Received: February 23, 2021. Revised: May 11, 2021.

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Inference Model for Self-Adaptive IoT Service Systems

Aradea Aradea[1]* **Rianto Rianto[1]** **Husni Mubarok[1]**

[1]*Department of Informatics, Faculty of Engineering, Siliwangi University, Indonesia*
* Corresponding author's Email: aradea@unsil.ac.id

Abstract: Internet of Things (hereafter, IoT) service is a complex system because it should meet miscellaneous domain forms represented physically and virtually. The main challenge of IoT is to provide an inference model to resolve the dynamic context on a run-time basis. The system should have the ability to catch instances or concrete IoT services. On the other side, it should have the capability to adapt to the newest evidence of contexts. This paper introduces an inference model consisting of an IoT structure service artifact, a subsystem of contextual knowledge, and a subsystem of run-time adaptability reasoning. The results of model implementation on monitoring system of coronavirus disease revealed that the ability to adapt continuously and provide various alternative solutions to handle uncertain contexts, which is refered to sensor, network and server failure. The example of experiment result when a sensor failure occurs, the data received by the main server from the node is the average of the three previous data.

Keywords: Autonomic computing, Inference model, Self-adaptive systems, IoT service, Service knowledge.

1. Introduction

Internet of Things (hereafter, IoT) and its network architecture are complex systems since it was built from various integrated IoT devices shaped to distribute services to comply with the general objective of IoT applications [1]. To illustrate, IoT service systems exist in modern software environments with the run-time characteristic. Most of the systems driven by data indicate a dynamic character, an uncertain function, high connectivity, and scalability. It can produce significant risks and difficulties to evaluate at the design-time [2]. Likewise, IoT service systems are implemented in various lives, such as health, agriculture, traffic management, retail, logistic, remote monitoring, smart cities, process automation, etc. IoT can call be called a new paradigm providing a set of new services for every innovation of future generating technology. This situation could turn up various kinds of problems considering various utility factors, such as domain application, middleware domain, networks domain, object domain, etc. [3].

P. O. Antonino, A. Morgenstern, B. Kallweit, M. Becker, and T. Kuhn, [4] contend that adaptability is a property adhering to IoT-enabled cyber-physical systems. It should self-adjustment with a growing situation of context in which the system is running. Besides, the architecture should be well-designed. In this case, the service systems can adjust to their environment in a real-time manner with an appropriate element distribution level [5]. The statements have become a motivation for the writers to foster an IoT service system possessing adaptive capability based on related components of reasoning. These steps have been initiated by defining generic service artifacts and their inference model to accommodate various IoT sources.

I. L. Yen, F. Bastani, S. Y. Hwang, W. Zhu, and G. Zhou [6] had initiated to define IoT services artifact requirements. They extent the services model of software to support IoT specifications. As a result, service artifacts have flexibility in the implementation of various domains. Conversely, the effort has not yet been examined, such as the need for adapting the produced service artifacts. Generally speaking, the system requirements for different IoT application domains have not been well- defined [1]. This situation aligns with how the service model can meet the specifications of a very dynamic variety of

IoT service domains. The main challenge is providing self-adaptive systems to meet the scope and life cycle of IoT systems [7]. It is relevant to the need for an inference model that can automate the control of service artifacts for the future needs of IoT services. By doing so, the services are expected to be more sensitive, automatically-adaptive, organized, smart, autonomously-acting based on the environmental events of a context.

This paper introduces an inference model for self-adaptive IoT service systems cultivated based on autonomic computing and confidence factors. The remaining paper consist of section two that describe related work, section three depict proposed method, that consists of basic model, inference model components and model of an inference rule, section four illustrate implementation model to case study that consist of case specification, experiment, and evaluate of experiment result. Lastly, this paper closed by conclusion and future work.

2. Related works

A plethora of experts had investigated miscellaneous problems and complexities related to the provision of IoT services. As an example, S. Y. Shin, S. Nejati, and M. Sabetzadeh, [8] proposed software-defined networks (SDN) of IoT service. The approach was claimed to be able to solve network stagnation in a real-time manner. Also, it could reduce network utilization, data transmission delay, and adaptation cost. For this reason, it remains crucial to ensure the criteria for quality service. Another fact, M. Moghaddam, E. Rutten, and G. Giraud, [9] scrutinized the systematic literature review about adaptive middleware supporting the Internet of Things (IoT) and Cyber-Physical Systems (CPS). They notably accentuated on identifying various middleware designs of reactive/proactive in an IoT service system. Moreover, M. Etemadi, M. Ghobaei-Arani, and A. Shahidinejad, [10] proposed an approach to provide IoT services system efficiently. This approach was assumed to be able to increase and decrease dynamic scaling from cloud resources. In particular, it functions to accommodate works load from IoT service on the cloud computing environment. Overall, these previous studies only focused on the ability of IoT service systems viewed from the networks and middleware domains.

Other experts in other domains, such as A. Urbieta, A. González-Beltrán, S.B. Mokhtar, M.A. Hossain, and L.Capra, [11], introduced a model of IoT service based on a context-aware specification to undertake to reason toward user tasks and service behaviors. The reasoning is related to various IoT environment sources. Grounded in such a notion, they proposed a solution to emphasize adaptive service composition to support dynamic reason. Additionally, D. Mocrii, Y. Chen, and P. Musilek, [12] had explored various needs of IoT services forming elements, including the needs that should be filled from the system of architecture concept, software, communication technology, privacy, and security. Referring to the relevant investigations, there have been a number of points that the IoT service system should have. One of them is the ability to adapt as a requirement from the system and its environment.

Required adaptability in IoT service had been initiated in the writers' previous studies [13, 14]. As a matter of fact, the development was inspired by I. L. Yen, F. Bastani, S.Y. Hwang, W. Zhu, and G. Zhou, [6] and I. Supriana, K. Surendro, Aradea, and E. Ramadhan, [7] integrated with the ontological approach [15]. The model provides terminology to represent all IoT service artifacts, including metadata to the entire attributes and the relation between IoT services. Service artifacts consist of constructing elements of the requirement model matching IoT services concepts or classes. Unfortunately, this model has not possessed particular features about how the context class interrelates to the general knowledge for fulfilling the needs of the artifact service class. In addition, the model has not touched the issue of how to control the process in artifact service class running in run-time through a particular mechanism. To fill this void, the present study aimed at fostering an inference model for IoT services of adaptation needs at run-time consisting of various environmental resources.

