TELKOMNIKA, Vol.16, No.3, June 2018, pp. 1276~1288
ISSN: 1693-6930, accredited **A** by DIKTI, Decree No: 58/DIKTI/Kep/2013
DOI: 10.12928/TELKOMNIKA.v16i3.7032

Self-adaptive Software Modeling Based on Contextual Requirements

Aradea*, Iping Supriana, Kridanto Surendro
School of Electrical Engineering and Informatics, Bandung Institute of Technology
Jl. Ganesha No 10, Bandung 40132, Indonesia
*Corresponding author, e-mail: aradea@unsil.ac.id, iping@informatika.org, endro@informatika.org

Abstract

The ability of self-adaptive software in responding to change is determined by contextual requirements, i.e. a requirement in capturing relevant context-atributes and modeling behavior for system adaptation. However, in most cases, modeling for self-adaptive software is does not take into consider the requirements evolution based on contextual requirements. This paper introduces an approach through requirements modeling languages directed to adaptation patterns to support requirements evolution. The model is prepared through contextual requirements approach that is integrated into MAPE-K (monitor, anayze, plan, execute - knowledge) patterns in goal-oriented requirements engineering. As an evaluation, the adaptation process is modeled for cleaner robot. The experimental results show that the requirements modeling process has been able to direct software into self-adaptive capability and meet the requirements evolution.

Keywords*: self-adaptive software, requirements modeling, contextual requirements, goal-oriented requirements engineering, rule based systems*

1. Introduction

The involvement of various elements of the real world that interact with software raises the needs of adaptive systems. Modeling for self-adaptive software is the answer to the problems and challenges. This activity determines the success or failure of a software system that can understand and act based on what is happening within its contextual requirements. In the area of self-adaptive systems, research on requirements engineering is much needed [1-3]. In fact, requirements engineering is an ongoing process because requirements are subject to change and must be managed throughout the system life cycle [4]. This is related to requirements evolution handling. Recent papers show a lack of research on requirements evolution for self-adaptive systems [5]. Further, approaches to linking requirements at design-time with runtime changing contextual requirements still require further investigation [1],[6-7]. Currently, techniques for executing requirements that depend on the relevant context are under-represented [8].

In this paper, we propose an approach where the system captures the real world conditions through a goal-based approach as the requirements description, and then transformed into software components through a control strategy as self-adaptive concept to establish adaptation behavior at run-time. Requirements modeling language adopted is i*/ Tropos model. The language is chosen because it has an intentional states perspective and a lightweight language [9]. So, it can represent a real-world context and is easy to use. However, this model still requires adjustment if it is used for requirements of self-adaptive software. The problems identified are related to the definition of the inheritance concept [10]. The current concept does not capture and represent the effect of contextual variability [11], which is the main characteristic of self-adaptive software. In addition i* model is still not able to describe the sequence of processes performed by agent. Meanwhile, in self-adaptive software, it is necessary to determine the adaptation patterns.

In this paper, we introduce (a) requirements modeling language that has embodied adaptation patterns through the extension of goal-oriented requirements engineering approach with MAPE-K control loops and context inheritance hierarchies, (b) control models for managing

Received August 18, 2017; Revised March 26, 2018; Accepted April 28, 2018

adaptation mechanisms which are realized through rule editor model, so that the addition or change of specification can be done by updating the knowledge base directly.

2. Proposed Method

The propose model, as shown in Figure 1, consists of (a) goal model-the representation of domain model that provides basic functions and contextual requirements, and (b) inference engine-which is the representation of a control model that manages the target system through adaptation patterns.

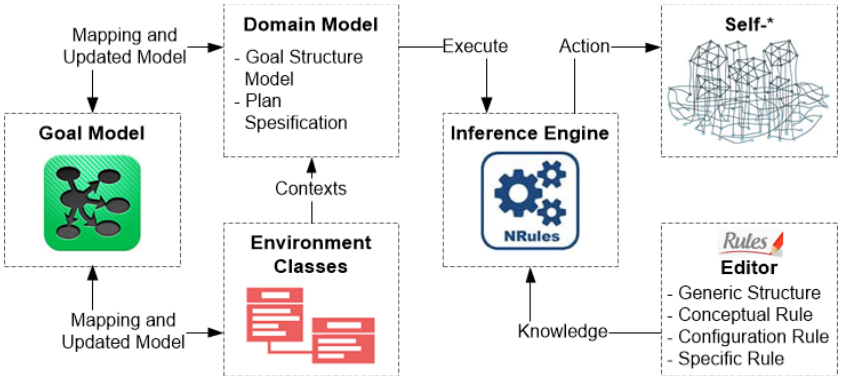

Figure 1. Model of self-adaptive software systems

The adopted goal approach is i*/ Tropos model [12-13], which is an agent-oriented modeling framework. In this approach, the agent can be viewed as part of a program used to represent social actors, individuals or organizations that have attributes and behaviors [14]. Further discussion can be seen in our previous paper [15]. Recently, the model has been expanded. So, it has the self-adaptation ability [16]. The proposed approach provides the ability to analyze variability at run-time, but here we complement it by mapping the self-adaptive software patterns and contextual requirements. In addition to defining the contextual requirements, the system is also directed to having the ability to monitor the variables of each decomposition of goal and plan entities attached to them. The system architecture can be seen in Figure 2.

