Journal of Ambient Intelligence and Humanized Computing
https://doi.org/10.1007/s12652-019-01367-2

ORIGINAL RESEARCH

A Nash equilibrium based decision-making method for internet of things

Euijong Lee[1] · Young-Duk Seo[2] · Young-Gab Kim[1]

Received: 27 September 2018 / Accepted: 12 June 2019

Abstract
In recent years, the Internet of Things (IoT) has gained global popularity. IoT can connect several objects and create a dynamic environment; thus, an IoT system must adapt to environmental changes. To adapt to a dynamic environment, an effective decision-making method is required for an IoT system. Game theory is a mathematical method for decision-making among decision makers, and it may be applied to decision-making for an IoT system. In addition, the concept of self-adaptive software may be applied to IoT because such software evaluates and changes its behavior to satisfy its intended use, and the IoT system then makes decisions and adapts to its dynamic environment. In this study, a decision-making method is proposed along with game theoretic decision-making and self-adaptive loop mechanisms for IoT. The proposed method is based on MAPE-K loops, which are general processes used in self-adaptive software with shared knowledge. In addition, Nash equilibrium is applied to extract candidate strategies, which are evaluated for selecting the most optimal solution. The proposed method was implemented as a prototype, and experiments were conducted to evaluate its runtime performance. The results show that the proposed method can be applied to an IoT environment at runtime.

Keywords Self-adaptive software · Game theory · Nash equilibrium · Internet of Things

1 Introduction

In recent years, Internet of Things (IoT) has become increasingly sophisticated and widespread, and it is applied in various fields [e.g., smart home (Jo and Yoon 2018; Park et al. 2019), smart city (Jeong and Park 2019; Choi and Ahn 2019), healthcare (Azimi et al. 2017), middleware (Ouechtati et al. 2018), and augmented reality (Jo and Kim 2019)]. IoT is a technology that connects objects using a wireless network. Therefore, IoT can connect several objects together to create a dynamic environment (Kaur and Kaur 2017; Rayes and Samer 2017). In addition, an IoT environment is changed through its surrounding environment. Therefore, an IoT system must dynamically satisfy its requirements and adapt to changes in the environment at runtime. Therefore, an IoT system needs a decision-making method for adapting to dynamic environmental changes (Balasubramaniam and Jagannath 2015; Mohammadi et al. 2019). Game theory is a mathematical theory, and is used for decision-making between rational decision-makers (Nisan et al. 2007; Straffin 1993). Game theory has influenced various fields, including economics, biology, politic science, and psychology. In addition, it was applied to computer science, artificial intelligence, networking, and decision-making algorithms (Shoham 2008; Algur and Kumar 2013; Kumari and Chakravarthy 2016). Therefore, game theory may be applied to decision-making for IoT. In addition, to adapt to a dynamic environment, self-adaptive software may be applied to IoT (Hughes 2018). Self-adaptive software detects environmental conditions, and changes its behavior or structure if the software requirements are violated (Salehie and Tahvildari 2009). In this study, we propose a game theoretic decision-making method using a self-adaptive concept for IoT. The proposed method consists of a self-adaptive loop and a game theoretic decision-making method.

✉ Young-Gab Kim
alwaysgabi@sejong.ac.kr

Euijong Lee
kongjjagae@sejong.ac.kr

Young-Duk Seo
mysid88@sejong.ac.kr

1 Department of Computer and Information Security, Sejong University, Seoul, Republic of Korea

2 Department of Data Science, Sejong University, Seoul, Republic of Korea

The reminder of this paper is organized as follows. Section 2 describes related studies, which are self-adaptive software and game theory. Section 3 introduces the game theoretic decision-making method for IoT. In Sect. 4, the empirical experiment is described. Section 5 provides some concluding remarks regarding this research.

2 Related work

In this section, self-adaptive software and Nash equilibrium are introduced. In addition, several studies that applied game theory to software are described.