3. Proposed method

3.1 Basic model

The self-adaptive systems (hereafter, SAS) objective is to realize the self-adjustment behavior of a particular software called adaptation requirements [16, 17]. This situation requires a system to acquire a number of aspects, namely (a) the ability to recognize a change in the application domain, (b) the capacity to determine transformation needed on a system based on application domain shifting, and (c) the capability to make changes in itself to produce alternative behavior [18]. Adaptation requirements on SAS are related to specification, refinement, and priority of responses to the change at the run-time [19]. SAS developers had proposed various approaches to fill the needs based on the focus of their

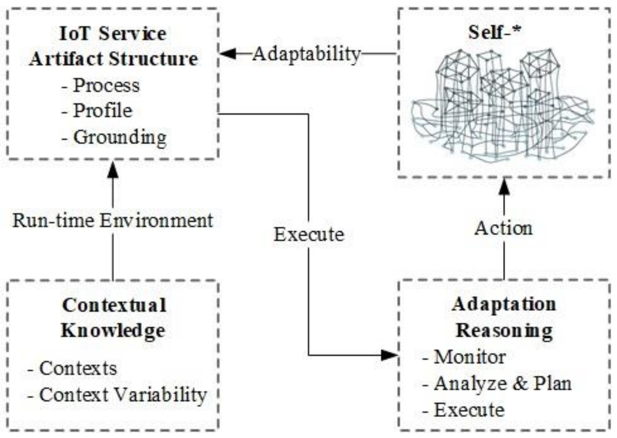


Figure. 1 Self-adaptive IoT service systems

problems. Thus, there is no a widely-approved definition or a required specification of adaptation [21]. This causes a scarcity of approaches to develop adaptation needs, including IoT service systems of adaptation requirements.

IoT is a modern software comprising a run-time property. This complex system could make a significant risk and a difficult evaluation at the design-time. Nevertheless, the run-time can complement the design-time evaluation [2]. This notion showcases the necessity of model development bridging situations either at the design-time or the run-time. With this in mind, changes and evolution of IoT service systems can be handled in real-time.

This study proposes an inference model representing the knowledge service of the IoT domain. The IoT knowledge service is prepared at the design-time. However, it potentially indicates a new fact about context at the run-time. To fill this requirement, the current study adopts an autonomic computing approach formulated as Event-Condition-Action (ECA) rules based on the confidence factors approach. More specifically, this study employed a model representing a general service artifact to accommodate various IoT resources based on self-adaptive architecture. Inference to the adaptability of service artifacts is determined by the service artifact catching the instance or concrete IoT service based on known context. Therefore, this system works autonomously at the run-time. This proposed model is an extent of the writer's previous works [13, 14].

3.2 Inference model component

The main component of the proposed model in this study is outlined in Fig. 1. It consists of (a) IoT service artifact structure, (b) contextual knowledge subsystem, and (c) adaptation reasoning subsystem.

IoT service artifact structure refers to a class deploying to catch instances or a concrete IoT service. It is fostered based on an ontological approach. This structure is an extent of the work of I. L..Yen, F. Bastani, S. Y. Hwang, W. Zhu, and G. Zhou [6]. They utilized the design pattern approach of autonomic computing and self-adaptation grounded in contextual requirements. The proposed IoT service artifact is displayed subsequently.

a. The process represents IoT service operation encompassing required service composition and reasoning mechanism at the run-time.
b. A profile represents both functional properties (F) and non-functional (NF) of IoT services from every composition of monitored services at the run-time.
c. Grounding is a technical detail representation of IoT services property comprising context attribute (C) as a random variable related to both functional (F) and non-functional (NF) of IoT services.

Contextual knowledge subsystem is a detailed representation of IoT services property both functional (F) and non-functional (NF) are extended from the contextual requirement approach [22-24]. The subsystem is constructed by accommodating uncertainty factors. It implemented a contextual attribute as a random variable representing two factors as outlined below:

a. Context (C) is a feature of the IoT domain relevant to a set of environmental assumptions.
b. Context variability (Cv) dimension is a factor representing the occurred IoT domain changes at the run-time.

The adaptation reasoning subsystem is a mechanism of monitoring and determining adaptation action at the run-time toward each IoT service. Patterned reasoning is cultivated to adopt an autonomic computing approach [25, 26, 27]. It is labeled as MAPE-K (monitor, analyze, plan, execute-knowledge) adaptation pattern, wherein:

a. Monitor (M) is a process to scan and collect context information from IoT services based on situational services and their environments.
b. Analysis & Plan (AP) is a reasoning process for analyzing and planning an action suitable for each IoT service.
c. Execution (E) is a process to determine adaptation appropriate to each IoT service.

3.3 Model of an Inference Rule

Inference model to execute contextual knowledge of reasoning toward each need of IoT services adaptation is defined by the following couples of rules:

Rule-1: Service Artifact & Contextual Requirements

- Each $F_i \in \{F_1, F_2, ..., F_n\}$ in IoT services artifact has a random variable context $C_i \in \{C_1, C_2, ..., C_n\}$ influencing to each $NF_j \in \{NF_1, NF_2, ..., NF_n\}$

Rule-2: Service Artifact & Monitoring Requirements

- Each $F_i \in \{F_1, F_2, ..., F_n\}$ containing random a variable context $C_i \in \{C_1, C_2, ..., C_n\}$ is monitored based on Cv dimension.

Rule-3: Service Artifact & Evidence Requirements

- Each $C_i \in \{C_1, C_2, ..., C_n\}$ in services artifact F_i and NF_j is influenced by evidence (Ev) related to a set of environmental assumptions and IoT domain changes/Cv dimension at run-time.

Rule-4: Service Artifact & Reasoning Requirements

- Each $F_i \in \{F_1, F_2, ..., F_n\}$ has a random variable context C_i analyzed and planned by AP. The adaptation action to execution service into a particular circumstance based on Ev and Cv at an M monitoring process.

Table 1 describes the algorithm for mechanism requirement toward contextual knowledge of each IoT service according to four rules. The inference model is constructed through an autonomic computing approach based on the MAPE-K pattern represented as ECA rules. Event-related to the context of IoT services monitored at the run-time, the condition related to changes of context condition of IoT services. Also, context at the run-time and action is a behavior of adaptation of IoT services on handling context variability at run-time.

Rules in knowledge base built as rules of the model editor. The developer can perform some operations like adding or specification changes by renewing the knowledge base directly or put it back. These rules function as a policy engine and system administrator adjusting to the system of the policy every time according to preference and its needs through available interfaces.

The reasoning mechanism based on the developed policy engine adopts the confidence factors (CF) approach [28, 29]. In particular, it is formulated in ECA rules. CF is utilized to determine alternative behavior options for IoT services when the system executes (action) based on the results of the monitor (event) and Analysis & Plan (condition). The basic formulation of CF refers to the ECA rules:

$$If\ C_i \leftarrow Ev\ Then\ Cv \leftarrow Ha \tag{1}$$

Where:

- $C_i \leftarrow Ev$ is evidence of context (C_i) influencing service functional (F)
- $Cv \leftarrow Ha$ is a hypothesis to determine alternative action of variability context (Cv) based on priority/suggestion evidence of quality/ non-functional (NF) service.
- The scope of a particular value ranges between 0 to 1.