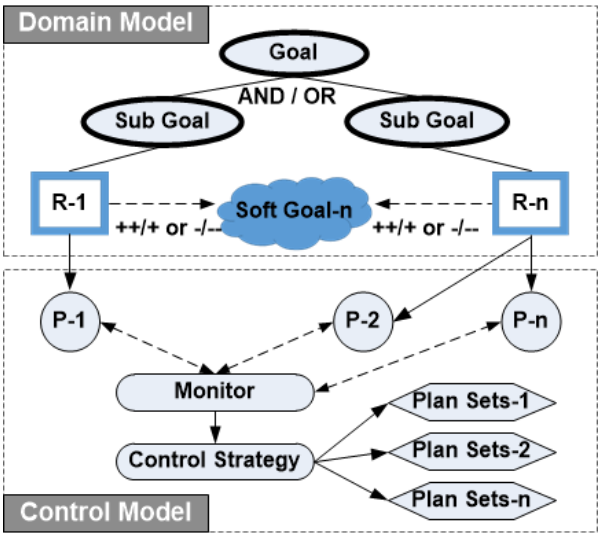

Figure 2. System architecture

On the domain model, goal (functional) is decomposed (AND / OR) into sub-goals. So, it can be identified by requirements ($R\text{-}_1$, $R\text{-}_2$, $R\text{-}_n$) from each goal that affect on certain parameters, and have positive or negative contribution (++ / + or - / -) to one or more softgoal (non-functional). In the control model, the properties ($P\text{-}_1$, $P\text{-}_2$, $P\text{-}_n$) of each of these goals are identified and transformed into software components, as well as observations on the possible changes. Further analysis was done through control strategies; the variation of adaptation was determined based on the determination rule defined as plan (Plan $Sets\text{-}_1$, Plan $Sets\text{-}_2$, Plan $Sets\text{-}_n$).

2.1. Domain model

In order to realize the system architecture shown in Figure 2, we expanded our previous work [17-18] by defining the mapping of goal models based on [19] into design patterns [2],[20-21] inspired by monitor-anayze-plan-execute-knowledge (MAPE-K) models [22] as shown in Figure 3. Currently, MAPE-K patterns have been acknowledged as the main characteristic of self-adaptation capabilities. So, we direct the requirements modeling language containing the pattern. A strategic rationale model consists of a number of nodes (goals, softgoals, resources, tasks, actors) and links (dependence links, decomposition links, means-ends links, contribution links) to describe the internal interactions of actors. These actors can relate to other actors, known as strategic dependency models. Each task decomposition links in i*/ Tropos represents a particular way in the system (Plan sets) as a sequential process. However, it is less able to describe the sequence of execution, especially for an AND-decomposed. This situation will be confusing if each goal/ task has a priority to implement adaptation patterns, such as monitor-anayze-plan-execute process in sequence. To solve this problem we use approach [10] so that task decomposition can be changed to clarify the sequence of processes.

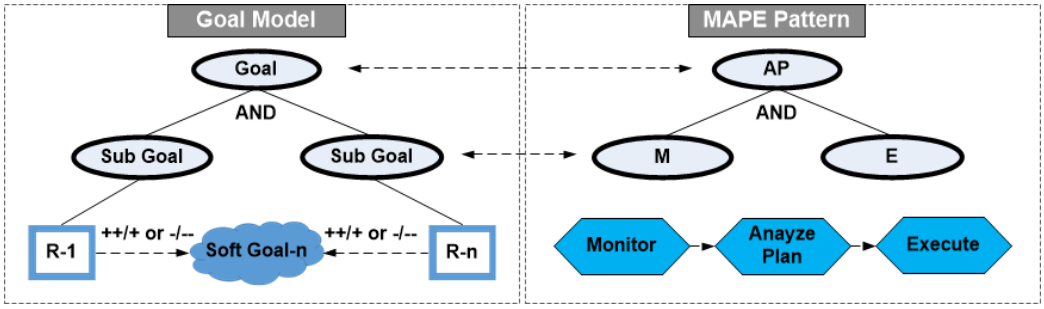

Figure 3. Mapping goal models into MAPE-K patterns

In addition, contextual variability handling as a key feature of self-adaptive software is still not covered in i*/ Tropos. Thus, we add this capability through contextual requirements (cr), ie requirements that apply in certain contexts [8],[23]. The concept is adapted from approach [11], in which related context is organized into context inheritance hierarchies, so that adaptation pattern (MAPE-K) embedded in the goal model becomes bound and can be adjusted when it should be active or deactivate. Figure 4 shows inheritance hierarchies context in goal model, where each goal/ sub goal can have contextual requirements (cr). If a parent's goal has cr, then it will be inherited to each child's goal, and child's goal can have another cr as shown in Figure 4a. If a parent's goal has more than one choice of cr, then cr also applies to each child's goal combined with cr of child's goal, as shown in Figure 4b. This principle is based on context-based visibility, ie a goal/ sub goal can only be achieved when a particular context (cr) is active (monitorable context) so that the model will be visible/ active and can be seen as a propositional DNF formula. In addition to monitorable context, this concept is also applied to domain changes (non-monitorable context) such as view points, versions, etc., in our version to meet requirements evolution. Rules for organizing it are accommodated through rule editor models discussed in section 2.2.

