How to cite this article:

Aradea, A., Supriana, I., & Surendro, K. (2020). Self-adaptive model based on goal-oriented requirements engineering for handling service variability. *Journal of Information and Communication Technology, 19*(2), 225-250. https://doi.org/10.32890/jict2020.19.2.4

SELF-ADAPTIVE MODEL BASED ON GOAL-ORIENTED REQUIREMENTS ENGINEERING FOR HANDLING SERVICE VARIABILITY

Aradea Aradea, Iping Supriana & Kridanto Surendro
School of Electrical Engineering and Informatics,
Bandung Institute of Technology, Indonesia

aradea@unsil.ac.id; iping,endro@informatika.org

ABSTRACT

Service system is currently facing environmental complexity problems, such as the need of a distributed, heterogeneous, decentralized, and interdependent system which operates dynamically and unpredictably. This condition requires the service system to have an ability to adapt in order to realize sustainable functions. The success of service adaptation is determined by its ability to handle variability at runtime. The purpose of this research is to realize service flexibility through variability modeling, which is an extension of previous work to enrich the adaptability view. The methodology was developed through the monitor-analyse-plan-execute-knowledge control loops approach integrated into the adaptive service (service level) element within the adaptive enterprise service system metamodel based on goal-oriented requirements engineering. Service adaptation scenario was prepared through proactive and reactive adaptation mechanisms. For evaluation, the model was applied to the case of a configuration management system. The experimental results showed that the model is able to adapt to runtime variability and accomodates the growth of the service component items shown by the description of the system scalability. The proposed model has a better alternative design in analyzing variability with a

Received: 31/10/2018 **Revised:** 2/5/2019 **Accepted:** 8/5/2019 **Published:** 31/3/2020

total response that can be applicable in normal operations and overload. It also meets the expected level (level-5: adapting) of the adaptive capability maturity model as a standard for assessment of a service system adaptation**.**

Keywords: Self-adaptive systems, service variability, goal-based, MAPE-K, rule-based systems.

INTRODUCTION

The service system has now become an important part in various activities, where different elements of the real-world system can interact with the system. The involvement of various elements and activities raises the issue of complexity in its development, for example the characteristics of system entities related to rapid organization growth, hardware ubiquitous, the dynamic and unpredictable environment, etc. These conditions require the service system to have an ability to adapt to environmental characteristics and uncertainty at runtime. The main factor behind the uncertainty is the variability at runtime that refers to changes which occur in the system requirements, environment, related systems, and the system itself (Abbas & Anderson, 2017). In our previous work (Surendro, Aradea, & Supriana, 2016), we introduced a requirements engineering for cloud computing adaptive (RECCA) model focused on cloud services variability. In this work, we propose three views, namely architectural view, alignment view, and adaptability view from which the three views of the requirements engineering process capture the service system requirements. However, the adaptability view only focuses on providing external services, which are cloud services. Meanwhile, the need for a service system within an enterprise of reality will also require a service provided by an internal party. This research extends the capability of the adaptability view, where the service flexibility factor becomes the main focus so that the system is able to adapt to the services requirements provided by both external and internal parties. Providing these services will address the runtime variability and growth of service items.

Based on a review of related works, there are still some missing pieces that has motivated us to conduct this research. For example, the results of work by Qureshi, Jureta, and Perini (2012), Clark, Warnier, and Brazier (2011), Morandini, Penserini, and Marchetto (2017), and Mendoca, Rodrigues, Alves, Ali, and Baresi (2016) indicate the need to further investigate the dynamic evolutionary needs of the requirements model for service variability. Meanwhile, Abuseta and Swesi (2015), Arciani, Riccobene, and Scandura (2015), Knauss, Damian, Franch, Rook, Muller, and Thomo (2016), Paz and Arboleda (2016) have utilized the advantages of autonomic computing to develop adaptation mechanisms. However, the representation of domain models (goal models) in the concept has yet to be investigated. Based on these facts, we see an opportunity to deal with limitations in the adaptive enterprise service system (AESS) metamodel proposed by Surendro et al. (2016). A more detailed discussion of this gap is presented in the related works section.

This paper introduces the handling of service variability where the lifecycle of the adaptive service element in the AESS metamodel is formulated as a monitor-analyze-plan-execute-knowledge (MAPE-K) pattern through goal-oriented requirements engineering (GORE). Adaptation mechanisms are developed through two strategies. The first is proactive adaptation prepared through a set of variability rules to anticipate changes in the service context. The second is the reactive adaptation prepared through a set of evolution rules to follow up on the needs for additions or changes to the new function service system at runtime.

RELATED WORKS

There have been some work on the concept of self-adaptive service. For example, Perini (2012) discussed various challenges related to requirements from the engineering perspective for self-adaptive service based applications, in which a challenge viewpoint is defined for design-time and run-time requirements. The proposed model in this paper may be regarded as one of the answers to the challenge. Qureshi and Perini (2010) proposed a framework for continuous adaptive requirements engineering (CARE) supporting self-adaptive service-based applications using Techne's language to map the goal model into ontology domains. This concept can help in detailing the behavior of the system to meet its goals and adaptation actions. However, the mechanism of reasoning for changes in domain assumptions, preferences and contexts still requires further research. Meanwhile, our research proposes a dynamic rule model so that reasoning at run-time can be done automatically. Clark et al. (2011) introduced self-adaptive monitoring services to adapt to changes based on risk levels. This model focused on service monitoring capabilities to respond to change. Our proposed model is not only prepared for handling changes, but accomodates system evolution requirements as the growth of service items also becomes one of the actions of the monitoring results.

Anna et al. (2019), proposed a model of requirements engineering for adaptive systems based on goal model, and Mendoca et al. (2016) proposed a model of contextualed runtime goal through a probabilistic approach. However, the dynamic evolution requirements are still not covered in these works, while our model provides this capability through the plug and play model. The expansion of autonomic computing (Abeywickrama & Ovaska, 2017) has now become a major concern of researchers in developing self-adaptive models. Arciani et al. (2015) introduced a framework for modeling and validating distributed self-adaptive service-oriented applications using the formal method. Further, Knauss et al. (2016) introduced a model of contextual requirements using machine learning and data mining approach. Paz and Arboleda (2016) also proposed a model for the guide of dynamic adaptation planning with formal methods. The works focused on the generic function of

MAPE-K control loops for reasoning at runtime. However, in our model, the representation of entities is from the domain problem as a domain model (goal model) under additional concerns. So, it has advantages in terms of capturing the requirements domain. Meanwhile, Surendro et al. (2016) adopted the AESS metamodel to handle service variability in service catalogs limited to providing external services.

Based on these related works, we argue that handling service variability can be improved through the ability to realize the dynamic evolution of service requirements based on adaptation patterns embedded in the service level elements of the AESS metamodel. Service requirements are defined through GORE to represent domain models. Meanwhile, adaptation strategies are realized through a generic function of the MAPE-K pattern. The proposed method section discusses in more detail the approach used in this paper.

RESEARCH METHODOLOGY

This research is divided into five phases as presented in Table 1. Phase 1 reviews some related research to identify gaps and define research problems. In Phase 2, the research problem is identified, that is how to handle service variability in the AESS metamodel.