2.1 Self-adaptive software

Self-adaptive software adapts to its environment by changing its behavior or structure at runtime (Salehie and Tahvildari 2009). One definition of self-adaptive software was defined by the Defense Advanced Research Projects Agency (Laddaga and Robertson 2004) as follow:

"Self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible".

To achieve self-adaptation, various methods and perspectives have been studied (Knauss et al. 2016; Lee and Baik 2015; Lee et al. 2016; Kim et al. 2017; Lee et al. 2017; Lee et al. 2018; Filieri et al. 2011; Filieri and Tamburrelli 2013; Filieri et al. 2016; Tallabaci and Souza 2013; Garlan et al. 2004; Zhang et al. 2017). Various techniques have been applied to self-adaptive software, for example, machine learning (Knauss et al. 2016), model-checking (Lee et al. 2017; Lee et al. 2018; Lee and Baik 2015; Filieri et al. 2011; Filieri and Tamburrelli 2013; Filieri et al. 2016), goal-based modelling (Tallabaci and Souza 2013), data mining (Knauss et al. 2016), architecture (Garlan et al. 2004) and ontology (Seo et al. 2018) based methods. Some studies have applied self-adaptive software to IoT with various aspects (Muccini et al. 2018; Azimi et al. 2017; Ribeiro et al. 2016; Sylla et al. 2017; Henry et al. 2018). In addition, previous studies have their own aspects and distinctive characteristics. However, many self-adaptive software studies have a common feature, namely, a loop mechanism (Salehie and Tahvildari 2009; Tallabaci and Souza 2013; Lee et al. 2017; Lee et al. 2018; Lee and Baik 2015; Seo et al. 2018; Kim et al. 2017; Ouechtati et al. 2018).

To accomplish self-adaption, a loop mechanism was proposed (Kephart and Chess 2003; Salehie and Tahvildari 2009) and implemented in several types of autonomic computing and self-adaptive software. This loop is called a MAPE loop, and consists of four processes: The monitoring process is responsible for collecting data from sensors and internal software changes. The detection (analysis) process is responsible for analyzing the symptoms related with changes to a situation using the monitored data. In addition, this process detects when an adaptation is required (i.e., when a requirement is violated). The decision (planning) process is responsible for decision-making for adaptation. Therefore, the decision process generates strategies and selects the optimal solution to achieve the best outcome. The action (execution) process is responsible for executing the optimal solution. A MAPE loop with a shared knowledge base is called a MAPE-K loop (Kephart and Chess 2003). A MAPE loop and a MAPE-K loop can be applied in IoT with various patterns (Muccini et al. 2018). In this study, the concept of self-adaptive software is applied for decision-making at runtime, and thus a MAPE-K loop is applied, the details of which are described in Sect. 3.1.

2.2 Game theory and software

Game theory is a mathematical theory that facilitates decision-making. It has influenced several fields, such as political science, biology, economics, and psychology (Nisan et al. 2007; Straffin 1993). In addition, game theory can be applied in computer science to facilitate decision-making in algorithms, artificial intelligence, and networks (Shoham 2008). Several studies have applied game theory in the software field. Bhatia and Sood (2017) applied game theory to detect an information overflow with the potential to compromise national security. Kumari and Chakravarthy (2016) used a cooperative game to achieve data privacy, and called their proposed method cooperative privacy (CoP) Azzedin and Yahaya (2016) used game theory for free riding in file-sharing software (i.e., BitTorrent). In addition, game theory was applied to the field of networking. Tao et al. (2017) proposed a game-theoretic model for a behavior analysis of IoT protocols. Kutsuna and Fujita (2011) proposed a congestion control scheme for high-speed networks, and a minority game was applied to avoid network congestion. Algur and Kumar (2013) suggested a game theoretic bandwidth allocation algorithm for IEEE 302.16e. Semasinghe et al. (2017) suggested a game-theoretic model for wireless IoT network resource management. Zheng et al. (2015) proposed a graphical game for formulating the problem of energy saving in a green cellular network environment.