Table 1. Adaptation of IoT service pattern algorithm

Adaptation of IoT service pattern

```
Input
Fi ← value in Ci
NFj ← value in Ci
Do
Let
    (Fi, NFj) ← inference model
// Monitor (M) Pattern
for all Fi and NFj in service artifact, do
    Fi ← get values Ci in service artifact
    NFj ← get values Ci in service artifact
    for each values Ci in IoT service artifact, do
        if S.system Cv in service artifact ← evidence Ev then
            send information (S.system(Cv)) to analyzerManager
        end if
    end for
end for
// Analysis & Plan (AP) Pattern
for each S.system Cv in analyzerManager do
    Ci ← joint value (S.system(Cv)) from Fi and NFj
    S ← update actual Ci as S.System Cv
    for each S.system (actual Ci) do
        (Fi, NFj) ← ECA reasoning
        if Cv = new evidence Ev (Ci) then
            create an adaptation request (new Ci requirements) and
            update value (new S.system Ci) for service artifact
        end if
    end for
end for
// Execute (E) Pattern
for all actual Ci (new S.system) in-service artifact do
    changeService ← new changeService in service artifact
    send changeService to one or more executors
    for each changeService in executor do
        S.system ← set a new value for service artifact
        actuator ← update behavior via one or more actuators
    end for
end for
Output
Self-adaptation for IoT service artifact based on the control loop
```

Received: February 23, 2021. Revised: May 11, 2021.

So the CF rule:

$$CF\,(Ha, Cv) = CF\,(C_i, Cv)\,CF\,(Ha, Ev) \qquad (2)$$

Where:

- *CF* (C_i, Cv) is a certainty factor of C_i fact making antecedent from rules based on uncertainty Cv fact.
- *CF (Ha, Ev)* is a certainty factor in the hypothesis assuming that certain known fact if CF $(C_i, Cv) = 1$.
- *CF (Ha, Cv)* is the certainty factor hypothesis that based on uncertainty fact Cv by priority/ suggestion fact of quality/ NF services.

If *CF* (C_i, Cv) consists of C_1, C_2, C_n, then applies min [*CF* (C_1, Cv), *CF* (C_2, Cv), *CF* (C_n, Cv)] * *CF (Ha, Ev)*. Subsequently, the results will determine the rule raised to determining adaptation action based on the highest certainty value. The weight of each CF value from C_n is to state a confidence level. It can be obtained from the confidence of experts, data users, or inferences from another rule on a rule basis, previous data, uncertain data, and monitoring infrastructure-based rules. The rule of the editor model realized this model is recognized as an engine policy.

4. Case study: continuous Covid-19 monitoring system

Currently, the coronavirus disease (henceforth, Covid-19) outbreak has become the most striking issue for people around the world. Even, World Health Organization (WHO) has officially decided such an outbreak as a pandemic. Against this background, this study is dedicated to these circumstances. More specifically, it is expected to be one of the solutions to the development of application in the field of intelligent software systems. Fig. 2 deciphers a continuous Covid-19 monitoring system case. The problem domain of this case raises issues discussed by previous researchers. As an example, M. Kamal, A. Aljohani, and E. Alanazi, [30] attempted to adjust and expand their empirical investigation based on the proposed model of this research.

4.1 Case specification

The environmental system includes an actor as a general user, a patient, a doctor, and other health stakeholders. The monitored resources are the medical track record of the patients, disease data, user positions, and regions classified by the Covid-19 spread spectrum. In this case, all monitored sources are related to wearable devices. Moreover, they have rapid changes and face uncertainty influenced by an uncertain contextual variability. This situation is caused by changes, data growths, invalid information, failures of system or service infrastructures, and unpredictable novel conditions.

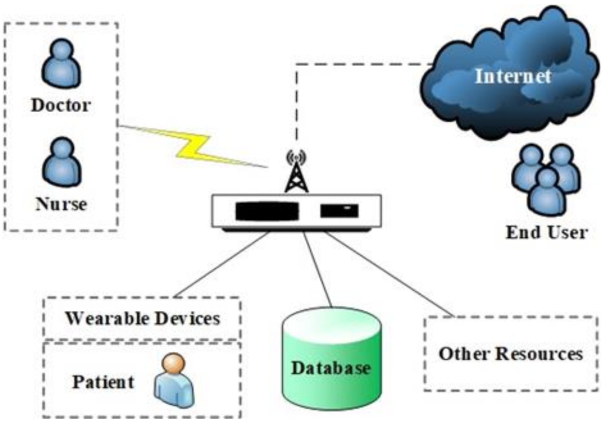


Figure. 2 A continuous covid-19 monitoring system

Table 2. Case specification of IoT services

Component	Specification
IoT service artifact	SA: user, patient, doctor, health stakeholder F: management of user, management of a patient, management of symptoms, management of areas NF: accurate, rate SR: map of the area, medical record, user data, diseases data
Contextual knowledge	C_1: patient status; C_2: zone type; C_3: user position; C_4: user type; C_5: sensor failures; C_6: sever faults; C_7: networks inference
Adaptation requirements	Cv_1: event to classified patient status Cv_2: event defining zone type Cv_3: event determining user position Cv_4: event to determine user types Cv_5: event $C_1 \wedge (C_5 \vee C_6 \vee C_7)$
Note: SA: service actor; F: functional service; NF: non-functional service; SR: service resources; C: contexts; Cv: context variability	

Referring to Fig. 2, the Covid-19 monitoring system should be able to conduct continuous monitoring. This ability can be realized through the adaptability system as a response to incidents both in the system itself or its environment. In other words, the continuous monitoring system is specified based on the main component of the proposed inference model consisting of IoT service artifacts, contextual knowledge subsystem, and adaptation reasoning subsystem. On each model of components, specification of every variability accommodates all needs of IoT service environment, such as contextual variability of services system, IoT environment, and adaptation requirements at the run-time, as illustrated in Table 2.

4.2 Experiment

Experiments in this study were divided into two main parts. The first part described the system experiment in normal conditions, namely the system monitors and services management. This first system experiment was based on service functions and meets the quality attributes (non-functional). In the second experiment, several situations were carried out where the service system was exposed to the variability of the system itself and its environment. These cover changes in context and disruption or failure of service system infrastructure.

4.2.1. Experiment-I: IoT context monitoring

In relation to the case specification in Table 2, adaptation requirements in normal situations of IoT service systems define patient status, classify zone type, determine user position, and define user type. Monitored Context variability (*Cv*) related to $C_i \in \{C_1, C_2, C_3, C_4\}$ and values of the variable for each context C_1: patient status (Normal, ODP (Orang Dalam Pemantauan): Person in Surveillance, PDP (Pasien Dalam Pemantauan): Patient in Surveillance, PPC (Pasien Positif Covid-19): Patient Positive Covid-19, PSC (Pasien Sembuh Covid-19): Patient Recovering from Covid-19); C_2: zone types (green, yellow, orange, red); C_3: user position (spatial coordinate); and C_4: user types (general user, patient, doctor, nurse). The first experiment considered scenarios to determine a patient's status or a person's status. A person/patient connected to smart medical devices to monitor his/her health condition. The data of patient condition apart from being used as a reference to determine a patient status. Also, it is used as a report to determine the Covid-19 zone classification. These data became the primary sources of information for each type of IoT service user in various coordinate positions.