Table 1

Research design

Research Design		
Phase	Activity	Description
1 Systematic Literature Review	a. Explore research areas and related work b. Survey engineering approaches	Understand the domain(s) of research, related work, and existing engineering approaches
2 Research Problem	a. Identify research gaps b. Define research problems	Define research limitations and opportunities for improvement
3 Requirements Modeling	a. Define elements of model requirements b. Map model elements	Define the requirements of the model and its controls
4 Proposed Method	a. Develop an approach b. Formulate algorithm	Develop an approach through the integration of goal models and MAPE-K in the AESS metamodel
5 Empirical Evaluation	a. Apply case studies b. Compare proposed models with related work	Evaluate proposed method through the quality attributes of dQAS and ACMM

Phase 3 defines the elements of the model needs by mapping the AESS metamodel into the requirements model and its control needs to obtain three views of the model, namely architecture, alignment, and adaptability. Furthermore, the third phase results are used in Phase 4 to realize the improvement of the adaptability view by introducing approaches at each level of the AESS metamodel through the integration of goal-oriented requirements engineering and the MAPE-K adaptation cycle. Finally, in Phase 5, an evaluation is carried out to prove that the proposed model provides relevant contributions. Empirical evaluation is conducted through a discussion of case studies using the domain Quality Attribute Scenarios (dQAS) and Adaptive Capability Maturity Model (ACMM).

PROPOSED METHOD

In order to realize the self-adaptation capabilities for service variability, we utilized the methods of some previous researches, for example, Abuseta and Swesi (2015) who proposed a design pattern for a model of MAPE-K. Some of the patterns are used in the model which we proposed and also expanded by adding the ability to plug and play as a form of service system evolution. Nakagawa, Ohsuga, and Honiden (2012) research also inspired the proposed model, in which our models can enrich its features. Further, Morandini (2017) proposed Tropos4AS where the primitive of goal model is used as requirements description, adopted and expanded in our model. This work has advantages in terms of capturing context variability and we equip it with domain assumption through the rule editor and embed the control loops approach.

The configuration developed in the proposed model is an extension of our previous work (Aradea, Supriana, Surendro, & Darmawan, 2017a; 2017b) to enrich the adaptability view of the RECCA model. We introduced three requirement views: architecture, alignment, and adaptability in the model. The architectural view is enabled to understand the environment. Then, the results of that understanding are mapped into the service system requirements through an alignment view. Finally, the adaptability view determines the adaptation mechanism. The adaptability view is realized through an event-condition-action (ECA) method that represents the MAPE-K concept, but does not explicitly define the mapping mechanism of the goal model as a description of requirements. In addition, the function of the adaptability view is only focused on the cloud service variability. The construction of the proposed model is to complement the adaptability view by preparing a more relaxed adaptation mechanism for service variability.

In the AESS metamodel, the design principle consists of agility, a living system (system of systems) and service principles, where the core elements are divided into three levels: adaptive enterprise service system (enterprise level), adaptive service system (capability level) and adaptive service (service level) (Gill, 2015). Enterprise level is a conceptual element of the adaptive enterprise architecture metamodel, while capability level is defined as a system abstraction that can represent a human individual, function, business unit, department or team, etc. In the RECCA model, both levels are configured through an architectural view to capture environmental elements and alignment view to define their service requirements. In this paper, we focus on the adaptability view for service level expansion that is context aware and is continually evolving and self-adapting.

Figure 1 illustrates an overview of the proposed self-adaptive model which is an extension of the RECCA model based on the principles of the AESS metamodel. Enterprise and capability levels are defined as the domain model by adopting goal-oriented requirements engineering (goal model) and service level is realized as control strategy through the implementation of the MAPE-K adaptation cycle, which consists of scans and senses (monitors [M]), interprets and analyses (analyze and plan [AP]), decides and responds (execute [E]) to internal and external changes.

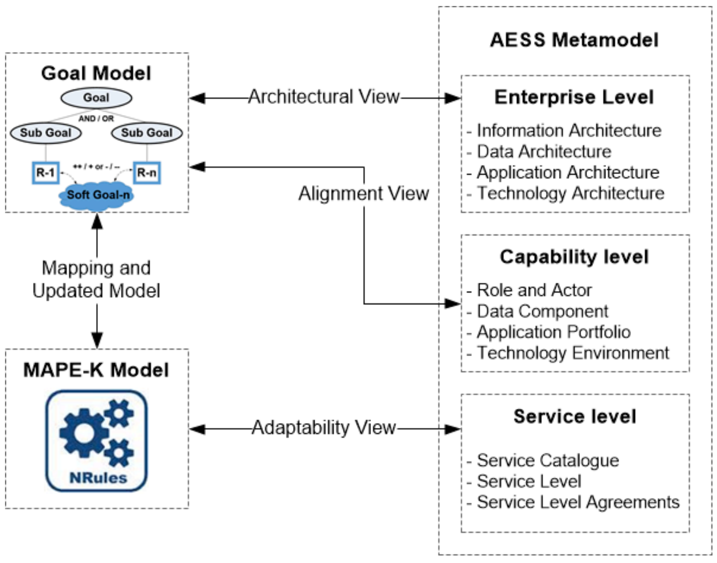


Figure 1. An overview of the self-adaptive model.

The decomposition of goal model (functional) can represent the service requirements (R) in every sub-goal that are influenced by each property and have positive or negative (++ / + or - / -) contribution to soft-goal (non-functional). The role of the control strategy (MAPE-K) in this case is to identify and monitor the possibilities of changes in the service. Decomposition model adopts the concept of component mapping (Nakagawa et al., 2012) for software components (Hirsch, Kramer, Magee, & Uchitel, 2006) by utilizing some design patterns (Abuseta & Swesi, 2015) and modifies in accordance with the requirements of service systems. Figure 2 illustrates a model for transforming model goals into software components. Each parent goal with AND-Decomposition is defined as a goal to analyze and plan (AP), while every child's goal is defined as a goal to monitor (M) and execute (E), which is fully regulated in the knowledge (K).

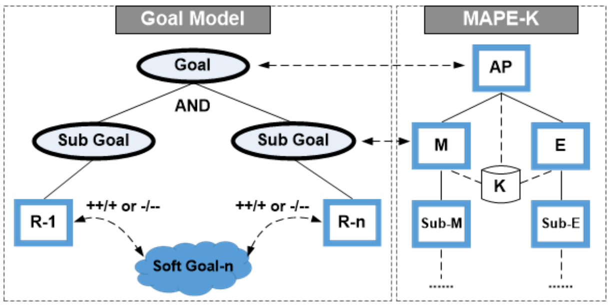


Figure 2. Goal Mapping to MAPE-K Components.

The control strategy to adjust each component is started by the M (monitor) component function as shown in Algorithm 1. There are a number of properties (P) on the goal (G) model (m) that should be monitored concurrently. This activity represents the runtime states which are time-triggered or event-triggered to respond to requests or events. State (S) of system at runtime is represented by a combination of internal and external property values. Violation of the state is detected by way of any violation of the threshold of each goal property and the new state will be stored in the system state log to be analyzed.

