapsto true$ if *m1* has been authorised to access *s1* and $authorisedAccess(m2, s2) \mapsto false$ if *m2* does not have access to *s2*.

The state a Web service can be in at any particular moment in time is comparable to the states in the life-cycle of a thread in an Operating system, and is defined by the free type STATUS. The four states – idle, busy, blocked, and crashed are mutually exclusive, i.e. a Web service can only be in one state at any given time (to avoid ambiguity with the state space of a Web service, we refer to a state as status in the **Z** specification below). A more detailed discussion on the states of a Web service is presented in Lall et al., [46].

$$STATUS ::= idle \mid busy \mid blocked \mid crashed$$

The function *createdOn* allows any agent to discover the unique node where a particular agent was created. This is useful for determining the extent to which an agent may be trusted.

$$createdOn : IDENTITY \nrightarrow NODE$$

Following the principle of maximising communication with the users [52], a large set of responses may be generated by the various operations:

$$REPORT ::= Creator\_of\_the\_mobile\_agent \mid Allow\_access\_to\_Web\_service \mid$$
$$Web\_Service\_added\_to\_community \mid Agent\_already\_at\_destination \mid$$
$$Web\_Service\_removed\_from\_community \mid$$
$$Agent\_migrated\_to\_new\_home \mid Agent\_identity\_already\_exists \mid$$
$$Stationary\_agent\_created\_sucessfully \mid$$
$$Webservice\_does\_not\_perform\_similar\_tasks \mid$$
$$Web\_service\_not\_part\_of\_the\_Community \mid$$
$$Parent\_of\_the\_agent\_does\_not\_exist \mid$$
$$Web\_service\_already\_part\_of\_the\_Community \mid$$
$$Mobile\_agent\_created\_sucessfully$$

### 5.4.2 Abstract State Space

An agent is modelled as having an identity (*agentID*) and a *home* where it is created. Additionally, it is created to perform a set of *tasks* for which it requires access to certain resources at a particular host, denoted by *accessResource*.

*_Agent_*

$agentID : IDENTITY$
$home : NODE$
$tasks : \mathbb{P}\ TASK$
$accessResource : NODE \nrightarrow \mathbb{P}\ RESOURCE$

---

$agentID \in identities$
$home = createdOn\ agentID$
$home \in \text{dom}\ accessResource$
$tasks \subseteq userTasks$
$accessResource\ home \subseteq resources$

Given a particular *agentID*, *createdOn* returns the node on which the particular agent was created (i.e. gives the home of the agent). This information is useful in establishing trust amongst the agents in a system. For example, knowledge of where a particular agent was created may give an indication of who created that agent, and if such agent is trusted then all agents created at that particular host may also be trusted. Agents are created for meeting certain user requirements, hence the *tasks* the agent is assigned are part of *userTasks.* At its home node, an agent has access to all the resources available at that host.

A mobile agent is modelled as an agent that has the ability to migrate to different nodes (hosts), whereas, a stationary agent has no such ability. For the purpose of this research, no distinction is made between an agent and a stationary agent.

*_Mobile_Agent_*

*Agent*
$agentItinerary : \text{seq}\ NODE$

---

$\#\ (agentItinerary) = \#\ (\text{ran}\ agentItinerary)$

For the migration of a mobile agent, the agent can follow a predefined itinerary made up of nodes or construct one as it migrates from one node to another. In the PMForNR systems architecture, an agent is given an itinerary when it is created, and it can visit a node more than once on the same journey. An agent itinerary gives an indication of where the agent was created (by determining the itinerary head), and where it has been. The number of elements in the mobile agent's itinerary is equal to the number of nodes the agent has visited. This information is useful in establishing a trust level for the agent. Besides having specific tasks, a stationary agent may be modelled to be the same as an agent.