The adaptation pattern of IoT service systems in managing patient data is the monitoring component (M) to monitor. Besides, it functions to collect C_1 contextual information from smart medical devices connected to patients. Furthermore, the Analysis & Plan (AP) component analyze and plans follow-up actions to deliver the information. Finally, the executing (E) component determines the action best suited for C_2, C_3, and C_4 context requirements. The simulation carried out in this first experiment is a dynamically changing context information $C_i \in \{C_1, C_2, C_3, C_4\}$. Table 3 indicates the C_1 context data.

Table 3. The fact of C_1 context at run-time

Day	BT	TH	GS	SS	Status
1	< 38	1	0	0	0
2	<38	1	0	0	0
3	>38	1	1	0	1
4	>38	1	1	2	1
5	>38	1	1	3	1
6	>38	1	1	3	1
7	>38	1	2	2	2
8	>38	1	3	2	2
9	>38	1	3	2	2
10	>38	1	3	1	3
11	>38	1	3	1	3
12	>38	1	2	1	3
13	>38	1	2	1	3
14	<38	1	2	0	3
15	<38	1	2	0	3
16	<38	1	0	0	4

Note:
BT (Body Temperature): Celsius
TH (Traveling History): 0. No; 1. Yes
GS (General Symptom): 0. No; 1 Cough; 2. Flu; 3. Breathless
SS (Specific Symptom): 0. No; 1. Loss of Smell; 2. Breathless
Status: 0. Normal; 1. ODP; 2. PDP; 3. PPC; 4. PSC

More specifically, Fig. 3 demonstrates the simulation of changes in the IoT domain occurring at the run-time in C_i for the C_1 context.

The data in Table 3 illustrate changes in service users monitoring C_1 (Cv_1) on the Covid-19 cases. The monitoring process was carried out by the monitoring component (M) applying wearable sensors, including the heart rate and temperature sensors. The graph revealed a change in the status of a user suspected of being a Covid-19 case. The statuses were categorized as ODP, PDP, PPC, PSC, and returning to the Normal category. These changes were analyzed by employing the Analysis & Plan (AP) component based on a rule translating data from the monitoring component (M) in the IoT device.

The filter applied in rules of analyzing & plan (AP) component was data from body temperature monitoring device combined with traveling and disease histories or occurred symptoms. Traveling histories were divided into several zones based on government regulation, namely green, yellow, orange, and red. The colors represented significant levels of Covid-19 spread. Diseases and symptoms histories were utilized to determine the level of surveillance or observations toward user/patient. Eventually, executing (E) components automatically sent the newest data to the system periodically. Likewise, it adjusted to some other context value changes.

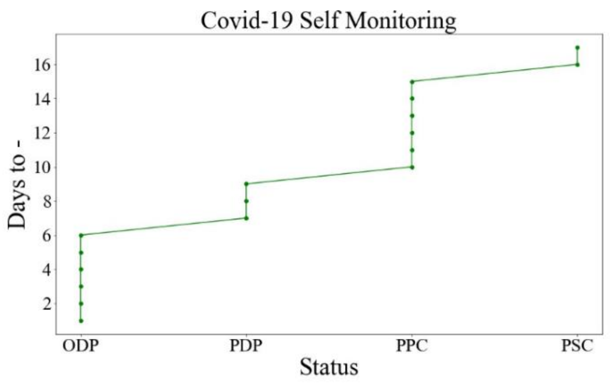


Figure. 3 Changes of C_1 (Cv_1) at run-time

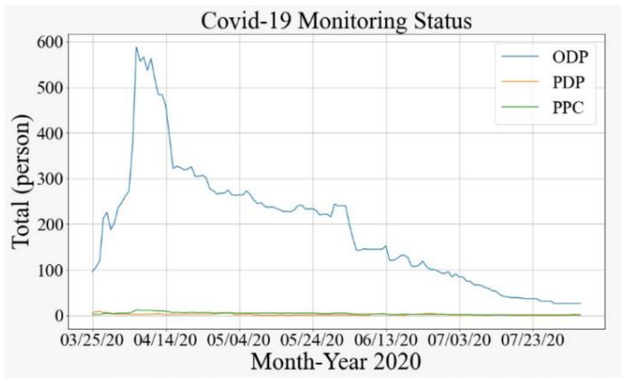


Figure. 4 Transformation of C_4 context based on the changes C_1 at run-time

Adjustment of context value affected by value changes of data at the run-time as illustrated in Fig. 3. In particular, it made IoT services systems monitor every context of Covid-19 continuously. Fig. 4 depicts context adjustment of adaptation initiated when monitoring component (M) existed at every context C_2, C_3, C_4 to take the new data value from context C_1. Consequently, it was applied to Cv_2, Cv_3, Cv_4. Furthermore, the AP component updated data values in the system to display the actual situation.

As an example, Fig. 4 designates an updating value of context C_4 data based on changes of C_1 context. Technically, it enabled to produce of an appropriate transformation with changes in self-monitoring status by all users/patients. In this situation, the AP component performed aggregation toward each change of all users/patients' statuses. At the same time, it allowed general users (C_3 context/user types) to identify the changes. The aggregation is performed daily. It was classified based on the months. Furthermore, the zoning or context (C_2) of area/place can be determined because zoning was affected by the user/patient status (NOR, ODP, PDP, PPC, and PSC). Finally, each executing component (E) for the context of C_2, C_3, and C_4 presented the actual state of the COVID-19 monitoring system in real-time. Given these facts, the monitoring activities carried out by this system have a continuous nature.

4.2.2. Experiment-I: IoT context variability

In the second experiment, the IoT service was faced with unexpected context variability. The monitored context variability (Cv) was related to $C_j \in \{C_5, C_6, C_7\}$. This condition would affect the non-functional (NF) of the service, namely accuracy, and speed. Employing the same scenario in the first experiment, the IoT service system was affected by uncertain factors, including C_5 (e.g. sensor failure), C_6 (e.g. severe problems), and C_7 (e.g. network outage). The developed scenario was a combination of several contexts, such as C_i and C_j. In this case, C_i and C_j should be able to handle the following situations, namely Cv_5: event $C_1 \wedge (C_5 \vee C_6 \vee C_7)$. The developed adaptation pattern was the monitor component (M) to context information of monitor C_i and C_j from several installed IoT service infrastructures. Furthermore, the Analysis & Plan (AP) component analyzed and planned the handling action against Cv_5 variability based on the reasoning process. Finally, the executing (E) component determined the most suitable adaptation action for each variability.

Another evidence focuses on the IoT service. The IoT service in this experiment was related to the C_1 context information. To illustrate, the patient status accessed by the service user was faced with a state of C_5 or C_6 or C_7. Hence, the system should adapt to keep the service runs. When the monitor component (M) identified the symptoms of interference between the three possible contexts, the Analysis & Plan (AP) component analyzed and planned adaptive actions through the generated rules. The target was to fulfill functional (F) services in managing patient status information based on service quality/non-functional (NF) service factors, namely accuracy, and speed. On the one hand, Context C_5 highlights the sensor failure corresponding to factor NF_1 (accuracy). On the other hand. The context C_7 (network disturbance) and C_6 (sever noise) corresponded to (NF_2 factor) speed. Briefly stated, three contexts (C_5, C_6, and C_7) can have adverse/negative (-) good/positive (+) effect on both NF quality factors depending on the choice of action taken by the system in determining the most appropriate alternative solution.