Algorithm 1

Monitoring algorithm

Monitoring States

```
for all G in m do
    m ⇽ ($\sum : f_n$) //run-time
    G ⇽ getValue(P) //time_or_event_triggered
    for each value(S) in P do
        S ⇽ combining internal and external value(S)
        if S in S.target(F) $\neq$ P.threshold then
          systemState ⇽ new S.system(S.instance) and
          systemStateLog ⇽ save(S.system) and
          send information(S.system) to analyzerManager
        end if
    end for
end for
```

Algorithm 2

Analyze-Plan algorithm

Analyze-Plan

```
δ⟸(S.system)// state of system
 for each δ in analyzerManager do
     analyzer ⟸update(logs) actual S.system
     search (δ) in symptomList
     if symptom ≠ ø then
        create(adaptationRequest) and
        update(adaptationRequest) for plan specification
        else
             addSymptom to symptomList and
             create(adaptationRequest) and
             send information(adaptationRequest) for plan specification
        end if
end for
```

Algorithm 3

Execute algorithm

Execute of Plan

```
for all δ is found do
      a ⟸ construct correctiveAction(addAction)
      changePlan ⟸ newChangePlan($a_n$)
      send changePlan to one or more executors
      for each a in executor do
           actuator ⟸ update($a_n$) //one or more actuators
           S.system ⟸ reconfiguration m with actuator
           // set new value for C(G.Node)
           systemStateLog ⟸ saveState(S.system)
      end for
end for
```

Violation of the goal system is analyzed based on the symptoms list. If the results of the analysis detected the presence of some symptoms, the system will accept the adaptation request and then reconfigure based on rule engine. Algorithm 2 shows the reconfiguration algorithms for AP (analyze and plan) component. Rule engine contains high-level goals that control the operation and functions of related systems. The general form is event-condition-action (ECA) rules. In our version, the rule engine is extended with a rule editor model where the specification changes can be done by editing the knowledge base directly or putting back into the system. Each adaptation request is represented as a state of system (S). A set of S contains the context (goal model) and the expected action for the target system. Changing plan contains the adaptation actions to be executed by the E (execute) component. The execute component (Algorithm 3) will use a number of actuators for setting the new value of the target system property. Adaptation strategies are developed through two rules of adaptation, namely a set rule of variability for proactive adaptation and a set rule of evolution for reactive adaptation. Proactive adaptation is prepared to anticipate changes in context information identified based on symptoms or events arising. The type of adaptation is formulated through ECA rules as follows:

WHEN <event> ; current situation when there is a change in service
IF <condition> ; certain events that occur so that the appropriate action is activated
THEN <action> ; adjustments to service changes for reactive adaptation behavior
VALID-TIME <time_period> ; suitability for service adaptation

Reactive adaptation is prepared to follow up on the need for service updates based on the results of operations from proactive adaptation. This type of adaptation utilizes the scheme of service levels in the AESS metamodel, where each service instance in the service catalog is generated based on the results of the MAPE-K pattern analysis, so that service requirements can be activated according to prevailing conditions. The rule specifications for both types of proactive and reactive adaptation can be defined through the rule editor according to the preferences and requirements of stakeholders.

EVALUATION AND RESULTS

The discussion presented in this section is an extension of the configuration management system case based on the ITIL Framework (OGC, 2007) which is now widely used by large companies in the world for the provision of IT services. The main target of this experiment is scalability, which accommodates users' requirements in accessing an application service based on the changes, assesses the characteristics of quality attributes in handling variability at runtime, and measures the adaptive capability maturity level. Goal modeling (GORE) is shown in Figure 3 while the mapping of the system components is shown in Figure 4.

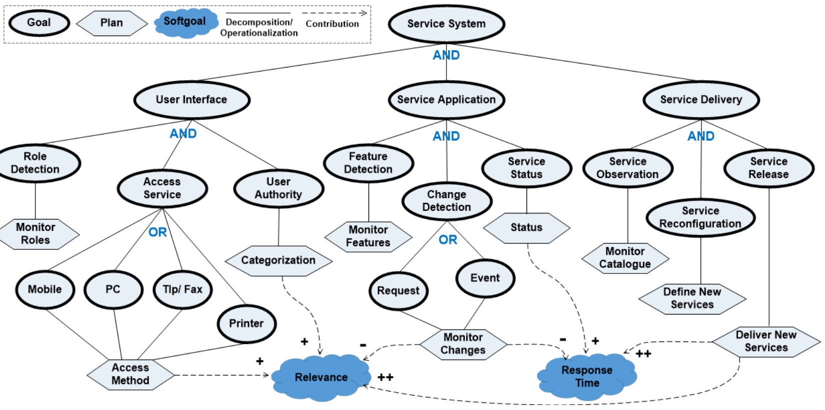


Figure 3. Goal modeling for service system.

The results of goal modeling in Figure 3 are representations of the architectural and alignment views based on the AESS Metamodel. Meanwhile, Figure 4 is a representation of the adaptability view based on the mapping from Figure 3. The mapping rule refers to Figure 2, that is each AND-Decomposition in goal modeling is generated into a composite component, so that three composite components are obtained, namely, user interface, service application, and service delivery. Each composite component represents the adaptation cycle through three types of primitive components, namely, M (monitor), AP (analyze and plan), and E (execute). The links of each composite and primitive components are defined by two types of ports, namely provider service ports and required service ports based on the links formed from the results of goal modeling. The mapping in Figure 4 generates an adaptability pattern for user requirements and service requirements represented by the service delivery function in the service application.

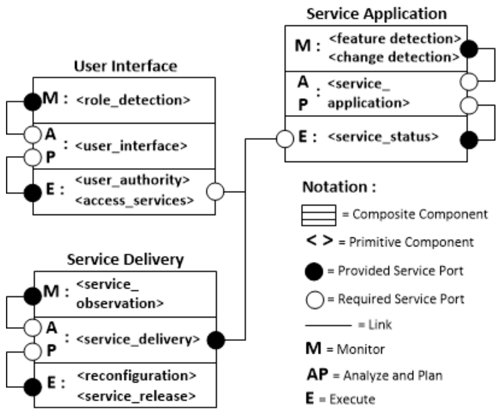

Service Application
M : <feature detection> <change detection>
A P : <service_ application>
E : <service_status>
User Interface
M : <role_detection>
A P : <user_interface>
E : <user_authority> <access_services>
Service Delivery
M : <service_ observation>
A P : <service_delivery>
E : <reconfiguration> <service_release>
Notation :
= Composite Component
< > = Primitive Component
● = Provided Service Port
○ = Required Service Port
= Link
M = Monitor
AP = Analyze and Plan
E = Execute

Figure 4. Component specification for service system.

Experiment

The property on context element monitored is service item or known as configuration item (CI) which consists of hardware, software, peripheral and network equipment. Handling these changes can be classified into two types of adaptation, which are: proactive adaptation and reactive adaptation. Proactive adaptation is to determine which components need to be updated, added and/ or deleted. It can be assigned as a symptom or event that can be identified. The setting of all these events can be assigned as rules, for example:

- access device events, when a new device is detected or unavailable device;
- authority events, when the permission mismatch is detected by the user or user's role changes;
- feature and service time events, when unavailability of features and / or time of service beyond the threshold is detected in the service catalog.