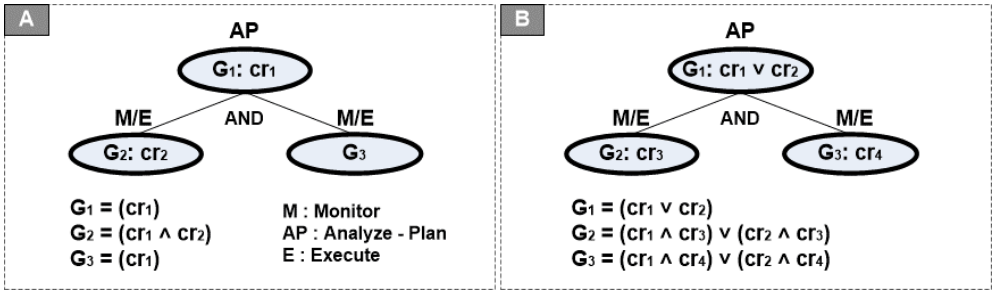

Figure 4. Context inheritance hierarchies in goal models

2.2. Control Model

The main objective of this strategy is to determine adaptation response options that are most relevant. The notation used to construct this algorithm consists of: (a) a goal model which consists of nodes (N) connected by attributes property (link) it owns, where each node consists of a number of state (S_n); (b) a set of state (S_n) which may consist of an initial status and target status influenced by several fact processings of contextual requiremets ($\sum : cr_n$) on the left hand side (LHS) which will determine the behavior of the action ($Q : a_n$) on the right hand side (RHS) through the transition function ($\delta : t_n$); (c) a set of contextual requirements (CR) within a parent node which can inherit its properties to each child node that binds the visibility of a series of nodes (N) to active; (d) recognition process conducted by observing $\sum : cr_n$ as trigger of S_n for each N, so as to determine the state as a reference for preparation of plans to realize a number of Q: a_n.

Determining requirements that apply in a context (as CR) basically deals with requirements evolution, ie changes in requirements, whether it adds new requirements or remove requirements that do not apply in certain contexts. So, the action behavior (Q: a_n) is defined as a set of states (S_n) which will determine the set of nodes (N) when to "activate" and / or "deactivate" based on the bonding rules of context inheritance hierarchies (CR) which are mapped into component operations [24] as shown in Table 1. In addition, we set a rule that represents each goal element in the component system. This rule utilizes the component mapping rules and goal decomposition rules [19]. If the goal decomposition is AND-decomposition, then the parent goal will need multiple relation (port) attribute for each child goal with one-to-one relationship. Meanwhile, if the goal decomposition is OR-decomposition, then parent goal will provide conditional relation (port) attribute to each child goal with one-to-many relationship. In this activity, setting the properties for each nodes (N) is also needed.

Table 1. Configuration Algorithm

Configuration of Component
N ← (nodes)
C ← (components)
for all N in goalModel do
N ← C : configuration components for operation
for each (∑, Q) ≠ ø do
(activate, deactivate) ← N
activate ← create component instance C from N = CR
deactivate ← delete component instance C from N = CR
end for
goalModel m ← reconfiguration(activate, deactivate)
enactModel(m)
end for

Control strategy utilizes some of the design patterns [2],[20-21] and modifies it in accordance with the contextual requirements which have been developed previously. Started with the function of the component M (monitor), this component will monitor the number of goal properties that represent system states at run-time. These activities are conducted in response to the request or event either as time-triggered or event-triggered. State system is a

combination of property values of internal and external goals (N and CR), and the deviation is detected based on the threshold. Analyze manager performs an analysis based on symptom list, where this symptoms list contains a set of symptoms that (CR: cr_1, cr_2 ... cr_n) have been set for the system or for storing new symptoms that will appear. If the analyze manager detects the presence of symptom, then the plan will receive adaptation requests, and reconfigure as adaptation response.

Reconfiguration algorithm represents the components of AP (analyze and plan), as shown in Table 2. The AP components contain a rule engine that has the event-condition-action (ECA) rules. The rule engine in our version is expanded with rule editor model, where the modification of the rule, for example update or change of the rule, can be made directly to knowledge base. Adaptation request is represented as a system state (S) which is detected based on the event that occurs, either based on the existing context inheritance hierarchies (CR) or new CR. A set δ can be expressed as $\delta \{t_n (cr_n, a_n) \mid n \neq \emptyset\}$, where cr_n is a fact of CR, and a_n is the expected action behavior (activate or deactivate) for a particular contextual requirements n (cr_n). The strategy used is forward strategy, which is to reuse existing fundamental component and match the required specifications. Then, change plan is executed by component E (execute) to perform adaptation actions. This component uses a number of actuators to bring the system back to the expected state.

Table 2. Reconfiguration Algorithm

Reconfiguration of Plan
S.system ← (init, target) *δ.selector ← (work, found, backs)* *for each S.system in analyzerManager do* *analyzer ← update(logs) actual S.system* *search(S.system) in symptomList* *if symptom ≠ ø then* *create(adaptationRequest(CR for N)) and* *update(adaptationRequest(CR for N)) for plan specification* *else* *addSymptom to symptomList and* *create(adaptationRequest) and* *send information(adaptationRequest) for plan specification* *end if* *end for* *for each adaptationRequest(S.system) do* *init ← set work N(∑, Q)* *while δ {t_n (cr_n, a_n) | n ≠ ø} do* *δ ← find that the LHS of the operator match with work say it found* *if found is only one then* *RHS ← set work(N ← cr_i)* *else if work is equivqlent(activate(N:cr_j)) with target(cr_i ← a_i) then* *stop succed* *end if* *if found is more then one then* *found ← set work one(N ← (cr_i ∨ cr_j)) of the found* *backs ← put rest (deactivate((N: cr_i ∨ cr_j) ← a_{ij})) of found* *end if* *if found is empty then* *else if backs is empty then* *stop failed* *else* *backs ← set work one(activate(N:cr_{ij})) of backs* *end if* *end if* *end for*

3. Results and Discussion

3.1. Case study: cleaner robot

Case motivation takes an example of the problems description that has been discussed by previous researchers [16] and [19] with regard to the cleaner robot scenario. Goal modelling for this case is shown in Figure 5; system requirements can be illustrated as follows: There are

a number of goal properties on context elements that must be monitored. They are (a) the environment property (external/ symptom): CR = cr_1: presence of waste, cr_2: critical battery power, cr_3: waste type, cr_4: preference; and (b) goal property (internal/ goal hierarchy): on Figure 5, functional system is represented as a goal and non-functional as softgoal. Possible goal changes that can occur include: (a) goal of robot behavior which is influenced by the condition of battery power and waste objects; and (b) goal of cleaning affected by the waste type encountered.