Grounded in the above description, the requirements to manage the variables of patient status information services remain vital. This enabled to showcase of the infrastructure failures, such as sensor, server, or network failures. Those variables were service event, infrastructure type, and error rate.

Dealing with this, the AP component would conduct a trigger on the rules based on each property referring to the recognition results of the monitor (M). These rules provided alternative behavior in managing context ($C_5 \vee C_6 \vee C_7$). Possible IoT service situations are as follows:

Rule-1: if (infrastructure_type = sensor) and (error_state = null) then service_event = recognition service
Rule-2: if (infrastructure_type = server) and (error_state = null) then service_event = storage service
Rule-3: if (infrastructure_type = network) and (error_state = null) then service_event = service delivery
Rule-4: if (infrastructure_type = not null) and (error_state = not null) then service_event = sensor exception handling service or internal storage service or software-defined networks service
Rule-5: if (infrastructure_type = new type) and (error_state = null) then service_event = create new infrastructure type
Rule-6: if (infrastructure_type = null) and (error_state = null) then service_event = change service delivery
Rule-7: if (error_state = not null) then service_event = send notification to user and service desk
Rule-8: if not [criteria] then service_event = change service delivery

Based on the abovementioned rules, the service plan (SP_n) was obtained. As shown in Table 4, SP_n provided alternative behavioral options for IoT service systems selected through service needs. The most appropriate behavior would be determined by the executing (E) component based on the reasoning results of the *Analysis* & *Plan* (AP) component through the policy engine.

A form of adaptation in IoT service-I event is selecting action of $SP_{1.1}$, $SP_{1.2}$, $SP_{1.3}$ services. When errors did not occur in service infrastructure, it applied standard services. Service adaptation on IoT service-III event and IoT service-1V event was incorporated into service software evolution. It was the system performing the process of adding, changing, and deleting component instances required by IoT services based on the actions of SP_3 and SP_4. Service adaptation needed by IoT service-V event was related to automatic feedback. In particular, the system sent notification SP_5 to users and the service desk related to current service information for considering the next actions.

Table 4. ECA IoT service event

Event (E)	**Condition (C)**	**Action (A)**
IoT service event-I	*(infrastructure_type = sensor); (infrastructure_type = server); (infrastructure_type = network); (error_state = null).*	$SP_{1.1}$ *= recognition service* $SP_{1.2}$ *= storage service* $SP_{1.3}$ *= service delivery*
IoT service event-II	*(infrastructure_type = not null); (error_state = not null).*	$SP_{2.1}$ *= sensor exception handling service* $SP_{2.2}$ *= internal storage service* $SP_{2.3}$ *= software-defined networks service*
IoT service event-III	*(infrastructure_type = new type); (error_state = null).*	SP_3 *= create new infrastructure type*
IoT service event-IV	*(infrastructure_type = null); not [criteria].*	SP_4 *= change service delivery*
IoT service event-V	*(error_state = not null).*	SP_5 *= send a notification to the user and service desk*

Meanwhile, the IoT service-II event related to the failures of IoT service infrastructure, such as sensor failures, erroneous servers, and network disruption.

Required adaptation in IoT service-II event functions to handle the variability of C_i context combination and C_j. IoT service encountered service an infrastructure failure when changes occurred to

Table 5. The fact of C_j context of IoT service at run-time

C_5					
Time	S_1	S_2	S_3	(C_i, Cv)	(Ha, Cv)
4	0,8	0,9	0,8	0,8	0,6
5	0,8	0,7	0,9	0,7	0,9
10	0,8	0,9	0,9	0,8	0,2
C_6					
Time	S_1	S_2	S_3	(C_i, Cv)	(Ha, Cv)
2	0,9	0,8	0,8	0,8	0,1
5	0,8	0,6	0,7	0,6	0,4
9	0,9	0,7	0,7	0,7	0,7
10	0,7	0,9	0,8	0,7	0,8
C_7					
Time	S_1	S_2	S_3	(C_i, Cv)	(Ha, Cv)
4	0,7	0,6	0,5	0,5	0,3
5	0,8	0,9	0,8	0,8	0,4
10	0,9	0,9	0,9	0,9	0,8
11	0,8	0,7	0,8	0,7	0,9

the C_1 context. As a result, the system was required to determine recovery priority among C_5 context options (sensor failure related to NF_1 accuracy factor), C_7 context (network disturbance), C_6 (sever error related to NF_2 speed factor). Table 5 presents facts in the context of IoT services. Time was the days when service infrastructure disruption occurred during the monitoring process. S_1, S_2, S_3 were the symptoms appearing from any context of C_j at that time, such as (Ci, Cv) and (Ha, Cv) according to Eq. (2). The outlined scenario is the value of the data captured from two sources. First, sources from the monitoring infrastructure monitoring the state of the sensors, servers, and network at run-time as an uncertainty factor in the captured data and rules. Second, sources derive from the expert of confidence and previous data as a policy engine, namely setting constraints (Ha, Cv). This source functions to indicate the IoT service quality priorities based on the quality attributes of NF_1 (accuracy related to the context of C_5: sensor failure) and NF_2 (speed associated with C_7: network disruption) and C_6 (problems of a server).

The quality attribute constraints of NF_1 was prioritized to the patient with ODP and PDP status. Thus, high accuracy was needed to detect the situation. The range value of weight was determined at about 0.6 to 1. This NF_1 quality attribute had a low priority value or decreases when the patient's condition was in PPC and PSC status with the weighted values set in the range of 0 to 0.5. Meanwhile, the quality attributes constraints of NF_2 were inversely proportional to NF_1, namely displaying high priority when the patient's condition was in PPC and PSC status. This became low when the patient's condition was in ODP and PDP statuses. Considering what was used during a patient was in PPC or PSC status, the speed factor was prioritized over the accuracy factor. This was required when the patient was in the ODP or PDP status and vice versa. The range of weighted scores was assigned to the NF_2 quality attribute when high and low priority was similar to the NF_1 quality attribute. Specifically, Table 6 designates the options of the $SP_{2.1}$, $SP_{2.2}$, $SP_{2.3}$ service actions. Similarly, Table 4 demonstrates selected options through the system based on Eq. (1) and (2).

Fig. 5 illustrates the behavior simulation of the event of the IoT service-II system. In this case, at a particular time, slice-2 of the monitor (M) component detected an occurrence of the C6 context. The Analysis & Plan (AP) component performed reasoning based on priority in the policy engine. The executing (E) component determined the SP2.2 action by generating component instances from the internal storage service module.