$Stationary\_Agent \triangleq Agent$

The abstract state of a Web service is given by the schema *Webservice*. The state in which a Web service can be is represented by the variable *status* and can either be *idle*, *busy*, *blocked*, or *crashed*. The set of resources that may be required by the Web service to perform its task is represented by *resReq*, while *resAssign* represents the resources assigned to the Web service at any time while executing a task. A resource cannot be requested and assigned at the same time. Similarly, a resource cannot be simultaneously assigned and available. Component *resNeeded* represents the generic set of resources needed for each task which the Web service has been designed for.

*_Webservice_*

$Id : IDENTITY$
$status : STATUS$
$tasks : \mathbb{P}\ TASK$
$resWaitingFor, resAssign : \mathbb{P}\ RESOURCE$
$resNeeded : TASK \nrightarrow \mathbb{P}\ RESOURCE$

---

$tasks = \text{dom}\ resNeeded$
$resAssign \cap resAvail = \emptyset$
$resWaitingFor \cup resAssign \subseteq \bigcup (\text{ran}\ resNeeded)$

A community of Web services is seen as a collection of Web services that perform similar tasks. Furthermore, there must be at least two Web services that perform similar tasks in the community. A Web service community may by modelled by the following schema:

*WSCommunity*

$communityMembers : \mathbb{P}\ Webservice$
$wsCommunityTask : \mathbb{P}\ TASK$
$numberOfMembers : \mathbb{N}_1$

---

$numberOfMembers \geq 2$
$\forall\ ws : Webservice \bullet$
$\quad ws \in communityMembers \Rightarrow ws.tasks = wsCommunityTask$

### 5.4.3 Initial States

The *Init_Agent* schema, the *Init_Mobile_Agent* schema, and the *Init_Stationary_Agent* schema define the initial states of the *Agent* schema, *Mobile_Agent* schema and the *Stationary_Agent* schema, respectively. The *Init_Agent* schema indicates that initially, when an agent is created, the system assigns it an identity and a home. Additionally, it has no task or resources assigned to it. The *Init_Mobile_Agent* schema specifies that initially a mobile agent has no tasks allocated, cannot access any resources on the host, and does not have any migration plans assigned to it. Similarly, the *Init_Stationary_Agent* schema specifies that the stationary agent cannot perform any task or access any resource on its home node.

*Init_Agent*

$Agent'$
$id! : IDENTITY$
$h! : NODE$

---

$agentID' = id!$
$home' = h!$
$tasks' = \emptyset$
$accessResource' = \emptyset$

Similarly, *Init_Mobile_Agent* and *Init_Stationary_Agent* may be represented by the following schemas:

*Init_Mobile_Agent*

$Mobile\_Agent'$

---

$agentItinerary' = \langle\ \rangle$

$Init\_Stationary\_Agent \triangleq Init\_Agent$

### 5.4.4 A Proof Obligation

The next step is to show that *Init_Agent*, *Init_Mobile_Agent* and *Init_Stationary_Agent* can be realised. This requires us to show that the variables *agentID'*, *tasks'*, *accessResource'* and *agentItinerary'* are each of the type indicated and the predicates of *Agent*, *Mobile_Agent* and *Stationary_Agent* still hold given the added restrictions $agentID' = id! \land home' = h! \land tasks' = \emptyset \land accessResource' = \emptyset \land agentItinerary' = \langle\ \rangle$. Generically these are stated as the *initialisation theorem* which reduces to three proof obligations:

$\vdash \exists\ Agent' \bullet Init\_Agent$ (1)
$\vdash \exists\ Stationary\_Agent' \bullet Init\_Stationary\_Agent$ (2)
$\vdash \exists\ Mobile\_Agent' \bullet Init\_Mobile\_Agent$ (3)

**Proof**

A proof of (1) follows:

- Given the predicate $agentID = id! \land home' = h! \land tasks' = \emptyset \land accessResource' = \emptyset$, it is required to show that $agentID \in \text{IDENTITY} \land home' \in \text{NODE} \land tasks' \in \mathbb{P}\ TASK \land accessResource' \in NODE \nrightarrow \mathbb{P}\ RESOURCE$. The proof is quite apparent, since *id*! is an element of *IDENTITY*, *h*! is an element of *NODE*, $\emptyset$ is an element of $\mathbb{P}\ TASK$ and $\emptyset$ is an element of $NODE \nrightarrow \mathbb{P}\ RESOURCE$.