For example, it will discuss the rule when access event of device appears. Based on the function of the component "role detection (M)" and "user authority (E)", there are a number of "access services (E)" which will be accessed by users via the interface options. Goal decomposition of this services access is OR-decomposition, thus showing variability related to the resource (device_type). In addition, based on the function components, it is possible to change "feature detection (M)" and "change detection (M)" in service due to unexpected events or errors (event_error). Based on the description, the plan can be represented as: plan (device_type, event_error). This plan is to create an alternative behavior in dealing with variability context; for example, the plan of "access method" and "determination of the status" must use the function of "service delivery (AP)" components, because it provides the full positive contribution (++) toward the "relevance" and "response time" soft-goal, as opposed to doing analyze and plan (AP); each of which only contributes to positive (+) that will affect, even negative contribution (-). Thus, the system has a consideration to analyze and plan (AP) for a "user interface" and "service application". Collection of this property value would set the system input variables "service delivery (AP)". The setting of behavior can be defined as follows:

- **Rule-1** : *if (device_type = mobile) and (event_error = null) the plan = internet_service delivery*
- **Rule-2** : *if (device_type = personal_computer) and (event_error = null) then plan = window_client_service or virtual_terminal_service delivery*
- **Rule-3** *: if (device_type = telephone or fax) and (event_error = null) then plan = text_messaging_service delivery*

- **Rule-4** : *if (device_type = printer) and (event_error = null) then plan = printouts_service delivery*
- **Rule-5** : *if (device_type = new_type) and (event_error = null) then plan = add new device_type [instance component]*
- **Rule-6** : *if (device_type = null) and (event_error = null) then plan = change_service delivery [instance component]*
- **Rule-7** : *if (event_error = not null) then plan = send notification to user and service_desk*
- **Rule-8** : *if not [criteria] then plan = change_service delivery [instance component]*

These rules can be mapped into the concept of ECA rules, as shown in Table 2. So, there are four action plans (P_n) as the alternative solution. Meanwhile, reactive adaptation can be done by determining the procedures for handling service disruptions. For example, the handling of service disruptions will identify single points of failure, as can be seen in Figure 5, in order to obtain the configuration item (CI) as shown in Table 3. The data is obtained based on the monitor (M) functions, "role detection" and "feature detection". Then, the component of "service observation (M)" will perform detection to determine components of CI that can be considered critically. Availability-service (*As*) in Table 6 consists of several CI components with different levels of availability-component (*Ac*).

Table 2

ECA: Access services

Access Services		
Event (E)	**Condition (C)**	**Action (A)**

access_device	*(device_type = mobile);*
	(device_type = personal_computer);
	(device_type = telephone or fax);
	(device_type = printer);
	(event_error = null);
		$P_{1.1}$ = internet_service
		$P_{1.2}$ = window_client or virtual_terminal
		$P_{1.3}$ = text_messaging
		$P_{1.4}$ = printouts

access_device	*(device_type = new_type);*
	(event_error = null);
		P_2 = add new device_type [instance component]

access_device	*(device_type = null);*
	not [criteria]; (event_error = null);
		P_3 = change_service delivery [instance component]

access_device	*(event_error = not null);*	*P_4 = send notification to user and service_desk*

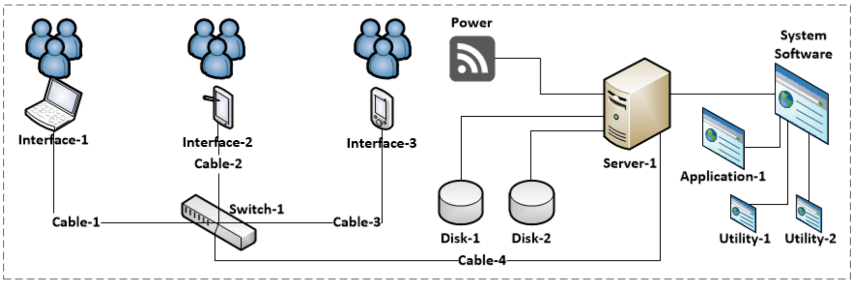


Figure 5. Single point of failure.

Table 3

Description of configuration item

Configuration Item (CI)	Option				Service (S) Dependency	User	Availability
	S_1	S_2	S_3	S_4	$S_{(1-4)}$	$S_{(1-4)}$	(%)
CI_1 : Server-1	1	1	2	1	Shared	1400	90.00%
CI_2 : Disk-1	1	1	-	1	Shared	1150	90.00%
CI_3 : Disk-2	1	1	2	-	Shared	1200	75.00%
CI_4 : System-1	1	1	2	1	Shared	1400	90.00%
CI_5 : Utility-1	1	-	-	-	L_1	550	73.00%
CI_6 : Utility-2	-	1	-	-	L_2	400	75.00%
CI_7 : Application-1	1	1	2	1	Shared	1400	90.00%
CI_8 : Interface-1	1	-	-	-	L_1	550	85.00%
CI_9 : Interface-2	-	1	-	-	L_2	400	85.00%
CI_{10} : Interface-3	-	-	2	1	Shared	550	85.00%
CI_{11} : Cable-1	1	-	-	-	L_1	550	90.00%
CI_{12} : Cable-2	-	1	-	-	L_2	400	90.00%
CI_{13} : Cable-3	-	-	2	1	Shared	450	90.00%
CI_{14} : Cable-4	1	1	2	1	Shared	1400	90.00%
CI_{15} : Switch-1	1	1	2	1	Shared	1400	90.00%
CI_{16} : Power-1	1	1	2	1	Shared	1400	90.00%

The calculation of service availability of stand-alone and redundant are formulated differently (OGC, 2007). The availability of services with a number of stand-alone CI is calculated by the following equation $As = Ac_1 * Ac_2 * Ac_3 ... Ac_n$. Thus, based on statistical data of MTBF (mean time between failures) and MTRS (mean time to restore service), the service availability of single point of failure in the availability column in Table 3 has a total value of 8.79%. The availability of each CI_n is obtained by the Equations 1, 2, 3

$$MTBF = \frac{Availability\ (time) - Downtime\ (time)}{The\ Amount\ of\ Interruption} \tag{1}$$

where *MTBF* denotes the average time that a configuration item (CI) or IT service can perform its agreed Function without interruption. This is measured from when the CI or IT service starts working, until it next fails (Lloyd & Rudd, 2011).

$$MTRS = \frac{Downtime\ (time)}{The\ Amount\ of\ Interruption} \tag{2}$$

where *MTRS* denotes the average time taken to restore a configuration item (CI) or IT service after a Failure. MTRS is measured from when the CI or IT service fails until it is fully restored and delivering its normal functionality (Lloyd & Rudd, 2011).

$$Availability = \frac{MTBF}{(MTRS + MTBF)} * 100\% \tag{3}$$

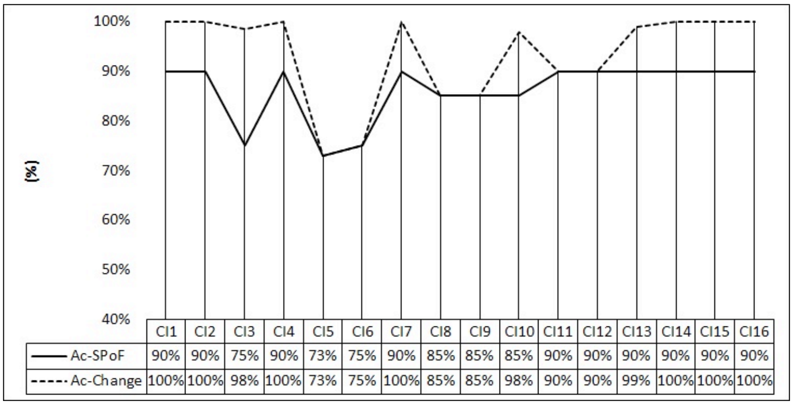
where *Availability* is the ability of a service, component or configuration item (CI) to perform its agreed function when required (Lloyd & Rudd, 2011).