Table 6. IoT service event-II at run-time

Time	$SP_{2.1}$	$SP_{2.2}$	$SP_{2.3}$
1	0	0	0
2	0	0,08	0
3	0	0	0
4	0,48	0	0,15
5	0,63	0,24	0,32
6	0	0	0
7	0	0	0
8	0	0	0
9	0	0,49	0
10	0,16	0,56	0,72
11	0	0	0,63
12	0	0	0
13	0	0	0
14	0	0	0

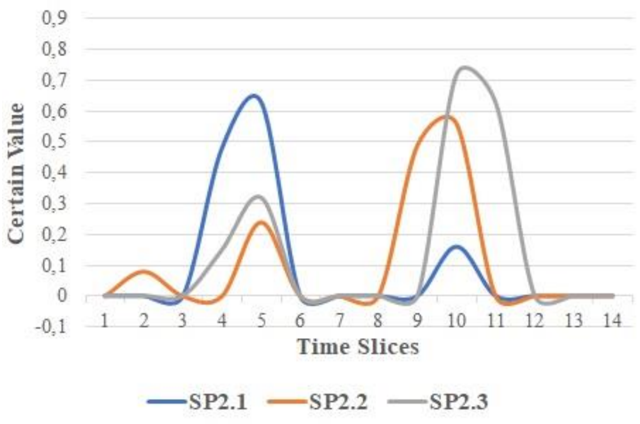


Figure. 5 The behavior of IoT service based on Cv_5 at run-time

In the time of slice-4, the system faced two options for recovery action caused by the $C_5 \vee C_7$ context. In the time of slice-5, it developed to become three options of recovery actions based on $C_5 \vee C_6 \vee C_7$ contexts. The system determined service recovery priorities based on the highest value weight, namely at the time of slice-4. The system generated component instances from the $SP_{2.1}$ module (sensor exception handling service) first. Then, it was followed by the $SP_{2.3}$ module (software-defined networks service). Meanwhile, in the time of slice-5, service recovery was focused on $SP_{2.1}$, module. Next, it continued to $SP_{2.3}$. Finally, it arrived at $SP_{2.2}$.

Adaptation action in the time of slice-5 made the IoT service system return to the normal state. However, it started from the time of slice-9 until the time of slice-11 in which the monitor (M) component identified reappearance of service infrastructure disruption. For this reason, the Analysis & Plan (AP) component immediately performed reasoning to determine recovery options. On the other hand, the executing (E) component brought back the service to

Received: February 23, 2021. Revised: May 11, 2021.

a normal state at the time of slice-12. As a matter of fact, the Adaptation behavior of IoT services displayed in Fig. 5 revealed that Covid-19 monitoring was a continuous system. In this sense, if infrastructure or service failure occurs, the recovery of services performs in run-time. With this in mind, these system abilities call as self-adaptive IoT service systems.

4.3 Experimentation

This experiment was carried out to broaden the adaptability view of the research conducted by (P. Michiel, W. Danny and S. Marlon, K. Yentl, W. Danny) in terms of the continuity of data transmission when failure occurs in the context of the uncertainty of our proposed model, namely sensor, network, and server failures.

The experiment for simulating the adaptation model in this paper is carried out by following the architecture in the figure 6.

Figure 7 describes the condition of the main server that is running and processes data sent from the node (RASPI) and processes data from the server aggregator.

Figure 8 depict the simulation result, that describes the system adaptation process when a run-time failure occurs. The failure scenario refers to the condition of the infrastructure which consists of sensor, network and server failures.

The first simulation when a sensor failure occurs, the node cannot receive data from the sensor. Automatically the data sent to the main server, taking from the average of the previous data stored in local storage (node).

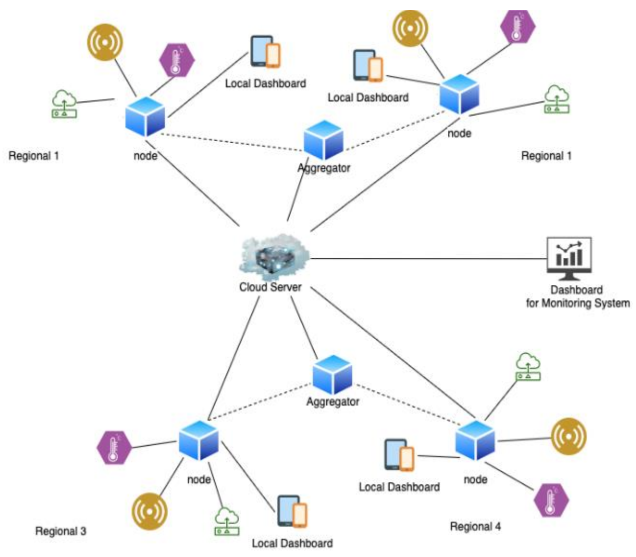
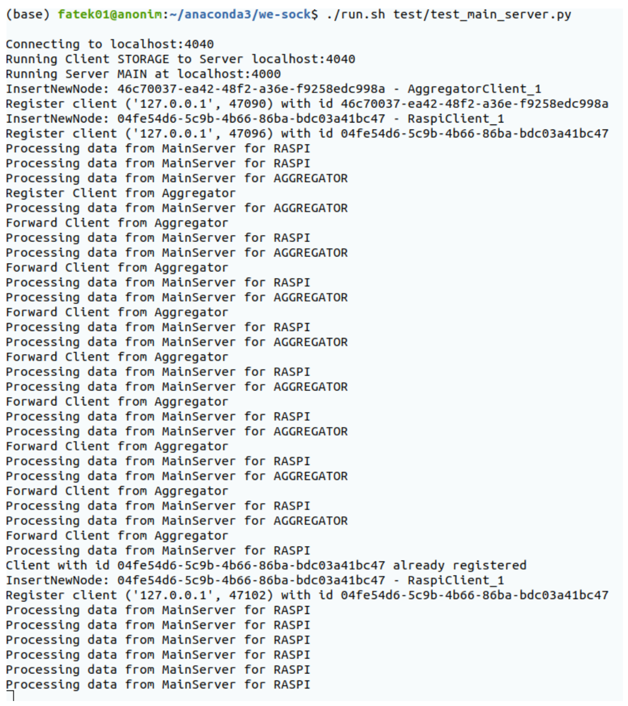