- Since components *agentID'* and *home'* are of type *IDENTITY* and *NODE* respectively and *tasks'*, and *accessResource'* are empty, all four predicates in *Init_Agent* hold.

As Agent and *Stationary_Agent* are represented by the same schema, the above proofs also serve as proof of (2).

The proof for (3) is obtained by proving that $agentItinerary' \in \text{seq}\ NODE$, given $agentIninerary' = \langle\ \rangle$. The proof is quite apparent, since $\langle\ \rangle$ is an element of seq *NODE*.

- Since *agentItinerary'* is empty, the predicate $agentItinerary' = \langle\ \rangle$ in schema *Init_Mobile_Agent* holds.

### 5.4.5 Partial System Operations

Referring to the sequence diagram in Fig. 7, the main operations that can be identified are create Communities, create clones of agents (or create child agents), migrate to, migrate back, assemble results, and interact with Web services (represented by Web service interface agents). The PMForNR makes extensive use of clones/child agents. A child agent assists in performing tasks (or subtasks) in parallel, on behalf of its creator. For example, the CMA may create child agents (TPAs) to perform tasks on various nodes in the community. Another example where child agents are useful in our systems architecture is when a CA creates multiple CAs to take up its responsibilities in an event when it becomes unavailable (due to security attacks carried out on it). Therefore, the operations to create child agents have a direct influence on the design of the PMForNR systems architecture. Similarly, the ability of agents to migrate closer to the Web service and negotiate on behalf of its users also promotes support for availability. It should be noted that both the mobile agents and the stationary agents are capable of creating child agents.

Creating child agents is represented by schemas *CreateMAChildAgent* and *CreateSAChildAgent* specified next.

*_CreateMAChildAgent_____*

$childAgent : Mobile\_Agent$
$childIdentity! : IDENTITY$
$childTasks? : \mathbb{P}\ TASK$
$childItinerary! : \text{seq}\ NODE$
$report! : REPORT$

---

$\forall a : Agent \bullet (CreateMAChild\ a = childAgent \land$
$childAgent.home = head\ childItinerary! \Rightarrow$
$childTasks? \subseteq a.tasks)$
$childAgent.agentID = childIdentity!$
$\forall ca1, ca2: childAgent \bullet (ca1.agentID = ca2.agentID \Rightarrow ca1 = ca2)$
$childAgent.accessResource\ (childAgent.home) \subseteq resources$
$childAgent.agentItinerary = childItinerary!$
$report! = Mobile\_agent\_created\_sucessfully$

Once a child mobile agent is created it is assigned a unique system generated identity, allocated a task/subtask, and its home is set to the node on which it is created. In addition, the child agent needs resources to perform its tasks; hence the home node must have the resources to support such tasks. An appropriate itinerary is assigned to the child mobile agent. A message indicating the successful creation of a child agent is generated.

*_CreateSAChildAgent*

*childAgent* : *Stationary_Agent*
*childIdentity*! : *IDENTITY*
*childTasks*? : ℙ *TASK*
*report*! : *REPORT*

---

∀ *a* : *Agent* • ( *CreateSAChild a* = *childAgent* ∧
*childAgent*. *home* = *createdOn a.agentID* ⇒
*childTasks*? ⊆ *a.tasks*)
*childAgent.agentID* = *childIdentity*!
∀ *ca*1, *ca*2: *childAgent* • (*ca*1.*agentID* = *ca*2.*agentID* ⇒*ca*1 = *ca*2)
*childAgent.accessResource* (*childAgent.home*) ⊆ *resources*
*report*! = *Stationary_agent_created_sucessfully*

Note that *CreateSAChildAgent* is similar to *CreateMAChildAgent*, except that a stationary agent has no itinerary defined. The agent migration (see Fig. 7) is represented by the operations *MigrateTo*() and *MigrateBack*(). The difference between the two operations is that in *MigrateBack*(), the node that the agent wishes to migrate to has been visited by that agent before. In *MigrateTo*(), the node to be visited, by a particular agent, has not been visited earlier. Thus, *MigrateBack*() is seen as a special case of *MigrateTo*(). Besides this, there is no difference between the two operations and therefore only the *MigrateTo*() operation that is modelled by the schema *MigrateTo* is shown. Upon migration, the new node (*newHome*?) to be visited is added to the agent's itinerary. The node to which the agent migrates must have the necessary resources available to the agent.