3.2. Experiment

In Figure 5, manage waste goals must be achieved when cr_1 is valid and cr_1 is inherited to each sub goals so that it binds all of its components to MAPE-K patterns, and some of its sub goals have another cr, ie cr_3 and cr_4 so that It is achieved when (cr_1 ∧ cr_3) and (cr_1 ∧ cr_4). Manage battery goals must be achieved when cr_2 is valid and achievement of one sub goal when cr_4 is valid so that (cr_2 ∧ cr_4).

Meanwhile, the setting of behavior goals will be achieved when cr_1 or cr_2 is valid and cr_4 is applicable to one subgoal. So, its achievement is when CR=(cr_1 ∧ cr_4) ∨ (cr_2 ∧ cr_4). This condition will become a determining factor when the pattern of some MAPE-K should be activated and/or deactivated. There are a number of object properties as shown in Table 3. The waste object is an object that should be cleaned by the robot by observing the battery power condition and how to clean it. The plug object is the object for battery charging of robot with properties that can be seen in Table 4.

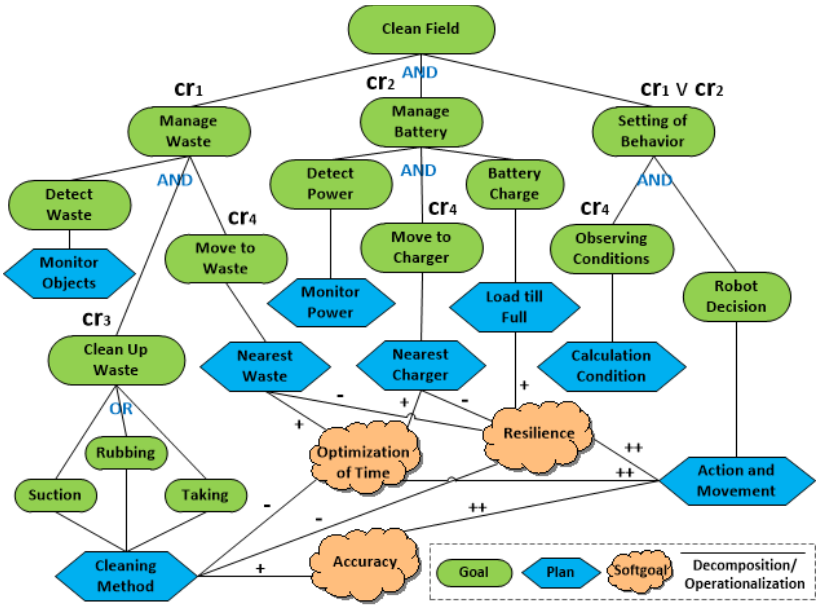

Figure 5. Goal modelling for cleaner robot

Table 3. Object Properties

Object Type	Robot Action	Processing Time (s)	Travel Time (s)
Contain w ater	Rub	2	Time to reach the target
Solid	Take	1	Time to reach the target
Contain dust	Suction	0,5	Time to reach the target
Plug	Battery charging	60	Time to reach the target

Table 4. Battery Properties

Battery Capacity (mAh)	Motor Load (A)	Speed (s)	Travel Time (s)	Distance (s)
1000	1	2	60	120

There are a number of waste objects which should be cleaned so that the robot has a consideration to analyze and plan (AP) "manage waste (cr_1)" or "manage battery (cr_2)", through the components of monitor (M) which are "detect waste" and "detect power". Based on the combination of every property value in Tables 3 and 4, the data collection obtained data by detecting the travel time based on the results of monitoring the presence of an object in an area (shown in Table 5). Data on Table 5 will be the input variables for robot to AP "setting of behavior ($cr_1 \vee cr_2$)", including "observing conditions" components which will determine the power requirements (f(x)) of each waste object with provisions:

$$Capacity\ (f(x)) = \left(\frac{(p_x\ +\ d_x\ )}{e}\right)\%$$

So, we obtained the data as shown in power requirements column on Table 5. Then, the sorting of power requirements from the smallest (as the fitness value) was done. The waste object sequence by the smallest power requirement can be obtained with provisions:

$$Sort\ (\min(f(x)))$$

The setting of robot behavior is associated with some rules that can be set and adjusted to the needs of threshold based on preference (cr_4). For example, cr_2 is raised when the detected power < 20% so that the robot charging the battery = activate(N: $cr_2 \wedge cr_4$) or if power > 20% robot will actively clean up waste = activate(N: $cr_1 \wedge cr_4$). Meanwhile, waste type (cr_3) is raised when waste that should be cleaned is detected so that the robot chooses how to clean. Based on those rules, the system will perform optimum value observation = activate (N: $cr_1 \wedge cr_4$) $\vee$ (N: $cr_2 \wedge cr_4$) by considering cr_2 and cr_3. Robot behavior in responding to waste type (cr_3) is arranged through rules to choose how to clean up the waste in accordance with the waste type encountered or create (instances components) how to clean if the waste encountered is a new type of waste.