Meanwhile, the calculation of the service availability with a number of redundant CI is conducted by the following equation: $As = Ac_1 + ((1 - Ac_1) * Ac_2)$ for one CI and one redundant CI, $As = Ac\ (n - 1) + ((1 - Ac_n\ (n - 1)) * Ac_n)$ for a number of (n) redundant CI. The service availability of redundant CI with shared dependencies can be seen in Table 7 with the number of redundant (n) which varies between 2 and 4. So, the total value of the redundant CI is 30.47%. These data are used as input variables for the "service delivery (AP)" component in determining any CI that can be considered as critical. Thus, the list of critical CI status is obtained as shown in Table 4; there are 10 critical CI with shared dependency requiring reconfiguration actions. The illustration of SLA percentage (%) change for each CI can be seen in Figure 6 with a total availability of services increase by 21.68% that is from 8.79% to 30.47%.

Table 4

Availability of redundant services

CI	Service (S)					
	Redundant				User	Availability
	L_1	L_2	L_3	L_4	$L_{(1-4)}$	(n = 2; 3; 4;)
CI_1	90.00%	90.00%	90.00%	90.00%	1400	99.99%
CI_2	90.00%	90.00%	-	90.00%	1150	99.90%
CI_3	75.00%	75.00%	75.00%	-	1200	98.44%
CI_4	90.00%	90.00%	90.00%	90.00%	1400	99.99%
CI_5	73.00%	-	-	-	550	73.00%
CI_6	-	75.00%	-	-	400	75.00%
CI_7	90.00%	90.00%	90.00%	90.00%	1400	99.99%
CI_8	85.00%	-	-	-	550	85.00%
CI_9	-	85.00%	-	-	400	85.00%
CI_{10}	-	-	85.00%	85.00%	550	97.75%
CI_{11}	90.00%	-	-	-	550	90.00%
CI_{12}	-	90.00%	-	-	400	90.00%
CI_{13}	-	-	90.00%	90.00%	450	99.00%
CI_{14}	90.00%	90.00%	90.00%	90.00%	1400	99.99%
CI_{15}	90.00%	90.00%	90.00%	90.00%	1400	99.99%
CI_{16}	90.00%	90.00%	90.00%	90.00%	1400	99.99%



	CI1	CI2	CI3	CI4	CI5	CI6	CI7	CI8	CI9	CI10	CI11	CI12	CI13	CI14	CI15	CI16
—— Ac-SPoF	90%	90%	75%	90%	73%	75%	90%	85%	85%	85%	90%	90%	90%	90%	90%	90%
----- Ac-Change	100%	100%	98%	100%	73%	75%	100%	85%	85%	98%	90%	90%	99%	100%	100%	100%

Figure 6. Critical CI

Based on data of critical CI, the system will then do a reconfiguration through the components of the "service reconfiguration (E)". Finally, the "service release (E)" component will deliver new services. For example:

- CI_1 is detected as critical CI, where "Server-1" is based on monitoring CPU usage necessary to improve and to avoid over utilization and contention. Thus, the system will add adaptation functions using a load balancing system.
- CI_7 is detected as critical CI, where "Application-1" is based on the monitoring of facilities provided which require the model to establish baseline performance through the addition of new features. Thus, the system will add new features through the cloud service so that the system will determine the cloud adoption mechanism.
- Treatment of any other critical CI is adjusted based on the event detected respectively.

The dashed line in Figure 7 shows the new added functions of the components. The following are previous works discussing the adaptation process of load balancing functions (Abuseta & Swesi, 2015; Darmawan & Aradea, 2017); the illustration of mapping the goal model into system components for load balancing is shown in Figure 8. The numbers of properties which should be monitored are shown in Tables 8 and 9. System will have the consideration to analyze and plan (AP) to organize "user's access". In addition, the system will also analyze and plan (AP) to set "performance of server farm". Based on the combination of each property value in Tables 8 and 9 will be the input variables for the system to "manage load (AP)" through "workload observation (M)" component. The combination of the value is performed by Equation 4.

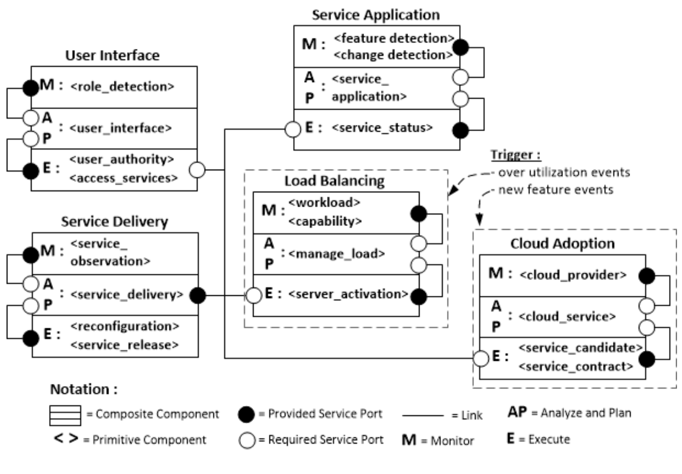


Figure 7. Component specification after the addition of new functions.

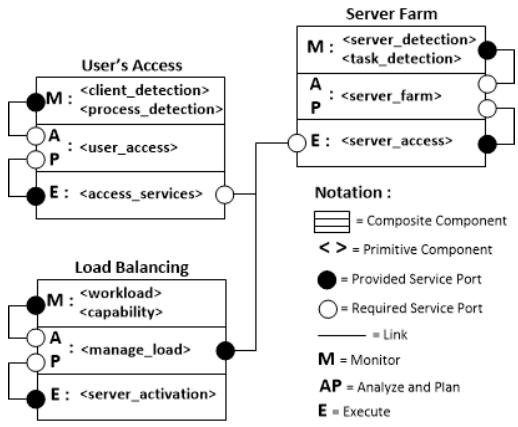


Figure 8. Component specification for load balancing system.

$$f(x) = \sum_{x=1}^{n} (c_n) \tag{4}$$

where x = server (task); n = number of clients; c = client.

in order to obtain the total task to be executed, that is 3209 tasks. The next stage "capability observation (M)" component will determine the ability of each server to the total task to be processed, through Equation 5.

$$f(x) = \frac{(\sum_{x=1}^{n}(c_n) + d_x)}{m_x \; x \; v_x} \tag{5}$$

where x = server (task); n = number of clients; c = client; d = distance; m = memory; v = speed.

Then, the server capability that can perform the task is sorted as quickly as possible based on the total task. Constraints to any desired process is set as k = 2 ms. The setting of system behavior to manage the server load is associated with some rules in response to symptoms or events, for example, high load event à when the server load is detected to be larger than 80%; unresponsive or very low load event à when detected, server does not perform the process.