*_MigrateTo*

Δ *Mobile_Agent*
*newHome*? : *NODE*
*report*! : *REPORT*

---

*agentID′* = *agentID*
*home* = *head agentItinerary*
*home′* = *newHome*?
*agentItinerary′* = ⟨*newHome*?⟩ ⌢*agentItinerary*
*tasks* = *tasks′*
*accessResource newHome*? ⊆ *resources*
*report*! = *Agent_migrated_to_new_home*

The addition of a Web service to a community is achieved by adding Web services that perform the same tasks as those required of the community. This operation is represented by the *AddTo_WSCommunity* schema. For a new Web service to be added to a community, it must perform the same tasks as those of the community. Similarly, the removal of a Web service from a community is represented by the *RemoveFrom_WSCommunity* schema. A Web service may be removed if it no longer satisfies the requirements of the community. Both of these schemas specify appropriate user feedback.

*_AddTo_WSCommunity*

Δ *WSCommunity*
*ws*?: *Webservice*
*report*! : *REPORT*

---

*ws*? ∉ *communityMembers*
(*ws*?.*tasks* = *wsCommunityTask*) ⇒
*communityMembers′* = *communityMembers* ∪ {*ws*?}
*numberOfMembers′* = *numberOfMembers* + 1
*report*! = *Web_Service_added_to_community*

*_RemoveFrom_WSCommunity*

$\Delta$ *WSCommunity*
*ws*?: *Webservice*
*report*! : *REPORT*

---

*ws*? $\in$ *communityMembers*
(*ws*?.*tasks* $\neq$ *wsCommunityTask*) $\Rightarrow$
  *communityMembers'* = *communityMembers* $\setminus$ {*ws*?}
*numberOfMembers'* = *numberOfMembers* - 1
*report*! = *Web_Service_removed_from_community*

### 5.4.6 Enquiry Operations

To allow access to certain resources at a particular node, and to address other security concerns, it may be necessary to establish the creators of that entity (e.g. mobile or stationary agent). This may be specified by the *Query_ParentOfMA*. This operation does not change the states of the agents.

*_Query_ParentOfMA*

$\Xi$ *Agent*
*ma*? : *Mobile_Agent*
*parent*! : *Agent*
*report*! : *REPORT*

---

*ma*? $\in$ ran *CreateMAChild*
*parent*! = *CreateMAChild*$\sim$*(ma*?)
*report*! = *Creator_of_the_mobile_agent*

### 5.4.7 Tabulating Preconditions

As prescribed by the Established Strategy for presenting a **Z** specification, we next summarise the partial operations together with their inputs, outputs, and their preconditions Table 1 below.

Table 1: Summary of partial operations of our systems architecture

| Operation | Input and Output | Preconditions |
|---|---|---|
| *CreateMAChildAgent* | *childIdentity*!: *IDENTITY*<br>*childTasks*? : $\mathbb{P}$ *TASK*<br>*childItinerary*! : seq *NODE*<br>*report*! : *REPORT* | *childTasks*? $\neq \emptyset$<br>*childItinerary*! $\neq \langle\ \rangle$<br>*childIdentity*! $\notin$ *identities* |
| *CreateSAChildAgent* | *childIdentity*!: *IDENTITY*<br>*childTasks*? : $\mathbb{P}$ *TASK*<br>*report*! : *REPORT* | *childTasks*? $\neq \emptyset$<br>*childIdentity*! $\notin$ *identities* |
| *MigrateTo* | *newHome*?: *NODE*<br>*report*! : *REPORT* | *newHome*? $\notin$ ran *agentItinerary* |
| *AddTo_WSCommunity* | *ws*? : *Webservice*<br>*report*! : *REPORT* | *ws*? $\notin$ *communityMembers* |
| *RemoveFrom_WSCommunity* | *ws*? : *Webservice*<br>*report*! : *REPORT* | *ws*? $\in$ *communityMembers* |
| *Query_ParentOfMA* | *ma*? : *Mobile_Agent*<br>*parent*! : *Agent*<br>*report*! : *REPORT* | *ma*? $\in$ ran *CreateMAChild* |