2.3 Nash equilibrium

In this section, Nash equilibrium applied to the proposed method is introduced. Nash equilibrium was introduced by John Forbes Nash, Jr., and can provide forecasts when there are non-cooperative players in a game (Nash 1950, 1951). From this perspective, Nash equilibrium is used to analyze a solution among several decision makers (Nisan et al. 2007;

Straffin 1993; Shoham 2008). With Nash equilibrium, there are players participating in a game who can act independently. Such actions are called strategies, and the strategies of one player affect those of the other players. When the players select a strategy, the payoff is determined based on the current set of selected strategies. Each player will choose a strategy to achieve an outcome with the highest payoff as possible. Base on this principle, Nash equilibrium means that players cannot select a better unilateral strategy, because any player can receive a better payoff by changing their own strategy. In other words, a situation of solidified strategies occurs, and thus the players cannot change their own strategy. Nash equilibrium can also be described as follows:

Let (S, f) be a game with n players, where

- $S = S_1 \times S_2 \times \ldots \times S_n$ is the strategy set of a profile;
- $Player_i \in \{1, \ldots, n\}$;
- $f(x) = \{f_1(x), \ldots, f_n(x)\}$ is the payoff function;
- a payoff function is evaluated at $x \in S$;
- x_i is the strategy profile of player i;
- x_{-i} is the strategy profile of the other players;
- player i selects strategy x_i resulting in strategy profile $x = (x_1 \cdots x_n)$, then player i obtains payoff $f_i(x)$; and
- $x^* \in S$ is a Nash equilibrium when $\forall i, x_i \in S_i$, namely, $f_i(x_i^*, x_{-i}^*) \geq f_i(x_i, x_{-i}^*)$.

By definition, any player under Nash equilibrium can select a better unilateral strategy because all players can receive a better payoff by changing their strategy. In other words, the set of strategies in Nash equilibrium are the best solutions to a game. However, if multiple Nash equilibriums occur, an additional rule is needed to select the optimum strategy. In this study, the concept of Nash equilibrium is applied for decision-making among several requirements of an IoT environment. Details regarding the use of Nash equilibrium are described in Sect. 3.2.

3 Nash equilibrium based decision-making method

In this study, a game theoretic decision-making method is proposed for IoT, which consists of four processes based on a MAPE-loop. In Sect. 3.1, an overview of the proposed method is provided. Sect. 3.2 describes the game theoretic strategy extraction method using Nash equilibrium, as well as the strategy evaluation method. Algorithms for the proposed approach are described in Sect. 3.3.

3.1 Overview

As previously mentioned, the proposed method uses the concept of self-adaptive software, which consists of an adaptation processes called a MAPE-K loop (Salehie and Tahvildari 2009). The adaptation consists of four processes, namely, monitoring, analysis (detection), planning (decision), and execution (action) with a shared knowledge base. An adaptation loop is used in the proposed method, and Fig. 1 shows an overview.