Table 5. Data Collection of Waste Object

Object Type	Processing Time (s) = p	Travel Time (s) = d	Battery Duration (s) = e	Power Requirements (%)
Puddle	2	5	60	12%
Leaf	1	7	60	13%
Dust	0,5	18	60	31%
Leaf	1	7	60	13%
Leaf	1	12	60	22%
Leaf	1	15	60	27%
Dust	0,5	7	60	13%
Leaf	1	16	60	28%
Dust	0,5	4	60	8%
Leaf	3	8	60	18%
Leaf	1	4	60	8%
Puddle	2	8	60	17%

Based on modeling in Figure 5, goal decomposition of "clean up waste" is OR-decomposition which shows the presence of variability associated with the selection of cleaning instructions, whether suction, rubbing, taking, assuming waste_type (cr_3) identified contains dust, water, and solid objects. In addition, this event will also be affected by unexpected events or errors (event_error), for example the action failure because of overload (cr_5) on the robot**,** emergence of obstacles (cr_6), sensor damage (cr_n), etc. The problem that can emerge here is that when determining the choice and when waste type encountered by the robot is a new type, the robot must reconfigure the system to provide an alternative behavior. Based on the description, it can obtain the variable of "waste_type" and "event_error" so that the plan can be represented as: *plan(waste_type, event_error)*.

The objective of this plan is to create an alternative behavior to cope with contextual (CR) variability to meet high-level goal and softgoal. For example, the plan of "cleaning method" must use the executes (E) component function "robot decision" based on the components of

"setting of behavior (AP)". It can give full positive contribution (++) to "accuracy", "optimization of time", and "resilience" softgoal, compared to simply analyzing and planning to "manage waste (AP)" which only contributes positively (+). It will be affected by negative contribution (-) when the plan is executed. Thus, the system has a consideration to analyze and plan (AP) to "manage waste" and "manage battery" based on the determination of property values of each parameter. The value of this property becomes input variables of robot to "setting of behavior (AP)". Setting behavior is associated with some rules that can be defined as follows:

Rule-1 : *if (waste_type = contain_dust) and (event_error = null) then plan = service_suction*
Rule-2 : *if (waste_type = contain_water) and (event_error = null) then plan = service_rubbing*
Rule-3 : *if (waste_type = solid) and (event_error = null) then plan = service_taking*
Rule-4 : *if (waste_type = new_type) and (event_error = not null) then plan = create new service_action*
Rule-5 : *if ((waste_type = null) and (event_error = not null) then plan = create new service_action*
Rule-6 : *if (event_error = not null) then plan = change service_action*
Rule-7 : *if not [criteria] then plan = change service_action*

Table 6. ECA-Cleaning Method

Event (E)	Condition (C)	Action (A)
(cr_3)	*(waste_type = contain_dust);* *(waste_type = contain_water);* *(trush_type = solid);* *(event_error = null);*	$P_{1.1}$ = *service_suction* $P_{1.2}$ = *service_rubbing* $P_{1.3}$ = *service_taking*
$(cr_5 \vee cr_6)$	*(waste_type = new_type);* *(waste_type = null);* *(event_error = not null);*	P_2 = *create new service_action [instance component]*
$(cr_5 \vee cr_6)$	*(event_error = not null);* *not [criteria];*	P_3 = *change service_action [instance component]*

Furthermore, the rule is mapped into the concept of ECA as shown in Table 6. "Event" refers to the current state of the robot; "condition" refers to the time when condition changes; “action” refers to under certain circumstances what can be done to adapt, in order to obtain three action of change plans (P_n) as an alternative solution for the adaptation. Based on the indicators of optimum value and the rule set forth in Table 6, the robot can adjust its behavior through the component "robot decision (E)" to execute the adaptation action to clean waste through option "rub (E)", "take (E) "," suction (E) ", or other actions considering the action "battery charge (E)". The illustration of the execution order of the robot system functions can be seen in Figure 6.

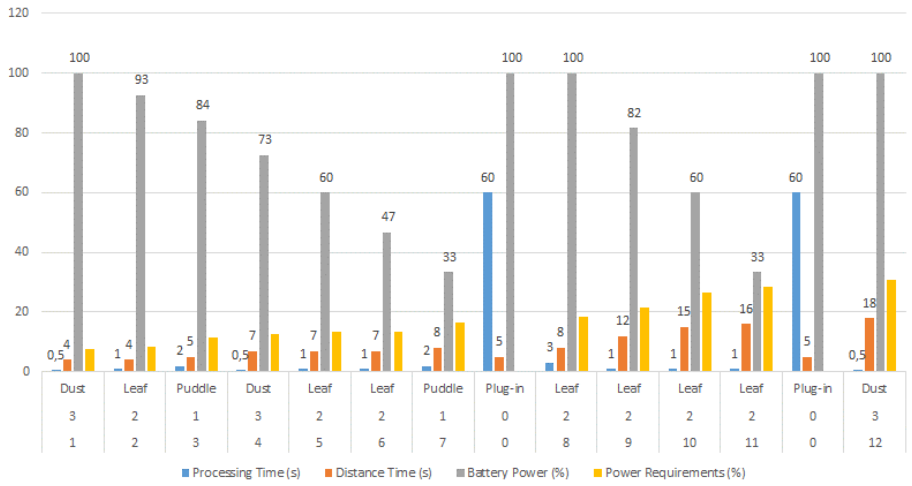

Figure 6. The illustration of the execution order of the cleaner robot system

A=$P_{1.1}$, $P_{1.2}$, $P_{1.3}$ in Table 6 is an adaptation action for monitorable context, while P_2 and P_3 show the dynamic evolutionary needs of CR, ie non-monitorable context for handling domain changes, such as view points, versions, etc. In this case, it adds new components to the software based on new CR that appears, ie cr_5=overload and cr_6=obstacle. This represents the fulfillment of requirements evolution in which the addition of components can be implemented at run-time. For example, the rules for cr_5 and cr_6 are as follows: (a) cr_5 is generated when the capacity is detected >80%, and capacity=0%. So, it is necessary to adjust the robot movement path to the position of bin; (b) cr_6 rearrange the robot movement path when obstacle objects suddenly appear. Adaptation actions for requirements evolution based on cr_5 and cr_6 can be composed as new MAPE-K composite components. Another example of the create new service_action in Table 6 is a new function to dispose waste in "clean up waste" which is influenced by cr_5. This can be added as a primitive component on “manage waste” MAPE-K composite component. The dashed line in Figure 7 shows that the new component is added to the existing component specification.