### 5.4.8 Error Conditions

While creating child agents, it is important to make sure the agents created are unique. Subsequently, schema *Wrong_MAChild_Creation* ensures the entities do not have the same identity. The same holds for the creation of stationary agents.

*_Wrong_MAChild_Creation*

$\Xi$ *Mobile_Agent*
*identity*? : *IDENTITY*
*report*! *: REPORT*

---

$\exists$ *a*: *Agent* • *a.agentID = identity*?
*report*! = *Mobile_agent_identity_already_exists*

*_Wrong_SAChild_Creation*

$\Xi$ *Stationary_Agent*
*identity*? : *IDENTITY*
*report*! *: REPORT*

---

$\exists$ *a*: *Agent* • *a.agentID = identity*?
*report*! = *Stationary_agent_identity_already_exists*

The node to which a mobile agent migrates must not be the same node that it is currently on. This is specified by the *Wrong_MigrateTo*.

*_Wrong_MigrateTo*

$\Xi$ *Mobile_Agent*
*newHome*? : *NODE*
*report*! : *REPORT*

---

*newHome*? = *head agentItinerary*
*report*! = *Agent_already_at_destination*

Adding a Web service that is already part of the Web service community should not be possible, hence schema *Wrong_WS_AddTo_WSCommunity* .

*_Wrong_WSAddTo_WSCommunity*

$\Xi$ *WSCommunity*
*ws*?: *Webservice*
*report*! : *REPORT*

---

*ws*? $\in$ *communityMembers*
*report*! = *Web_service_already_part_of_the_Community*

Note that set-theoretically it is immaterial to add an already existing element to a set (e.g. adding a web service to a community which it already belongs to), yet in software specification it is usually an indication of some conceptual error in the system.

Removing a Web service that is not part of a Web service community should also raise an error condition as specified in *Wrong_WS_Removed_From_WSCommunity*.

*Wrong_WS_Removed_From_WSCommunity*

$\Xi$ *WSCommunity*
*ws*?: *Webservice*
*report*! : *REPORT*

---

*ws*? $\notin$ *communityMembers*
*report*! = *Web_service_not_part_of_the_Community*

A similar situation to adding a web service already belonging to a community applies to removing a non-existent web service.

The addition of a Web service to a community that does not perform the same tasks as those of the community should not be allowed.

*WrongTask_WS_AddTo_WSCommunity*

$\Xi$ *WSCommunity*
*ws*?: *Webservice*
*report*! : *REPORT*

---

*ws*?.*tasks* $\neq$ *wsCommunityTask*
*report*! = *Webservice_does_not_perform_similar_tasks*

### 5.4.9 Total System Operations

Incorporating the partial operations listed in Table 1 with their error conditions presented in Section 5.4.8 leads to the total (robust) operations:

$$TotalCreateMAChild \triangleq CreateMAChildAgent \lor Wrong\_MAChild\_Creation$$
$$TotalCreateSAChild \triangleq CreateSAChildAgent \lor Wrong\_SAChild\_Creation$$
$$TotalMigrateTo \triangleq MigrateTo \lor Wrong\_MigrateTo$$
$$TotalAddToWSCommunity \triangleq AddTo\_WSCommunity \lor Wrong\_WSAddTo\_WSCommunity \lor WrongTask\_WS\_AddTo\_WSCommunity$$
$$TotalRemoveFrom\_WSCommunity \triangleq RemoveFrom\_WSCommunity \lor Wrong\_WS\_Removed\_From\_WSCommunity$$
$$TotalQuery\_ParentOfMA \triangleq Query\_ParentOfMA \lor Wrong\_Query\_ParentOfAgent$$