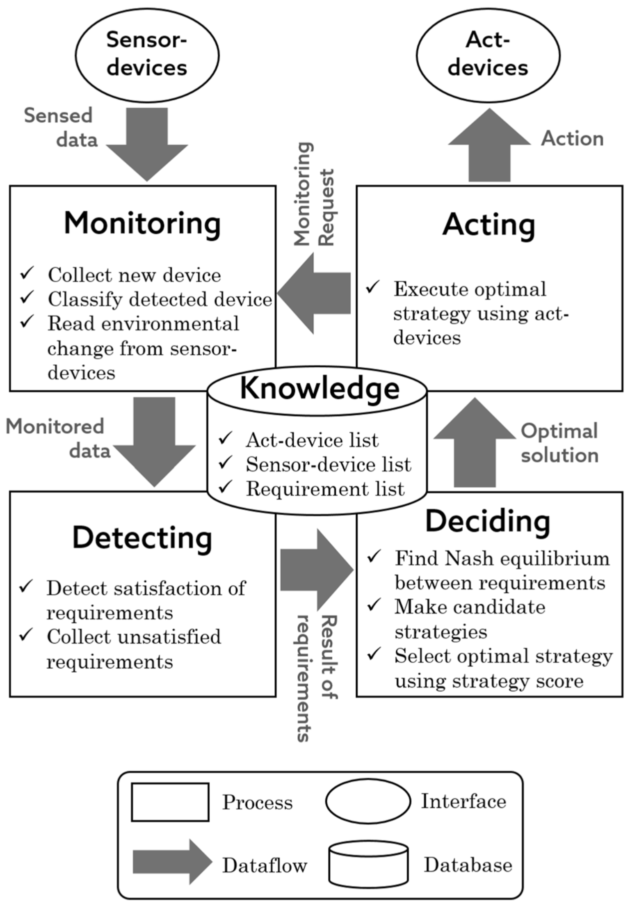
Sensor-devices
Act-devices
Sensed data
Action
Monitoring Request
Monitoring
✓ Collect new device
✓ Classify detected device
✓ Read environmental change from sensor-devices
Acting
✓ Execute optimal strategy using act-devices
Knowledge
✓ Act-device list
✓ Sensor-device list
✓ Requirement list
Monitored data
Optimal solution
Detecting
✓ Detect satisfaction of requirements
✓ Collect unsatisfied requirements
Deciding
✓ Find Nash equilibrium between requirements
✓ Make candidate strategies
✓ Select optimal strategy using strategy score
Result of requirements
Process
Interface
Dataflow
Database

Fig. 1 Overview of proposed method

As described in Fig. 1, there are interfaces related with the classification of devices. In this study, IoT devices are classified into only two types: act-devices and sensor-devices. Although it is possible to classify IoT devices in a more detailed manner, only two classifications are used herein for clarity and simplicity. Act-devices are used to change the environment, and thus should execute a physical operation. Therefore, act-devices have a physical device embedded (e.g., an LED, a fan, or a humidifier). A sensor-device is used to detect environmental changes, and thus should detect physical changes. Therefore, sensor-devices have readable devices embedded (e.g., a humidify, temperature, dust density, or light sensor). In addition, it is assumed that both device types can recognize which requirements are related with a particular operation. A database containing shared knowledge is used. The shared knowledge is a list of act-devices, and sensor-devices and their requirements. In addition, every element on the list contains the relationship with other devices and requirements. As shown in Fig. 1 knowledge is shared in the adaptation loop.

In the adaptation loop, the first process is monitoring. The monitoring process is responsible for searching out new addable devices and collecting changes to the environment. The process searches for new devices, and if there is a device with the potentially to be added, the process adds the new device to the shared knowledge. After searching for a new device, the monitoring process classifies the detected device based on the related requirements and the device type. In addition, the process collects environmental changes from the sensor-devices. The monitored data are transferred to the detection process. The detection process detects the satisfaction of the requirements using the monitored data. Results that satisfy the requirements are transferred to the decision process. The decision process is responsible for extracting candidate strategies and selecting an optimal strategy. A Nash equilibrium based method makes candidate strategies that contain actions of the act-devices to adapt to environmental changes. The candidate strategies are evaluated for the selection of an optimal solution. Details on the strategy extraction and evaluation methods are described in Sect. 3.2. After the evaluation, the optimal strategy is transferred to the action process. The action process is responsible for executing the optimal strategy. As mentioned previously, a strategy contains actions of act-device to adapt to environmental changes for satisfying the requirements. Therefore, the action process executes act-devices using the optimal solution for adaptation. Subsequently, the monitoring process is executed after the action process, and the MAPE-K loop is continued.