Table 5

Client property

Client (n)	Load	Client (n)	Load	Client (n)	Load	Client (n)	Load
C_1	44	C_{14}	65	C_{27}	77	C_{40}	45
C_2	23	C_{15}	77	C_{28}	54	C_{41}	52
C_3	75	C_{16}	56	C_{29}	77	C_{42}	21
C_4	20	C_{17}	34	C_{30}	80	C_{43}	76
C_5	56	C_{18}	67	C_{31}	99	C_{44}	79
C_6	45	C_{19}	43	C_{32}	90	C_{45}	27
C_7	67	C_{20}	65	C_{33}	87	C_{46}	33
C_8	99	C_{21}	99	C_{34}	88	C_{47}	45
C_9	34	C_{22}	43	C_{35}	76	C_{48}	61
C_{10}	99	C_{23}	65	C_{36}	82	C_{49}	77
C_{11}	66	C_{24}	75	C_{37}	77	C_{50}	55
C_{12}	67	C_{25}	60	C_{38}	76		
C_{13}	43	C_{26}	89	C_{39}	99		

Table 6

Server farm property

Server (x)	Speed (v) = ms	Memory (m) = ms	Distance (d) = ms
S_1	10	10	34
S_2	12	7	2
S_3	11	12	10
S_4	7	8	4
S_5	10	9	3
S_6	6	6	2
S_7	8	5	20
S_8	10	8	2
S_9	8	4	1
S_{10}	6	8	2
S_{11}	11	4	5

(continued)

Server (x)	Speed (v) = ms	Memory (m) = ms	Distance (d) = ms
S_{12}	14	7	7
S_{13}	8	5	3
S_{14}	15	6	4
S_{15}	8	5	7
S_{16}	7	8	10
S_{17}	11	10	8
S_{18}	12	9	5
S_{19}	8	4	9
S_{20}	9	7	12

Table 7

Description of server load balancing

Number of Server	Detection of Server	Sorting of Server		Without Balancing		Balancing Server	Amount of Server
	f(x) second	f(x) second	Fitness	Server	Residue	Balancing	
1	32.43	24.39	1219%	80%	1139%	76%	1
2	38.23	29.25	1462%	80%	1059%	76%	1
3	24.39	29.76	1488%	80%	979%	76%	1
4	57.38	32.43	1622%	80%	899%	76%	1
5	35.69	32.82	1641%	80%	819%	76%	1
6	89.19	35.69	1784%	80%	739%	76%	1
7	80.73	35.70	1785%	80%	659%	76%	1
8	40.14	38.23	1911%	80%	579%	76%	1
9	100.31	40.14	2007%	80%	499%	76%	1
10	66.90	57.38	2869%	80%	419%	76%	1
11	73.05	57.48	2874%	80%	339%	76%	1
12	32.82	66.90	3345%	80%	259%	76%	1
13	80.30	73.05	3652%	80%	179%	76%	1
14	35.70	80.30	4015%	80%	99%	76%	1
15	80.40	80.40	4020%	80%	19%	76%	1
16	57.48	80.73	4036%	19%	0%	76%	1
17	29.25	89.19	4460%	0%	0%	0%	0
18	29.76	100.31	5016%	0%	0%	0%	0
19	100.56	100.56	5028%	0%	0%	0%	0
20	51.13	100.56	5028%	0%	0%	0%	0
			Total	1219%		1219%	16

Furthermore, the system performs server activation (E)" by considering events for high load and a very low load / unresponsive through calculation of servers used and determining the value of balance by assessing the value of the smallest fitness divided by the number of servers used as in Equation 6.

$$balancing = \frac{\min(fitness(S_x))}{roundup(\frac{\min(fitness(S_x))}{threshold\ high\ load})} \tag{6}$$

Thus, the system can adjust and balance the ability of the server and specify the number of servers needed as can be seen in Table 7. A total of 50 client needs with 3209 tasks need 16 servers; the average load balance of each server is 76%. The illustration of these functions is shown in Figure 9; the top picture is a real condition or CPU maximum capabilities after balancing process is obtained with the required number of servers with respective load balancing as presented in the following table.

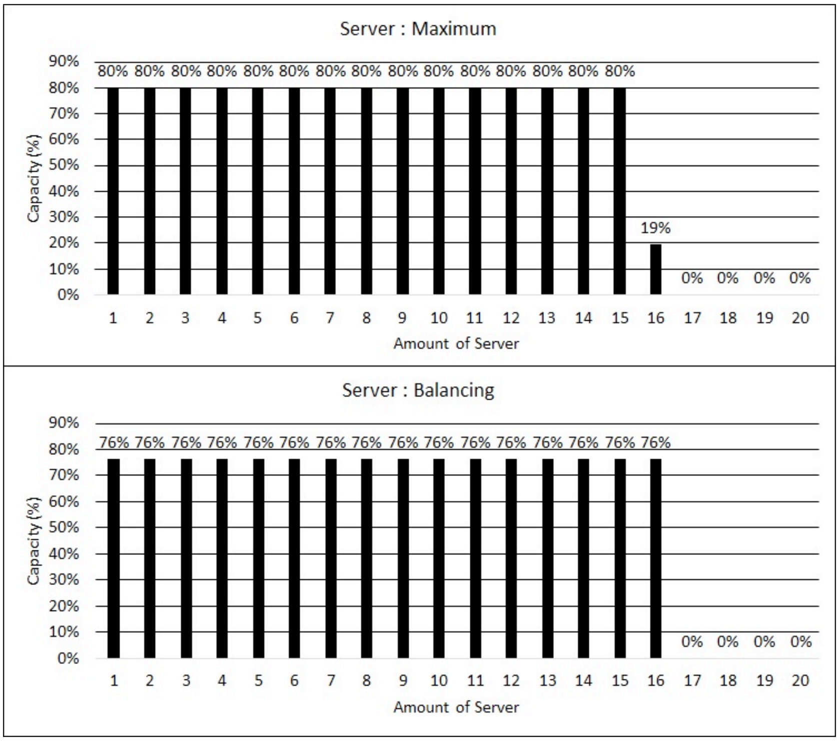
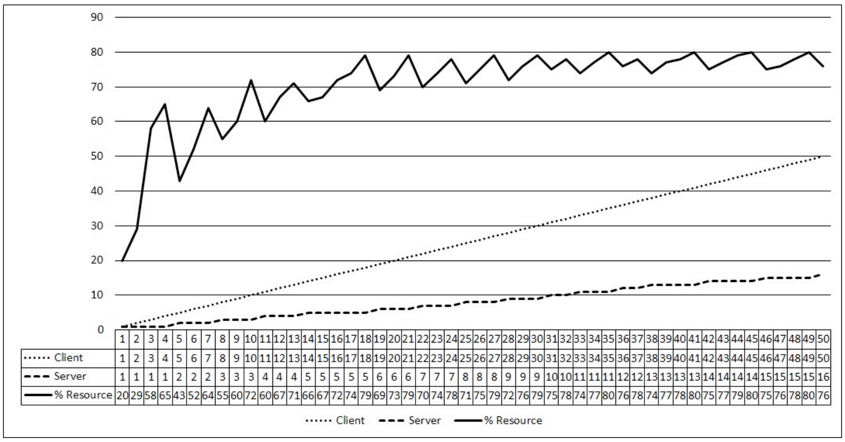


Figure 9. Illustration of server load balancing.

Evaluation

The evaluation consists of three activities: first, illustrating the service scalability; second, the model comparison to assess the design support in handling variability using the domain Quality Attribute Scenarios (dQAS); and third, evaluating adaptation maturity levels using Adaptive Capability Maturity Model (ACMM). The scalability of service system is related to the growth in the number of each CI in the service catalog at runtime. As an example of evaluation, scalability description of load balancing system is represented by growth in the number of clients and the load of each client that can continue to grow and change at runtime. As shown in Figure 10, the total 3209 tasks of 50 clients require 16 servers with an average load of 76%; but if the total tasks of clients change, for example increase or decrease the need for servers, then the average balance of the load will be adjusted. For example, with a maximum number of tasks of 45 clients, then 14 servers with an average load balance is activated automatically; if the maximum number of tasks is for 24 clients, then only 7 servers are enabled. If the maximum number of tasks is for 30 clients, then only 9 servers are enabled, and so on. Thus, the evaluation results show that the scale is linear with the number of clients and the number of server load for balancing size. Thus, the system is able to handle change and growth in context.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Client	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Server	1	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5	5	5	6	6	6	7	7	7	8
% Resource	20	29	58	65	43	52	64	55	60	72	60	67	71	66	67	72	74	79	69	73	79	70	74	78	71

	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
Client	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
Server	8	8	9	9	9	10	10	11	11	11	12	12	13	13	13	13	14	14	14	14	15	15	15	15	16
% Resource	75	79	72	76	79	75	78	74	77	80	76	78	74	77	78	80	75	77	79	80	75	76	78	80	76

Figure 10. Scalability of service process.