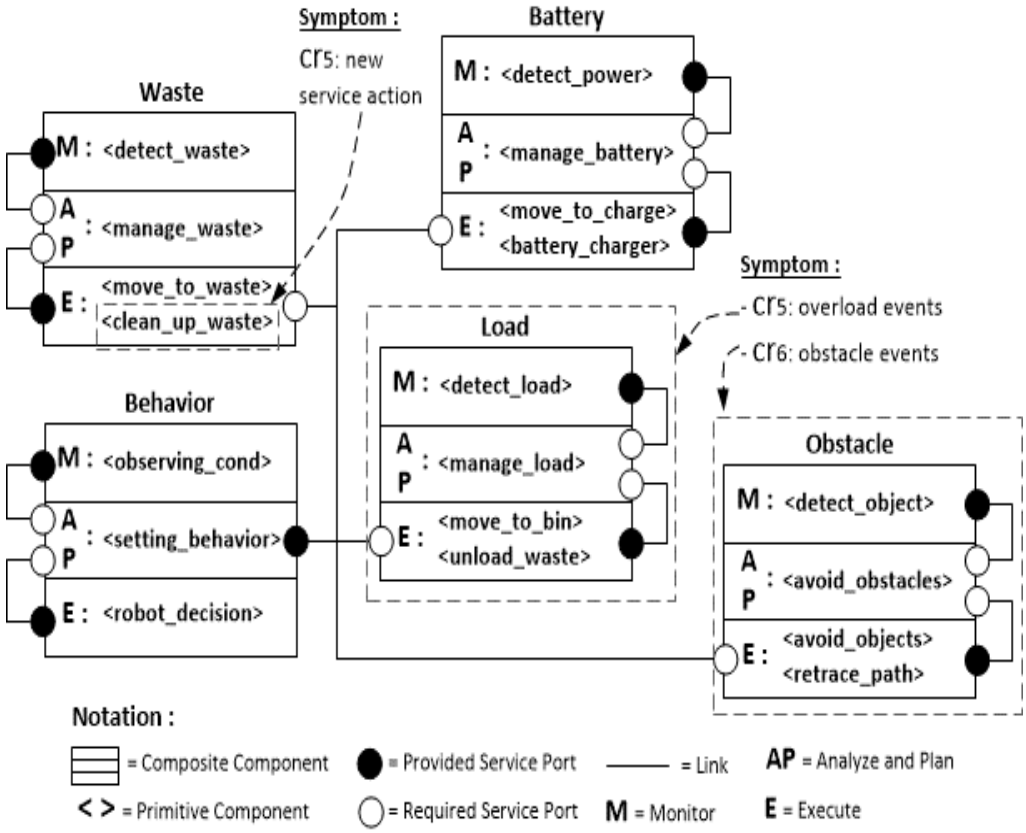

Figure 7. The components specification after the addition of new components

In this experiment, we also measure the scalability process represented by the number and type of waste objects that can continue to grow at run-time, as well as the average size of their execution time. As an experiment, we add a number of new waste objects gradually and randomly up to 50 objects as can be seen in Figure 8. The evaluation results show a linear scale between execution time and object size. It can be concluded that the system is capable of handling change and growth of contextual requirements together based on the number per second linearly. As a future work, we plan to expand the control strategy for knowledge domain related to the improvement of execution time.

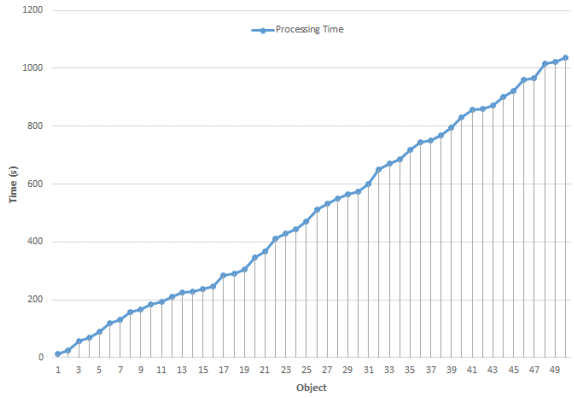

Figure 8. Process scalability of cleaner robot system

4. Discussion: Comparison with Related Work

The approach proposed in this paper is influenced by previous works. The comparison can be seen in Table 7. The researchers adopt and extend various approaches in realizing self-adaptation capabilities. For example, Morandini [16] uses the requirements that are similar to those of our proposed approach of extending i*/ Tropos. Morandini introduces design-time requirements such as goal types, environmental models and failure models, including operational semantics for dependencies and run-time reasoning. However, we add domain assumptions concept created explicitly through contextual requirements which will improve the process of analyzing domain variability.