3.2 Decision-making method based on Nash equilibrium

This section describes the proposed game theoretic decision-making and evaluation method. As stated before, a requirement can have multiple act-devices; for example, if the intensity of the illumination is a requirement, act-devices such as windows or lamps may be used. In addition, an act-device can affect multiple requirements; for example, if the windows are act-devices and are opened to adjust the light intensity, the intensity of the illumination may be adjusted, which may also affect the humidity, temperature, or other requirements. From a game theoretic view, it can be assumed that if certain requirements are violated and need to be adapted, this type of situation can be considered a game. In addition, the requirements are thought of as the players, and the actions of the act-devices are the strategies. Therefore, multiple requirements (players) should effectively operate act-devices (select their strategies) to satisfy their objectives. As previously mentioned, Nash equilibrium is used to extract candidate strategies, which can be described as follows:

Let a player be a requirement, and let (S, f) be a game with n requirements, where

- $S = S_1 \times S_2 \times \cdots \times S_n$ is the strategy set of profile;
- requirement $i \in \{1, \dots, n\}$;
- $f(x) = \{f_1(x), \dots, f_n(x)\}$ is the payoff function;
- a payoff function is evaluated at $x \in S$;
- x_i is an act-device profile of requirement i;
- x_{-i} is an act-device profile of the other requirements;
- requirement i operates act-device x_i resulting in strategy profile $x = (x_1 \cdots x_n)$, and requirement i obtains payoff $f_i(x)$;
- $x^* \in S$ is a Nash equilibrium for IoT when $\forall i, x_i \in S_i : f_i(x_i^*, x_{-i}^*) \geq f_i(x_i, x_{-i}^*)$;
- x^* can be an operation candidate at runtime; and
- a strategy with the highest Nash equilibrium value among the requirements is selected and implemented.

Formally, players cannot select an optimal solution when they reach the Nash equilibrium because the equilibrium allows any players to receive a better payoff by changing their strategy. Therefore, a Nash equilibrium situation denotes a possible operation that satisfies multiple requirements, and it may be a candidate solution for adaptation. However, if Nash equilibrium is a candidate solution, it is necessary to evaluate it to select the most optimal solution.

For a solution evaluation, three conditions are considered: the number of act-devices, the number of affected requirements, and the number of satisfied requirements. The details of this are described below.

- *The number of act-devices (ADs)* This is the number of act-devices (strategies) required to execute a solution. A smaller value is preferable because, if a solution can satisfy multiple requirements with a smaller number of act-devices as compared to other solutions, it may be more efficient.
- *The number of affected requirement (ARs)* This is the number of requirements that can be affected by the execution of act-devices for a solution. A smaller value is preferable because, if the execution of act-devices affects a larger number of requirements, a potential requirement violation may occur.
- *The number of satisfied requirement (SRs)* This is the number of requirements that can be satisfied when executing a solution. A larger value is preferable because, if a solution satisfies more requirements than other solutions, the solution is considered more efficient than the other solutions.

Equation 1 presents the evaluation of solution(ES) using AD, AR and SR.

$$ES = \alpha\left\{\log\left(\frac{SR+1}{AR+1}+1\right)\right\} + \beta\left\{\log\left(\frac{1}{AD+1}+1\right)\right\}. \tag{1}$$

In the equation, there are two terms: requirement related terms and act-device related terms. The requirement related term is $\left(\text{i.e.,}\left\{\log\left(\frac{SR+1}{AR+1}+1\right)\right\}\right)$ and this term denotes the contribution of a requirement satisfaction. AR and SR are used for this term. As mentioned earlier, a smaller AR and larger SR are more efficient, and thus SR is divided by AR. A value of 1 is added to the denominator and numerator to prevent dividing by zero. For normalization, a logarithmic function is assumed, and 1 is added to prevent a negative infinity. The second term $\left(\text{i.e.,}\left\{\log\left(\frac{1}{AD+1}+1\right)\right\}\right)$ is related with the execution of act-devices. As mentioned earlier, a smaller value is preferable, and thus a reciprocal number of ADs are used. A logarithm function is assumed for normalization, and a value of 1 is added to prevent an infinite output.