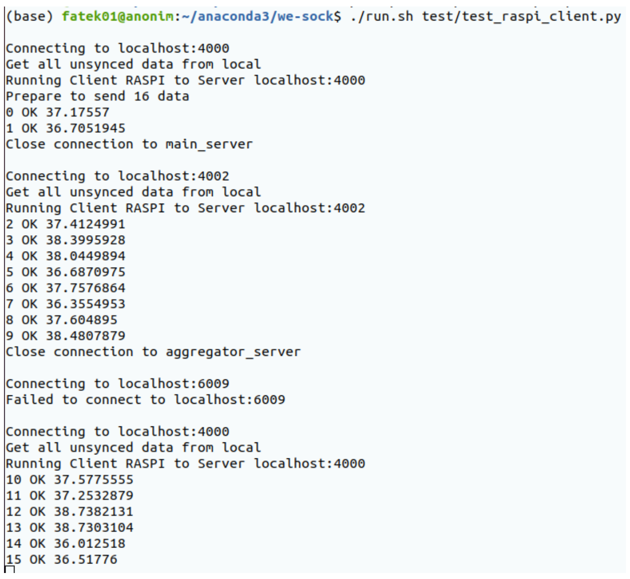
Figure. 6 Architecture of simulation adaptation model



```
(base) fatek01@anonim:~/anaconda3/we-sock$ ./run.sh test/test_main_server.py

Connecting to localhost:4040
Running Client STORAGE to Server localhost:4040
Running Server MAIN at localhost:4000
InsertNewNode: 46c70037-ea42-48f2-a36e-f9258edc998a - AggregatorClient_1
Register client ('127.0.0.1', 47090) with id 46c70037-ea42-48f2-a36e-f9258edc998a
InsertNewNode: 04fe54d6-5c9b-4b66-86ba-bdc03a41bc47 - RaspiClient_1
Register client ('127.0.0.1', 47096) with id 04fe54d6-5c9b-4b66-86ba-bdc03a41bc47
Processing data from MainServer for RASPI
Processing data from MainServer for RASPI
Processing data from MainServer for AGGREGATOR
Register Client from Aggregator
Processing data from MainServer for AGGREGATOR
Forward Client from Aggregator
Processing data from MainServer for RASPI
Processing data from MainServer for AGGREGATOR
Forward Client from Aggregator
Processing data from MainServer for RASPI
Processing data from MainServer for AGGREGATOR
Forward Client from Aggregator
Processing data from MainServer for RASPI
Processing data from MainServer for AGGREGATOR
Forward Client from Aggregator
Processing data from MainServer for RASPI
Processing data from MainServer for AGGREGATOR
Forward Client from Aggregator
Processing data from MainServer for RASPI
Processing data from MainServer for AGGREGATOR
Forward Client from Aggregator
Processing data from MainServer for RASPI
Processing data from MainServer for AGGREGATOR
Forward Client from Aggregator
Processing data from MainServer for RASPI
Processing data from MainServer for AGGREGATOR
Forward Client from Aggregator
Processing data from MainServer for RASPI
Client with id 04fe54d6-5c9b-4b66-86ba-bdc03a41bc47 already registered
InsertNewNode: 04fe54d6-5c9b-4b66-86ba-bdc03a41bc47 - RaspiClient_1
Register client ('127.0.0.1', 47102) with id 04fe54d6-5c9b-4b66-86ba-bdc03a41bc47
Processing data from MainServer for RASPI
Processing data from MainServer for RASPI
Processing data from MainServer for RASPI
Processing data from MainServer for RASPI
Processing data from MainServer for RASPI
Processing data from MainServer for RASPI
```

Figure. 7 Running main server



```
(base) fatek01@anonim:~/anaconda3/we-sock$ ./run.sh test/test_raspi_client.py

Connecting to localhost:4000
Get all unsynced data from local
Running Client RASPI to Server localhost:4000
Prepare to send 16 data
0 OK 37.17557
1 OK 36.7051945
Close connection to main_server

Connecting to localhost:4002
Get all unsynced data from local
Running Client RASPI to Server localhost:4002
2 OK 37.4124991
3 OK 38.3995928
4 OK 38.0449894
5 OK 36.6870975
6 OK 37.7576864
7 OK 36.3554953
8 OK 37.604895
9 OK 38.4807879
Close connection to aggregator_server

Connecting to localhost:6009
Failed to connect to localhost:6009

Connecting to localhost:4000
Get all unsynced data from local
Running Client RASPI to Server localhost:4000
10 OK 37.5775555
11 OK 37.2532879
12 OK 38.7382131
13 OK 38.7303104
14 OK 36.012518
15 OK 36.51776
```

Figure. 8 Adaptation simulation at run-time

The second simulation describes the failure of the main network to the main server. Data will automatically be sent via another route, namely the aggregator server and the aggregator server to continue sending data to the main server.

The third simulation describes a server failure, if this happens, then the action taken by the node is to store data locally which will then be synchronized after the server is functioning again.

5. Conclusion

This study introduces the inference model for self-adaptive IoT services systems developed based on the autonomic computing approach and formulated as Event-Condition-Action (hereafter, ECA) rules supported by the confidence factors approach. The main component consists of (a) IoT service artifact representing knowledge service of the entire IoT resources, (b) contextual knowledge as a detailed representation of IoT service environment feature containing uncertainty, and (c) adaptation reasoning representing a mechanism of surveillance (observation) and reasoning needs for the run-time adaptation during every IoT service. This model is prepared to fulfill the service system requirements at the design-time to capture instances or concrete IoT service. However, it can reveal new evidence about service contexts in the run-time to continuously adapt to new facts from IoT service contexts. The simulation results in the application illustrate three uncertain contexts, namely sensor, network and server failures at run-time. The experimental results show that the main server still receives data, even though there is a sensor or network failure so that the monitoring process is not interrupted.

As an evaluation, this model was implemented in the case of the Covid-19 monitoring system. The experiment was operationalized in two conditions, namely in normal circumstances and during context changes and disruptions. First, in normal circumstances, the system monitors and regulates IoT services according to service functions to meet their quality attributes. Second, the IoT service system was faced with the variability of the system itself and its environment, namely context changes, disruptions, or service infrastructure failures. The results of implementing this case study provide the ability to adapt to IoT service systems sustainably. In addition, it offers a variety of alternative solutions in dealing with the uncertainty of the IoT service context at the run-time. Empirically speaking, future studies can adapt the results of this study as a foundation to construct the requirements of the cyber-physical system domain, accommodate various computational resources, develop learning approaches and optimize reasoning for adaptation planning to determine the configuration of the service system in the run-time.

Conflicts of Interest

The authors declare that there is no conflict of interest in this paper.

Author Contributions

Conceptualization, methodology, formal analysis, software, validation, reviewing & supervision, writing—review and editing, Dr. Aradea Aradea; Resources, investigation, formal analysis, validation, writing—original draft preparation, Rianto Rianto; Investigation, formal analysis, software, validation, writing—original draft preparation, Husni Mubarok.

Acknowledgments

The present study is supported by the Institute for Research, Community Services, and Education Quality Assurance of Siliwangi University. Likewise, this study is a manifestation of succeeding the strategic program of the Ministry of Research, Technology and Higher Education of the Republic of Indonesia (No.266/UN58.21/PG/2020) related to the research developments in Indonesia.

References

[1] L. F. Rahman, T. Ozcelebi, and J. Lukkien, “Understanding IoT Systems: A Life Cycle Approach”, *Procedia Computer Science*, Vol. 130, pp. 1057-1062, 2018.

[2] D. Sobhy, L. Minku, R. Bahsoon, T. Chen, and R. Kazman, “Run-time Evaluation of Architectures: A Case Study of Diversification In IoT”, *The Journal of Systems and Software*, Vol. 159, pp. 110428, 2020.