Table 7. Comparison with Related Work

Model	Requirements Specifications	Design-time	Run-time	Requirements Evolution
Tropos4AS Morandini [16]	Goal, softgoal, plan, resources, relation, environment class	Designing variability: goal type, condition, and failure model	Transition rules: elicitation and recovery for failure	No
Adaptive STS Dalpiaz [25]	Goal, softgoal, plan, resources, domain assumptions, context	Designing contexts: activations rules, time limits, plan, goals	DLV- reasoner: reconcile and compensation for failure	No
ARML Qureshi [26]	Goal, task, quality constraints, domain assumptions, context	Designing ontology: goals, relation, preferences, rules	Inference rules: high-level goal (user) reasoning	No
GASD Wang [27]	Goal/ role model, resource, plan/ restriction, activity	Designing ontology: UML, goal tree, know ledge library	Inference engine: goals selection for failure and alternative behavior	No
SOTA Abeyw ickrama [28]	Goal: pre & post-condition, utilities: actor, entity	Designing utilities, grammar -language, goal to event-based	Model checker labeled transition system analyzer for verification	No
REFAS Fernandez [29]	Goal, softgoal, context, assets, claim, dependency	Designing concern level & operation, goal, soft dependency	Generic meta-model: verification & simulation for adaptation model	No
GODA Mendonça [30]	Goal, softgoal, plan, resources, contextual goal model	Designing contextual & runtime goal, model transformation	Probabilistic model checking for verification solution	No
GOCC Nakagaw a [19]	Goal/ requirement, entity/ object, agent, operation	Designing three-layer architecture for goal & S/W components	Parser engine for new patterns and conflicts, model generation	Yes
ZANSHIN Souza [32]	Goal, task, quality constraints, domain assumptions	Designing aw arness requirements, parameters model	Adaptation framew ork: qualitative adaptation, evolution requirements	Yes
Our Model	Goal, softgoal, plan, resources, domain assumptions, context	Designing contextual requirements, MAPE-K pattern, goal, rules	Inference engine: variability & evolution rules for contexts	Yes

Currently, contextual requirements concept has been used by some researchers, including Dalpiaz [25] linking contexts to variation point in goal model as an architectural model with self-reconfiguration capability for multi-agent behavior. However, requirements evolution is still not covered, similar to Morandini [16]. In this model, requirements are assumed to be unchanged over time. Qureshi [26] also equips i*/ Tropos through contextual requirements concept by mapping goal model into domain ontology using Techne language, but reasoning mechanism for changing domain assumptions, preferences and contexts related to requirements evolution still requires further research. In addition, Wang [27] also applied ontology concept (OWL Ontology) to goal tree in BDI agent. In fact, the reasoning mechanism can be improved. We propose a more flexible rule model so that run-time reasoning can be done automatically. Abeywickrama [28] proposed a goal pattern with a natural language approach; Fernandez [29] proposed a requirements model for self-adaptive systems to address uncertainty through the multi-view framework specification. However, their work did not include requirements evolution.

Mendonça [30] proposes contextual runtime goal models through probabilistic model checking that focuses on providing requirements specification and verification at design-time and run-time. However, they have not addressed the full perspective of self-adaptive software related to mapping into software components. From another perspective, Ying [31] introduced a formalization method for analyzing and concluding evolution for self-adaptive software structures through software components, but this approach still does not include contextual information, such as environmental descriptions as contextual requirements that trigger evolution. In this paper, we try to employ both perspectives, that is implementing contextual requirements and mapping software components. A similar work was done by Nakagawa [19] which has inspired us, but the focus and approach used are different. Our work may be complementary to enrich the feature, for example in terms of component specifications related to domain variability rules and dynamic evolution mechanisms that can be performed automatically based on detected contextual requirements.

In Table 7, Souza [32] also proposes models related to requirements evolution through awarness requirements. Our model works on different perspectives. A context-based adaptation strategy can support context awareness when associated with awarness requirements. In addition, entities of problem domains in domain models have not proposed specific ways to represent it, while we take advantage of goal oriented-requirements engineering through context inheritance integrated into MAPE-K pattern to represent domain variability and have consideration to manage its changes at software architecture levels.

5. Conclusion

The proposed model in this paper highlights the importance of addressing changes and growth in contextual requirements. We introduce requirements modeling language that has been redirected to adaptation patterns through context inheritance hierarchies to bind and manage the activation of MAPE-K model. Adaptation control is manifested through rule editor model with rule based systems approach that can be updated at run-time. The adaptation scenarios are prepared for two issues: first, related to variability rules to deal with changes in context information (domain variability); second, related to evolutionary rules for requirements evolution. Based on the experimental results, the model is able to handle both issues which are shown through case study descriptions. This model still requires further research to improve the aspect of dealing with the refinement of transformation concept between goal model into software component, which enriches the supporting features for broader context inference. In addition, conflict resolution among system components also requires further research to accommodate more complex conflicts. In the near future, we plan to investigate uncertainty issues related to requirements evolution to enrich the model proposed in this paper.

Acknowledgments

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

References

[1] Cheng B.H.C, Rogério de Lemos R, Giese H, Inverardi P, Magee J. *Software Engineering for Self-Adaptive Systems: A Research Roadmap*. LNCS 5525-Springer. 2009; 5525: 1–26.

[2] Rogério de Lemos R, et. al. *Software Engineering for Self-Adaptive Systems : A Second Research Roadmap*. Self-Adaptive Systems. LNCS 7475-Springer. 2013; 7475: 1–32.

[3] Weyns D, Usman Iftikhar M, Malek S, Andersson J. *Claims and Supporting Evidence for Self-Adaptive Systems: A Literature Study*. Proceedings of Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). 2012; 89–98.

[4] Besrour S, Ghani I. Measuring Security in Requirements Engineering. *International Journal of Informatics and Communication Technology (IJ-ICT)*. 2012; 1(2): 72-81.

[5] Sucipto S, Wahono R.S. A Systematic Literature Review of Requirements Engineering for Self-Adaptive Systems. *Journal of Software Engineering*. 2015; 1(1):17–27.

[6] Bencomo N. *Requirements for Self-Adaptation*. Generative and transformational techniques in software engineering IV. LNCS 7680-Springer Berlin. 2013; 7680: 271–296.