## 6.0 CONCLUSIONS AND FUTURE WORK

The PMForNR process model for formalising non-functional requirements of Service-based software systems proves to be feasible and is independent of any particular quality attribute. This was demonstrated by selecting a non-functional requirement, i.e. *availability* and applying the PMForNR process model to it without considering the application domain of the software system. This process model achieves its objective of being able to formalise quality attributes in a manner that is not confined to a particular quality attribute as it is based on a commonly used general-purpose modelling language (i.e. UML) and a general-purpose formal notation (i.e. **Z**). The proposed process model augments the strengths of UML by providing a mechanism to incorporate formalisms into its semi-formal notations. Besides providing an alternative to incorporating formalisms using ADLs, the lack of support for a wider range of non-functional requirements in ADLs is offset by the process presented in this work. The formalism of the systems architecture leads to specifications that are more precise and are therefore, more likely to lead to unambiguous statements during the implementation stages. The very first step in the PMForNR process model requires that the quality attribute be specified, though informally. This leads to an early incorporation of non-functional properties into the development process and subsequently to provide benefits such as less costly error corrections.

There are several dimensions in which this work may be expanded. For instance, mappings between UML and other formal methods such as Rodin may be investigated. Augmenting existing mechanisms for the automatic generation of UML models from architectures ought to receive attention as well as enhancing tool support for the automatic generation of formal specifications from UML models.

## REFERENCES


[1] R. Weinreich, & G. Buchgeher, "Towards supporting the software architecture life cycle", *Journal of Systems and Software,* Vol. 85, No. 3, 2012, pp. 546-561.

[2] H.P. Breivold, I. Crnkovic, & M. Larsson, "A systematic review of software architecture evolution research". *Information and Software Technology,* Vol. 54, No. 1, 2012, pp. 16-40.

[3] G. Buchgeher, & R. Weinreich, "An approach for combining model-based and scenario-based software architecture analysis", *Fifth International Conference on Software Engineering Advances (ICSEA)*, 2010, pp. 141-148.

[4] N. Medvidovic, & R. N. Taylor, "A classification and comparison framework for software architecture description languages", *IEEE Transactions on Software Engineering,* Vol. 26, No. 1, 2000*,* 26**,** pp. 70-93.

[5] R. Kassem, M. Briday, J. L. Béchennec, G. Savaton, & Y. Trinquet, "Harmless, a hardware architecture description language dedicated to real-time embedded system simulation", *Journal of Systems Architecture,* Vol. 58, No. 8, 2012, pp. 318-337.

[6] L. De Silva, & D. Balasubramaniam, "Controlling software architecture erosion: A survey", *Journal of Systems and Software,* Vol. 85**,** No. 1, 2012, pp. 132-151.

[7] A. Al Mamun, M. Tichy, & J. Hansson, "Towards formalizing assumptions on architectural level: A proof-of concept", *Research Report on Software Engineering and Management*, 2012.

[8] N. Medvidovic, R.N. Taylor, & E.J. Jr. Whitehead, "Formal modeling of software architectures at multiple levels of abstraction", *ejw,* Vol. 714, 1996, pp. 824-2776.

[9] J. Magee, & J. Kramer, "Dynamic structure in software architectures", *ACM SIGSOFT Software Engineering Notes,* Vol. 21, No. 6, 1996, pp. 3-14.

[10] S. Vestal, "MetaH Programmer's Manual", Version 1.27. Honeywell, Inc., Minneapolis, MN, 6, 1998.

[11] D.C. Luckham, "Rapide: A language and toolset for causal event modelling of distributed system architectures", *Worldwide Computing and Its Applications—WWCA'98.* Springer. 1998.

[12] R. Allen, & D. Garlan, "The Wright architectural specification language", *Rapport technique CMU-CS-96-TBD,* Carnegie Mellon University, School of Computer Science, 1996.