[3] A. Čolaković and M. Hadžialić, “Internet of Things (IoT): A Review of Enabling Technologies, Challenges, and Open Research Issues”, *Computer Networks*, Vol. 144, pp. 17-39, 2018.

[4] P. O. Antonino, A. Morgenstern, B. Kallweit, M. Becker, and T. Kuhn, “Straightforward Specification of Adaptation Architecture-Significant Requirements of IoT-Enabled Cyber-Physical Systems”, In: *Proc of IEEE International Conference on Software Architecture Companion*, pp. 19-26, 2018.

[5] H. Muccini, R. Spalazzese, M. T. Moghaddam, and M. Sharaf, “Self-Adaptive IoT Architectures: An Emergency Handling Case Study”, In: *Proc of 12th European Conference on Software Architecture: Companion Proceedings (ECSA '18)*, pp. 6, 2018.

[6] I. L. Yen, F. Bastani, S. Y. Hwang, W. Zhu, and G. Zhou, “*From Software Services To IoT Services: The modeling perspective*”, Vol. 10371, 2017.

[7] I. Supriana, K. Surendro, Aradea, and E. Ramadhan, “Self-Adaptive Cyber City System”, In: *Proc of International Conference On Advanced Informatics: Concepts, Theory And Application (ICAICTA)*, pp. 1-6, 2016.

[8] S. Y. Shin, S. Nejati, M. Sabetzadeh, L. Briand, C. Arora, and F. Zimmer, “Dynamic Adaptation of Software-Defined Networks for IoT Systems:

A Search-Based Approach", In: *Proc of IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 137-148, 2020.

[9] M. Moghaddam, E. Rutten, and G. Giraud, "Protocol for A Systematic Literature Review On Adaptative Middleware Support for IoT and CPS", 2020.

[10] M. Etemadi, M. Ghobaei-Arani, and A. Shahidinejad, "Resource Provisioning for IoT Services In The Fog Computing Environment: An Autonomic Approach", *Computer Communications*, Vol. 161, pp. 109-131, 2020.

[11] A. Urbieta, A. González-Beltrán, S. B. Mokhtar, M. A. Hossain, and L.Capra, "Adaptive And Context-Aware Service Composition for IoT-Based Smart Cities", *Future Generation Computer Systems*, Vol. 76, pp. 262-274, 2017.

[12] D. Mocrii, Y. Chen, and P. Musilek, "IoT-Based Smart Homes: A Review Of System Architecture, Software, Communications, Privacy and Security", *Internet of Things*, Vol. 1-2, pp. 81-98, 2018.

[13] Aradea, I. Supriana, K. Surendro, and H. Mubarok, "Self-Adaptation Modeling for Service Evolution On The Internet of Things (IoT)", In: *proc of IOP Conference Series: Materials Science and Engineering*, Vol. 550, No. 1, pp. 012026, 2019.

[14] Aradea, Rianto and H. Mubarok, "Development of Service Knowledge Model for IoT-Based Systems Adaptability", *Technical Report, Department of Informatics*, 2019.

[15] K. Siegemund, "Contributions to Ontology-Driven Requirements Engineering", *Doctoral Dissertation, Technischen Universität Dresden, Fakultät Informatik, Lehrstuhl Softwaretechnologie*, 2015.

[16] M. Salehie and L. Tahvildari, "Self-Adaptive Software: Landscape and Research Challenges", *ACM Transactions on Autonomous and Adaptive Systems(TAAS)*, Vol. 4, No. 2, pp. 1-42, 2009.

[17] S. Li, "Evaluating Mission-Critical Self-Adaptive Software Systems: A Testing-Based Approach", *Thesis, Electrical, and Computer Engineering*, 2010.

[18] N. Subramanian and L. Chung, "Software Architecture Adaptability: An NFR Approach", In: *Proc of the 4th International Workshop on Principles of Software Evolution*, pp. 52-61. 2001.

[19] J. Pimentel, M. Lencastre, and J. Castro, "Implicit Priorities In Adaptation Requirements", In: *Proc of 10th International Conference on the Quality of Information and Communications Technology*, pp.83-86, 2016.

[20] Aradea, I. Supriana, K. Surendro, and I. Darmawan, "Variety of Approaches In Self-Adaptation Requirements: A Case Study", In: *Herawan T, Ghazali R, Nawi N, Deris M (eds) Recent Advances On Soft Computing and Data Mining. Advances in Intelligent Systems and Computing*, Vol. 549, pp. 253-262, 2017.

[21] J. Andersson, R. D. Lemos, S. Malek, and D. Weyns, "Modeling Dimensions of Self-Adaptive Software Systems", In: *Cheng B.H.C., de Lemos R., Giese H., Inverardi P., Magee J. (eds) Software Engineering for Self-Adaptive Systems. Lecture Notes in Computer Science*, Vol. 5525, 2009.

[22] Aradea, I. Supriana, and K. Surendro, "Self-Adaptive Software Modeling Based On Contextual Requirements", *Telecommunication, Computing, Electronics, and Control (Telkomnika)*, Vol. 16, No. 3, pp. 1276-1288, 2018.

[23] Aradea, I. Supriana, and K. Surendro, "Self-Adaptive Model Based On Goal-Oriented Requirements Engineering for Handling Service Variability", *Journal of Information and Communication Technology (JICT)*, Vol. 19, No. 2, pp. 225-250, 2020.

[24] A. Lapouchnian, and J. Mylopoulos, "Capturing Contextual Variability In I* Models", In: *Proc of the 5th International I* Workshop*, pp. 96-101, 2011.

[25] Y. Abuseta and K. Swesi, "Design Patterns for Self-Adaptive Systems Engineering", *International Journal of Software Engineering & Applications (IJSEA)*, Vol. 6, No. 4, pp. 11-28, 2015.

[26] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing", *Computer*, Vol. 36, No. 1, pp. 41-50, 2003.

[27] Aradea, I. Supriana, and K. Surendro, "ARAS: Adaptation Requirements For Adaptive Systems - Handling Run-Time Uncertainty of Contextual Requirements", *Technical Report, School of Electrical Engineering and Informatics*, 2019.

[28] S. L. Kendal and M. Creen, *An Introduction to Knowledge Engineering*, pp. 243-247, 2007.

[29] S. J. Russell and P. Norvig, "Artiicial Intelligence: A Modern Approach (Third Edition)", *Prentice Hall Series in Artificial Intelligence*, 2010.

[30] M. Kamal, A. Aljohani, and E. Alanazi, "IoT Meets COVID-19: Status, Challenges, and Opportunities", *arXiv preprint arXiv:2007*, pp. 12268, 2020.

[31] M. Provoost and D. Weyns, “DingNet: A Self-Adaptive Internet-of-Things Exemplar”, In: *Proc of the IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 195-201, 2019.

[32] M. Saelens, Y. Kinoo, and D. Weyns, “HeyCitI: Healthy Cycling in a City using Self-Adaptive Internet-of-Things”, In: *Proc of IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, pp. 226-227, 2020.