Furthermore, we evaluated the model through dQAS (Abbas, Andersson, & Weyns, 2012) to compare the adaptability of the RECCA model with the proposed model in the same case study. The dQAS characterized the quality attributes in the configuration management system experiment (Figure 3) through 9 dQAS elements, as shown in Table 8. The evaluation results showed that the proposed model provided more response alternatives in each variant, VC. From 6 combinations (variant VC), there were 20 total responses that could be alternative solutions for all stimuli. In addition, the responses were given to both normal and overload operating conditions. Meanwhile, the RECCA model had 13 total responses and was applied only under normal operating conditions, and there were some unsupported response requirements, such as R2, R5 and R6. This suggested that the proposed model could reduce the uncertainty factor caused by variability at runtime where requirements could be realized through alternative designs.

In the RECCA model, we describe the evaluation based on the criteria and controls of the ACMM (Gill, 2015). The evaluation results indicated that the maturity level of the adaptation model varies (level-4 or 5) depending on the adaptation cycle applied. Meanwhile, based on the proposed model and the artifacts generated from the RECCA model, adaptation maturity is definitely at level-5 (adapting) which is the highest level of adaptive capability maturity model. Figure 11 shows the maturity criteria of each level in the ACMM, and the achievement of level-5 is made through the interaction cycle with scans and sense patterns of changes and adjustments between the context and rationalization (service level) based on the MAPE-K pattern, i.e.: the ability to monitor, assess, and respond to changes for continuous adaptation is realized through the integration of goal models as contextual environments (target system) and adaptation cycles of the MAPE-K pattern. There is integrated engagement and governance for adaptation through artifacts generated from the architecture and alignment views that are managed through an adaptability view and there is good support for adaptation through automatic computing mechanisms at the service level.

Table 8

Evaluation of domain quality attribute scenarios

Element	**RECCA Model**	**Proposed Model**
Source (SO)	[SO1] Access device; [SO2] User; [SO3] Service features	
Stimulus (ST*)*	[ST1] New devices; [ST2] User's role changes; [ST3] Unavailability of features [ST4] Service failures	
Artifact (A*)*	[A1] User interface fragment; [A2] Service application fragment; [A3] Service delivery fragment	
Environment (E*)*	[E1] Runtime under normal operating; [E2] Runtime under overload operating	
Response (R*)*	[R1] Select access devices (P_1); [R2] Add new device type (P_2); [R3] Change service delivery (P_3); [R4] Send notification to user and service desk (P_4); [R5] Calculate service level availability (SPoF); [R6] Service reconfiguration (CI: load balancing); [R7] Service reconfiguration (CI: cloud adoption); [R8] Service delivery (configuration item (CI))	

(continued)

Element	RECCA Model	Proposed Model
Variants (V)	[V1] Role detection; [V2] Access services; [V3] User Authority; [V4] Feature detection; [V5] Change detection; [V6] Sevice Status; [V7] Service observation; [V8] Service reconfiguration; [V9] Service release	
Valid QAS Configurations (VC)	[VC1] V1 ∧ V2 ∧ V3; [VC2] V4 ∧ V5 ∧ V6; [VC3] V7 ∧ V8 ∧ V9; [VC4] V2; [VC5] V5; [VC6] V8	

Fragment Constraints (FC)	[Mandatory] {SO1, SO2, SO3} ∧ {A1, A2, A3} ∧ {E1} ∧ {R1, R8}
	[Variant VC1] {ST1, ST2} ∧ {R3, R4}
	[Variant VC2] {ST3} ∧ {R3, R4, R7}
	[Variant VC3] {ST3, ST4} ∧ {R3, R4, R7}
	[Variant VC4] {ST1} ∧ {R4}
	[Variant VC5] {ST2, ST3} ∧ {R3, R4, R7}
	[Variant VC6] {ST4} ∧ {R7}
		[Mandatory] {SO1, SO2, SO3} ∧ {A1, A2, A3} ∧ {E1} ∧ {R1, R8}
		[Variant VC1] {ST1, ST2} ∧ {R2, R3, R4}
		[Variant VC2] {ST3} ∧ {R3, R4, R7}
		[Variant VC3] {ST3, ST4} ∧ {E2} ∧ {R3, R4, R5, R6, R7}
		[Variant VC4] {ST1} ∧ {R2, R3, R4}
		[Variant VC5] {ST2, ST3} ∧ {R3, R4, R7}
		[Variant VC6] {ST4} ∧ {E2} ∧ {R5, R6, R7}

Response Measure (RM)	[RM1] R1, access device detected within x seconds [RM2] R2, not supported [RM3] R3, CI in the service catalog is changed within x seconds [RM4] R4, notifications are sent within x seconds [RM5] R5, not supported [RM6] R6, not supported [RM7] R7, service request sent within x seconds [RM8] R8, CI is sent within x seconds	[RM1] R1, access device detected within x seconds [RM2] R2, new device type added within x seconds [RM3] R3, CI in the service catalog is changed within x seconds [RM4] R4, notifications are sent within x seconds [RM5] R5, critical CI is specified within x seconds [RM6] R6, load balancing function is added within x seconds [RM7] R7, service request sent within x seconds [RM8] R8, CI is sent within x seconds
Total Responses	13 (normal operation)	20 (normal operation and overload)

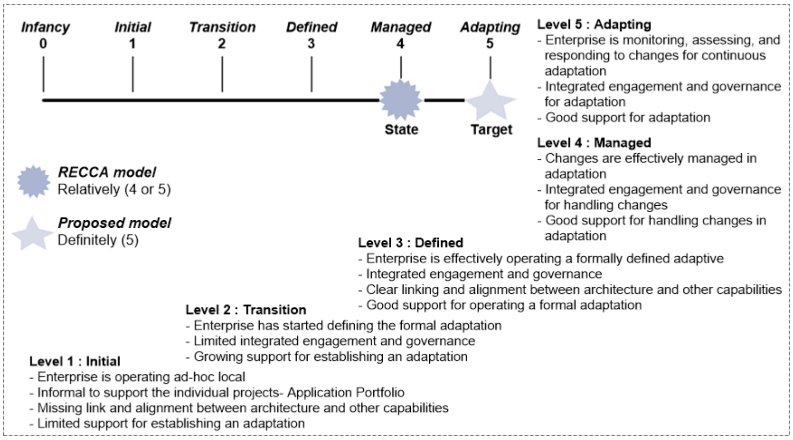

Infancy 0
Initial 1
Transition 2
Defined 3
Managed 4
Adapting 5
State
Target
RECCA model
Relatively (4 or 5)
Proposed model
Definitely (5)
Level 5 : Adapting
- Enterprise is monitoring, assessing, and responding to changes for continuous adaptation
- Integrated engagement and governance for adaptation
- Good support for adaptation
Level 4 : Managed
- Changes are effectively managed in adaptation
- Integrated engagement and governance for handling changes
- Good support for handling changes in adaptation
Level 3 : Defined
- Enterprise is effectively operating a formally defined adaptive
- Integrated engagement and governance
- Clear linking and alignment between architecture and other capabilities
- Good support for operating a formal adaptation
Level 2 : Transition
- Enterprise has started defining the formal adaptation
- Limited integrated engagement and governance
- Growing support for establishing an adaptation
Level 1 : Initial
- Enterprise is operating ad-hoc local
- Informal to support the individual projects- Application Portfolio
- Missing link and alignment between architecture and other capabilities
- Limited support for establishing an adaptation

Figure 11. Adaptive capability maturity model.