[13] R.G. Raj and S. Abdul-Kareem. "A Pattern Based Approach for The Derivation Of Base Forms Of Verbs From Participles And Tenses For Flexible NLP". *Malaysian Journal of Computer Science*, Vol. 24(2), pp 63-72, 2011.

[14] P. Zhang, H. Muccini, & B. Li, "A classification and comparison of model checking software architecture techniques", *Journal of Systems and Software,* Vol. 83, No. 5, 2010, pp. 723-744.

[15] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, & A. Tang, "What industry needs from architectural languages: A survey", *IEEE Transactions on Software Engineering,* Vol. 39**,** No. 6, 2013, pp. 869-891.

[16] R. K. Pandey, "Object constraint language (OCL): past, present and future", *SIGSOFT Softw. Eng. Notes,* Vol. 36, No. 1, 2011, pp. 1-4.

[17] E. Woods, & R. Hilliard, "Architecture Description Languages in Practice", Session Report. *The Working IEEE/IFIP Conference on Software Architecture (WICSA)*, 2005, pp. 243-246.

[18] T. Vajk, Z. David, M. Asztalos, G. Mezei, & T. Levendovszky, "Runtime model validation with parallel object constraint language", *Proceedings of the 8th International Workshop on Model-Driven Engineering, Verification and Validation,* New Zealand, ACM, 2011.

[19] M. Nouh, R. Ziarati, D. Mouheb, D. Alhadidi, M. Debbabi, L. Wang, & M. Pourzandi, "Aspect weaver: a model transformation approach for UML models", *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research,* Canada: IBM Corp. 2010.

[20] R.G. Raj and V. Balakrishnan, "A Model For Determining The Degree Of Contradictions In Information", *Malaysian Journal of Computer Science*, Vol. 24(3): September 2011. pp 160-16.

[21] Z. Joe, & J. P. Christopher, "Modeling Architectural Non Functional Requirements: From Use Case to Control Case", *IEEE International Conference on e-Business Engineering ICEBE '06*, Oct. 2006, pp. 315-322.

[22] A. Vemulapalli, & N. Subramanian, "Evaluating consistency between BPEL specifications and functional requirements of complex computing systems using the NFR approach", *4th Annual IEEE Systems Conference*, April 2010, pp. 153-158.

[23] L. B. Huang, V. Balakrishnan, R.G. Raj, "Improving the relevancy of document search using the multi-term adjacency keyword-order model." *Malaysian Journal of Computer Science*, Vol. 25, No. 1, pp. 1-10, 2012..

[24] Z. Xuede, "A Formal Testing Framework for UML Statecharts", *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007)*, July 2007, pp. 882-887.

[25] Y. Wei, & D. Yugen, "Research on Reverse Engineering from Formal Models to UML Models", *Third International Symposium on Parallel Architectures, Algorithms and Programming (PAAP)*, Dec 2010, pp. 406-411.

[26] J. Cabot, & R. Clarisó, "UML/OCL verification in practice", *ChaMDE 2008*, 31-35.

[27] R. Wille, M. Soeken, & R. Drechsler, "Debugging of inconsistent UML/OCL models", *Proceedings of the Conference on Design, Automation and Test in Europe.* Dresden, Germany, 2012, pp. 1078 - 1083.

[28] R. Pressman, *Software Engineering: A Practitioner's Approach*, McGraw-Hill, Inc. 2010

[29] T. Gouasmi, A. Regayeg, & A. H. Kacem, "Automatic Generation of an Operational CSP-Z Specification from an Abstract Temporal^Z Specification", *IEEE 36th Annual Computer Software and Applications Conference Workshops (COMPSACW)*, July 2012, pp. 248-253.

[30] H. B. Enderton, *Elements of set theory*, Academic Press Inc. 1997.

[31] J. P. Bowen, *Formal specification and documentation using Z: A case study approach*, International Thomson Computer Press. 1996.

[32] J. Woodcock, & J. Davies, *Using Z: specification, refinement, and proof*, Prentice Hall. 1996.