The coefficients α and β are mediators for adjusting the power of the requirement and the act-device terms, and the sum of both mediators is 1. The results of the equation denote the strategy evaluation of a solution, and a solution with the largest number can be an optimal solution.

3.3 Algorithms for Nash equilibrium based decision-making method

In this section, the algorithms of the proposed approach are described. Algorithm 1 shows a strategy extraction using Nash equilibrium. Input data are the requirement and act-device array, which is an element of shared knowledge, and the output data are the optimal strategy for adapting to the environment. First, an array is initialized for saving the candidate strategies (line 1). The algorithm consists of three loops (lines 2–27, lines 5–25, and lines 10–23). The first loop repeats for each requirement. In the first loop, the requirement variable and strategy array list are initialized for saving temporary values (lines 3 and 4). In addition, the second loop is embedded, and repeats for each act-device that is related to a requirement (lines 5–25). In the third loop (lines 10–23), there is switch/case statement for checking the relationship between a requirement and a strategy with an act-device (lines 12–22). Details of the algorithm for checking the relationship are described in Algorithm 2. In the switch/case statements, if Nash equilibrium exists among a requirement, a strategy, and an act-device, this status is added to the temporal strategy array (lines 13–15). As mentioned earlier, the Nash equilibrium has the possibility of satisfying multiple requirements; thus, it can indicate a strategy. If there is no Nash equilibrium, and there are no conflicting requirements, it can also indicate a strategy (lines 16–18) because the current state has the possibility of having Nash equilibrium. In addition, other cases are deleted from the strategy array (lines 19–21). In other words, if there is a requirement that is unsatisfied by executing an act-device, it may create a conflict among the requirements. After the first loop is completed, the most optimal strategy is returned (line 28). The optimal solution is evaluated based on the strategy score.

Algorithm 1: Strategy extraction algorithm using Nash equilibrium

```
   Data: Requirement array, act-device array
   Result: Adaptive(optimal) strategy
1  candidateStrategyArr = null;
2  while requirementArr.hasNext() do
3      requirement = requirmentArr.now();
4      tempStrategyArr = null;
5      while requirement.actDevice.hasNext() do
6          actDevice = requirement.actDevice.now();
7          if candidateStrategyArr = null then
8              tempStrategyArr.add(requirement,
                 actDevice);
9          else
10             while candidateStrategyArr.hasNext()
                 do
11                 candiStrategy =
                     candidateStrategyArr.now();
12                 switch relationCheck(requirement,
                     candiStrategy, actDevice) do
13                     case NashEquilibirum do
14                         tempStrategy.add(candiStrategy,
                             requirement, actDevice);
15                     end
16                     case NotConflict do
17                         tempStrategy.add(candiStrategy,
                             requirement, actDevice);
18                     end
19                     case Conflict do
20                         candidateStrategyArr.now.delete();
21                     end
22                 end
23             end
24         end
25     end
26     candidateStrategyArr = tempStrategyArr;
27 end
28 return candidtateStrategyArr.getOptimal();
```

Algorithm 2 is used for extracting the relationship among a requirement, strategy, and act-device (line 12 in Algorithm 1). In Algorithm 2, the input data are the requirement, strategy, and an act-device, and the result is the relationship of the input data (i.e., Nash equilibrium, conflict, and no conflict). First, an integer value is initialized, and the value saves the number of requirements that are affected by the execution of the act-device. If the execution of the act-device causes a violation of the input requirement, a message-containing conflict is returned (line 14). Otherwise, the loop repeats for each requirement of the input strategy (lines 3–8). In the loop, if a requirement from a strategy is violated through the execution