[7] Abeywickrama D.B, Ovaska E. A Survey of Autonomic Computing Methods in Digital Service Ecosystems. *Service Oriented Computing and Applications*. 2017; 11(1): 1-31.

[8] Knauss A, Damian D, Franch X, Rook A, Müller H.A, Thomo A. ACon: A Learning-Based Approach to Deal With Uncertainty in Contextual Requirements at Runtime. *Information and Software Technology*. Elsevier B.V. 2016; 70(2016): 85-99.

[9] Jureta I. The Design of Requirements Modelling Languages. How to Make Formalisms for Problem Solving in Requirements Engineering. Edition Number 1. XII 286. Springer. 2015.

[10] Surendro K, Martini C. Hierarchical i* Modeling in Requirement Engineering. *TELKOMNIKA (Telecommunication, Computing, Electronics and Control)*. 2016; 14(2): 784-790.

[11] Lapouchnian A, Mylopoulos J. *Capturing Contextual Variability in i* Models*, CEUR Proceedings of the 5th International i* Workshop (iStar 2011). 2011; 96–101.

[12] Yu E, Giorgini P, Maiden N, Mylopoulos J. Social Modeling for Requirements Engineering. Cambridge Massachusetts-London: The MIT Press. 2011.

[13] Bresciani P, Perini A, Giorgini P, Giunchiglia F, Mylopoulos J. Tropos: An agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-Agent Systems*. 2004; 8(3): 203–236.

[14] Mulyani H, Djatna T, Sitanggang I.S. Agent Based Modeling on Dynamic Spreading Dengue Fever Epidemic. *TELKOMNIKA (Telecommunication, Computing, Electronics and Control)*. 2017; 15(3): 1380-1388.

[15] Aradea, Supriana I, Surendro K. *An Overview of Multi Agent System Approach in Knowledge Management Model*. International Conference on Information Technology Systems and Innovation (ICITSI). Bandung - Indonesia. 2014; 62-69.

[16] Morandini M, Penserini L, Perini A, Marchetto A. Engineering Requirements for Adaptive Systems. *Requirements Engineering*. Springer London. 2017; 22(1): 77–103.

[17] Aradea, Supriana I, Surendro K, Darmawan I. *Variety of Approaches In Self-Adaptation Requirements: A Case Study*. Recent Advances on Soft Computing and Data Mining, Advances in Intelligent Systems and Computing. Springer. 2017; 549: 253-262.

[18] Aradea, Supriana I, Surendro K, Darmawan I. *Integration of Self-Adaptation Approach on Requirements Modeling*. Recent Advances on Soft Computing and Data Mining. Advances in Intelligent Systems and Computing. Springer. 2017; 549: 233-243.

[19] Nakagawa H, Ohsuga A, Honiden S. *Towards Dynamic Evolution of Self-Adaptive Systems Based on Dynamic Updating of Control Loops*. International Conference on Self-Adaptive and Self-Organizing Systems. Lyon, France. IEEE 2012; 59-68.

[20] Abuseta Y, Swesi K. Design Patterns for Self Adaptive Systems Engineering. *International Journal of Software Engineering & Applications (IJSEA)*. 2015; 6(4): 11-28.

[21] Weyns D. et al. *On Patterns for Decentralized Control in Self-Adaptive Systems*. In R. de Lemos, H. Giese, H. A. Muller, and M. Shaw, editors. Software Engineering for Self-Adaptive Systems. LNCS-Springer. 2012; 7475: 76–107.

[22] Kephart J.O, Chess D.M. The Vision of Autonomic Computing. *IEEE Computer*. 2003; 36(1): 41–50.

[23] Inverardi P, Mori M. *Requirements Models at Run-time to Support Consistent System Evolutions*. International Workshop on Requirements@Run.Time. 2011; 1–8. IEEE.

[24] Hirsch D, Kramer J, Magee J, Uchitel S. *Modes for software architectures*. European Workshop on Software Architecture. LNCS. 2006; 4344: 113–126.

[25] Dalpiaz F, Giorgini P, Mylopoulos J. Adaptive socio-technical systems: a requirements-based approach. *Requirements Engineering*. Springer. 2013; 18(1):1–24.

[26] Qureshi N.A, Jureta I.J, Perini A. Towards a Requirements Modeling Language for Self-Adaptive Systems. LNCS in Springer. REFSQ. 2012; 7195: 263-279.

[27] Wang T, Li B, Zhao L, Zhang X. *A Goal-Driven Self-Adaptive Software System Design Framework Based on Agent*. Physics Procedia. ICAPIE. 2012; 24: 2010-2016. Elsevier B.V.

[28] Abeywickrama D.B, Zambonelli F. *Model Checking Goal-Oriented Requirements for Self-Adaptive Systems*. 19th IEEE International Conference and Workshops on Engineering of Computer-Based Systems. 2012; 33-42.
[29] Munoz-Fernandez J.C. Towards a Requirements Specication Multi-View Framework for Self-Adaptive Systems. *CLEI Electronic Journal*. 2015; 18(2): 5.
[30] Mendonça D.F, Rodrigues G.N, Alves V, Ali R, Baresi L. GODA: A Goal-Oriented Requirements Engineering Framework for Runtime Dependability Analysis. *Information and Software Technology*. Elsevier B.V. 2016; 80(2016): 245-264.
[31] Ying W, Howard M. Software Running Dynamic Reconfiguration Model Based on B Diagram. *TELKOMNIKA (Telecommunication, Computing, Electronics and Control)*. 2016; 14(2A): 101-107.
[32] Souza V.E.S, Lapouchnian A, Angelopoulos K, Mylopoulos J. Requirements-Driven Software Evolution. *Computer Science - Research and Development*. Springer. 2013; 28(4):311-329.