[33] M. Utting, P. Malik, & I. Toyn, "Transformation rules for Z", *Proceedings of the Fifteenth Australasian Symposium on Computing: The Australasian Theory,* Vol.94. Wellington, New Zealand: Australian Computer Society, Inc. 2009.

[34] G. K. Palshikar, "Applying formal specifications to real-world software development", *IEEE Software*, Vol.18, No. 6, 2001, pp. 89-97.

[35] M. Butler, & S. Hallerstede, "The Rodin formal modelling tool", *BCS-FACS Christmas 2007 Meeting-Formal Methods In Industry*, London, 2007.

[36] M. Shaw, R. Deline, D. V. Klein, T. L. Ross, D. M. Young, & G. Zelesnik, "Abstractions for software architecture and tools to support them", *IEEE Transactions on Software Engineering,* Vol. 21, No. 4, 1995, pp. 314-335.

[37] C. V. Eenoo, O. Hylooz, & K.M. Khan, "Addressing non-functional properties in software architecture using adl", *The Sixth Australasian Workshop on Software and System Architectures (AWSA 2005)*, Brisbane, Australia, Mar 2005, pp 6-12.

[38] X. Franch, & P. Botella, "Putting non-functional requirements into software architecture", *Proceedings of the Ninth International Workshop on Software Specification and Design*, Apr 1998, pp. 60-67.

[39] N. S. Rosa, P. R. Cunha, & G. R. Justo, "Process NFL: A language for describing non-functional properties", *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS.2002)*, 2002, pp. 3676-3685.

[40] G. Ying, K.G. Aditya, & L. Zheng, "HCLP Based Service Composition", *IEEE/WIC/ACM International Conference Web Intelligence and Intelligent Agent Technology Workshops, WI-IAT 2006 Workshops*, Dec. 2006, pp. 138-141.

[41] X. Wang, T. Vitvar, M. Kerrigan, & I. Toma, "A qos-aware selection model for semantic web services", *Service-Oriented Computing (ICSOC 2006)*, 2006, pp. 390-401.

[42] A. S. Ran, "A model for web services discovery with QoS", *SIGecom Exch,* Vol. 4, No. 1, 2003, 1-10.

[43] B. Potter, D. Till, & J. Sinclair, *An introduction to formal specification and Z*, Prentice Hall PTR. 1996.

[44] W. Scacchi, "Process models in software engineering", *Encyclopedia of software engineering*. John Wiley and sons, New york, 2001.

[45] M. Shroff, & R. B. France, "Towards a formalization of UML class structures in Z", *Proceedings of the 21st Annual International Computer Software and Applications Conference, COMPSAC '97*, Aug 1997, pp. 646-651.

[46] Z. Maamar, Q. Z., Sheng, & D. Benslimane, "Sustaining web services high availability using communities", *Third International Conference on Availability, Reliability and Security*, 2008, 834-841.

[47] L. Chung, & J. Do Prado Leite, "On non-functional requirements in software engineering", *Conceptual modeling: Foundations and applications*, Springer-Verlag, Berlin, 2009, pp. 363-379.

[48] S. Subramanian, "Highly-Available Web Service Community", *Sixth International Conference on Information Technology: New Generations, 2009. ITNG '09*, April 2009, pp. 296-301.

[49] D. Hanfei, H. Qinfen, Z. Tiegang, & Z. Bing, "Formal Discussion on Relationship between Virtualization and Cloud Computing", *International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, Dec 2010, pp. 448-453.

[50] S. Haag, & M. Cummings, *Management information systems for the Information Age.* McGraw-Hill Irwin, 2013.

[51] M. Lall, J. A. Van Der Poll, & L. M.Venter, "An Agent-based framework for enhancing the availability of Web services", The International Conference on Computing, Networking and Digital Technologies (ICCNDT 2012), Gulf University, Bahrain, Nov 2012, pp. 154 - 165.

[52] J. A. Van Der Poll, & P. Kotze, "Enhancing the established strategy for constructing a Z specification: reviewed article", *South African Computer Journal***,** No, 35, 2005, pp. 118-131.