Table 1 Details of the hardware environments

Hardware	CPU clock (GHz)	CPU core	RAM (GB)	OS
Laptop (Intel i5-5200U)	2.7	2	8	Windows 10
Desktop (Intel i5-4670)	3.4	4	16	Windows 10
Server (Intel Xeon E3- 1230L v3)	1.8	4	4	Windows 10
Samsung Galaxy S8	2.31	8	4	Android 8.0.0

of an act-device, a conflict message is immediately returned (lines 4 and 5). This occurs because, if there is an unsatisfied requirement, Nash equilibrium cannot occur by definition. However, if there is a requirement that is affected by the execution of an act-device, and the execution may satisfy the requirement, the number of affected requirements is increased (line 7). The end of the loop occurs when all requirements of the strategy are satisfied. After the loop, if the value containing the number of affected requirements by an act-device is greater than 1, it means Nash equilibrium occurs among the inputted data. Therefore, Nash equilibrium is returned (lines 1–10). In addition, if there are no affected requirements from the execution of an act-device, it means that the combination of inputted data is a candidate for Nash equilibrium. Therefore, a message containing no conflicts is returned (lines 11 and 12). The algorithm was implemented and tested using Java. Details of the experiment are described in Sect. 4.

Algorithm 2: Relationship extraction algorithm among a requirement, a strategy and an act-device

```
Data: A requirement, a strategy and an act-device
Result: Relationship of input data
1  numberOfAffectedReq = 0;
2  if requirement is satisfied with the act-device then
3      while strategy.hasNextRequirement() do
4          if the requirement from strategy is
             unsatisfied with the act-device) then
5              return Conflict;
6          else if the requirement from strategy is
             satisfied and affected with the act-device
             then
7              numberOfAffectedReq++;
8      end
9      if numberOfAffectedReq ≥ 1 then
10         return isNashEquilibrium
11     else
12         return NotConflict;
13 else
14     return Conflict;
15 end
```

4 Empirical evaluation

A prototype of the proposed framework was implemented using JAVA 1.8 to ensure compatibility with various devices. Details of the test devices are described in Table 1. The experiment was conducted in various environments with different hardware for evaluating the performance. For the evaluation, IoT environments were randomly generated with different numbers of act-devices and requirements. In the test set, a requirement has at least one sensor-device and one act-device. In addition, the remaining act-devices are randomly assigned to the requirements, and environment values are randomly generated for diverseness. Each experiment is iterated 100 times, and the time required for the decision-making is measured.

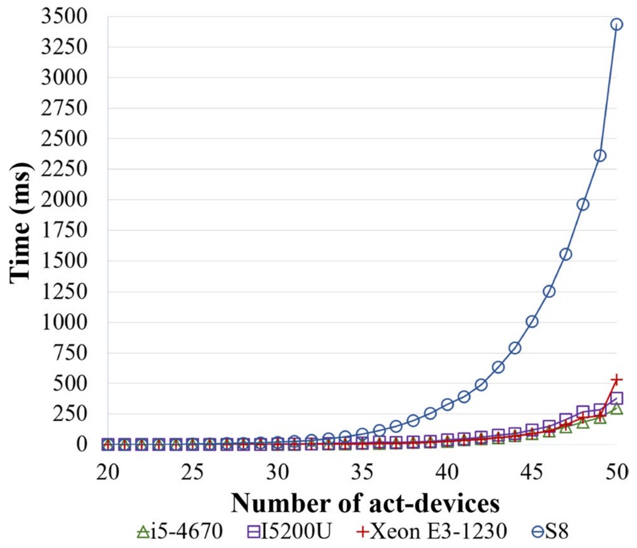

Fig. 2 Results of strategy extraction time with increasing act-devices

The first experiment consisted of ten fixed requirements and variable act-devices (i.e., 20–50). Figure 2 shows the results; because the proposed method requires more time to make an optimal solution, the time increased with a large number of act-devices. In addition, if there are a large number of act-devices, a large number of Nash equilibriums may exist. Therefore, it takes more time to extract equilibriums with a large number of act-devices. The maximum average time of the decision-making was less than 3.5 s for mobile devices, but less than 500 ms for a high computing environment even with 50 act-devices. However, the result shows that the decision-making was calculated within a reasonable amount of time with high and low computing environments.