CONCLUSION AND FUTURE WORK

This paper introduces an adaptation model to address service variability. The lifecycle of each service element within the AESS metamodel is formulated as a control loop (MAPE-K) pattern based on the description of requirements. Adaptation mechanisms are realized through proactive and reactive adaptation scenarios, both of which are service requirements that can be viewed as a set of variants selected at runtime through a concept of variability rules for service change and evolution. The evaluation results showed that the proposed model is able to describe the scalability of services related to change and growth of new service requirements. The proposed model has an alternative design that is better than the previous work in variability modeling, where an alternative response in each variant, VC is capable of handling any stimulus under normal operating and overload conditions. In addition to these achievements, the adaptive capability maturity of the proposed model also improves the results of previous work.

Future research could detail additional features to enrich the description of service system requirements, as well as expand the context inference mechanism of the rule editor to accommodate more sophisticated conflict resolutions. Some approaches, such as strategies for machine learning and requirements reflection, could be taken into account in further studies.

ACKNOWLEDGMENT

The work conducted in this study was supported by the Ministry of Research, Technology and Higher Education of the Republic of Indonesia (No. 181.A/ADD/UN58.21/LT/2017).

REFERENCES

Abbas, N., Andersson, J., & Weyns, D. (2012). Modeling variability in product lines using domain quality attribute scenarios. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/2361999.2362028

Abbas, N., Andersson, J., & Weyns, D. (2018). ASPLe : A methodology to develop self-adaptive software systems with systematic reuse.

Abeywickrama, D. B., & Ovaska, E. (2017). A survey of autonomic computing methods in digital service ecosystems. *Service Oriented Computing and Applications*, *11*(1), 1–31. https://doi.org/10.1007/s11761-016-0203-8

Abuseta, Y., & Swesi, K. (2015). Design patterns for self adaptive systems engineering. *International Journal of Software Engineering*

& Applications (IJSEA), *6*(4), 11–28. https://doi.org/10.5121/ijsea.2015.6402

Anna, D. D., Dalpiaz, F., & Dastani, M. (2019). Requirements-driven evolution of sociotechnical systems via probabilistic reasoning and hill climbing. *Automated Software Engineering*. https://doi.org/10.1007/s10515-019-00255-5

Aradea, D., Supriana, I., Surendro, K., & Darmawan, I. (2017). Integration of self-adaptation approach on requirements modeling. In T. Herawan, R. Ghazali, N. M. Nawi, & M. M. Deris (Eds.), *Recent advances on soft computing and data mining (*pp. 233–243). Springer International Publishing.

Arcaini, P., Riccobene, E., & Scandurra, P. (2015). Modeling and validating self-adaptive service-oriented applications. *ACM SIGAPP Applied Computing Review*, 15, 35–48. https://doi.org/10.1145/2835260.2835262

Clark, K., Warnier, M., & Brazier, F. M. T. (2011). Self-adaptive service monitoring. In A. Bouchachia (Ed.), *Adaptive and intelligent systems* (pp. 119–130). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-23857-4_15

Gill, A. Q. (2015). *Adaptive cloud enterprise architecture*. In World Scientic Publishing Co. Pte. Ltd. https://doi.org/10.1142/9789814632133_fmatter

Hirsch, D., Kramer, J., Magee, J., & Uchitel, S. (2006). Modes for software architectures. In V. Gruhn & F. Oquendo (Eds.), *Software architecture* (pp. 113–126). Springer Berlin Heidelberg.

Knauss, A., Damian, D., Franch, X., Rook, A., Müller, H. A., & Thomo, A. (2016). ACon: A learning-based approach to deal with uncertainty in contextual requirements at runtime. *Information and Software Technology*, 70, 85–99. https://doi.org/https://doi.org/10.1016/j.infsof.2015.10.001

Lee, E., Seo, Y.-D., & Kim, Y.-G. (2019). Self-adaptive framework based on MAPE loop for Internet of things. *In sensors* (Vol. 19, Issue 13). https://doi.org/10.3390/s19132996

Lloyd, V., & Rudd, C. (2011). ITIL version 3 service design. *The office of government commerce*, 449. https://doi.org/10.1016/j.im.2003.02.002

Mendonça, D. F., Rodrigues, G. N., Ali, R., Alves, V., & Baresi, L. (2016). GODA: A goal-oriented requirements engineering framework for runtime dependability analysis. *Information and Software Technology*, 80, 245–264. https://doi.org/https://doi.org/10.1016/j.infsof.2016.09.005

Morandini, M., Penserini, L., Perini, A., & Marchetto, A. (2017). Engineering requirements for adaptive systems. *Requirements Engineering*, *22*(1), 77–103. https://doi.org/10.1007/s00766-015-0236-0

Nakagawa, H., Ohsuga, A., & Honiden, S. (2012). *Towards dynamic evolution of self-adaptive systems based on dynamic updating of control loops*.

2012 IEEE Sixth International Conference on Self-Adaptive and Self-Organizing Systems, 59–68. https://doi.org/10.1109/SASO.2012.17

Paz, A., & Arboleda, H. (2016). A model to guide dynamic adaptation planning in self-adaptive systems. *Electronic Notes in Theoretical Computer Science*, 321, 67–88. https://doi.org/https://doi.org/10.1016/j.entcs.2016.02.005

Perini, A. (2012). *Self-adaptive service based applications: Challenges in requirements engineering*. 2012 Sixth International Conference on Research Challenges in Information Science (RCIS), 1. https://doi.org/10.1109/RCIS.2012.6240416

Qureshi, N. A., Jureta, I. J., & Perini, A. (2012). Towards a requirements modeling language for self-adaptive systems. In B. Regnell & D. Damian (Eds.), *Requirements engineering: Foundation for software quality* (pp. 263–279). Springer Berlin Heidelberg.

Qureshi, N. A., & Perini, A. (2010). *Continuous adaptive requirements engineering: An architecture for self-adaptive service-based applications*. 2010 First International Workshop on Requirements@Run.Time, 17–24. https://doi.org/10.1109/RERUNTIME.2010.5628553

Solano, G. F., Caldas, R. D., Rodrigues, G. N., Vogel, T., & Pelliccione, P. (2019). *Taming uncertainty in the assurance process of self-adaptive systems: A Goal-oriented approach*. 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 89–99. https://doi.org/10.1109/SEAMS.2019.00020

Surendro, K., Aradea., & Supriana, I. (2016). Requirements engineering for cloud computing adaptive model. *Journal of Information and Communication Technology*, 2180- 3862, 15, 1–17.

Zavala, E., Franch, X., & Marco, J. (2019). *Adaptive monitoring: A systematic mapping*. In Information and Software Technology (Vol. 105, pp. 161–189). Elsevier B. V. https://doi.org/10.1016/j.infsof.2018.08.013

Zavala, E., Franch, X., Marco, J., & Berger, C. (2020). HAFLoop: An architecture for supporting highly adaptive feedback loops in self-adaptive systems. *Future Generation Computer Systems*, 105, 607–630. https://doi.org/https://doi.org/10.1016/j.future.2019.12.026