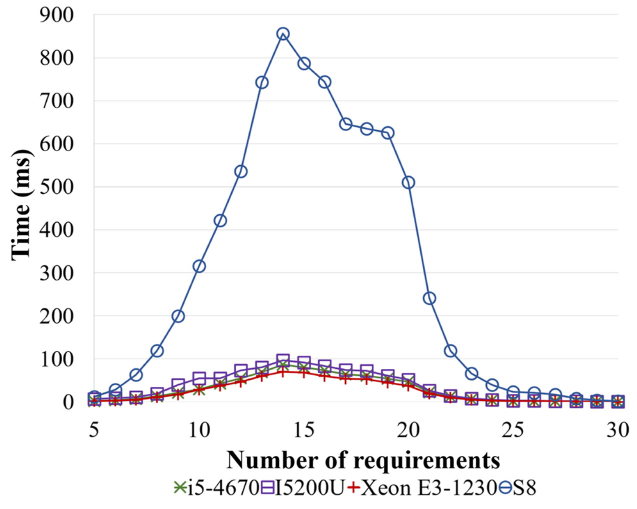

Fig. 3 Results of strategy extraction time with increasing requirements

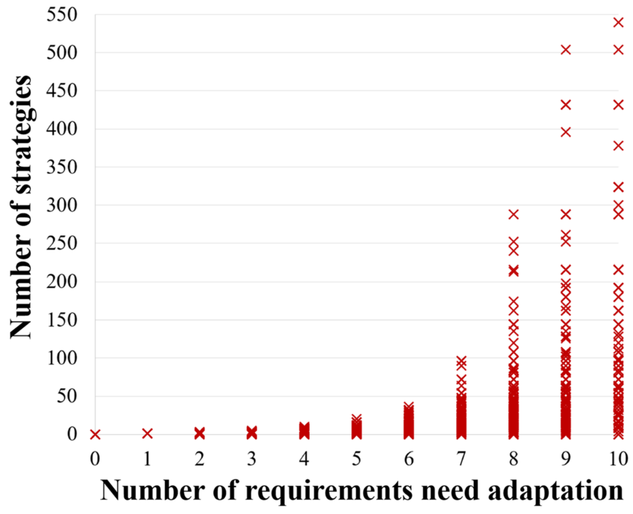

Fig. 4 Results of strategy extraction

The second experiment is similar with the first, but consisted of 40 fixed act-devices and variable requirements (i.e., 5–30). As the results in Fig. 3 show, there was a tendency to increase or decrease with an increasing number of requirements, the reason for which is the complexity of interconnections among the requirements. To extract the Nash equilibrium, the possible strategies of a requirement is checked against the other requirements, and thus, if an interconnection among the requirements is complex, more time to extract the equilibriums is required than with a simple interconnection. Therefore, the results show that the decision-making is affected more by the complexity than by the number of requirements. However, the second experiment results also show a reasonable amount of time even with a complex interconnection of requirements and a low computing environment.

The third experiment was conducted using ten requirements, and 15 act-devices randomly assigned to two requirements. The experiment was iterated 10,000 times using randomly generated situations. The purpose of the experiment was to investigate the number of strategies that can satisfy the requirements. Figure 4 shows the number of strategies extracted in randomly generated situations. In this experiment, there were ten requirements, and thus the number of requirements in need of adaptation was distributed from zero to ten. It can be seen that the number of strategies increased as the number of requirements increased. The extracted strategies were used to satisfy all requirements. It is possible that there were no strategies satisfying all requirements. However, the third experiment shows that the proposed approach can extract strategies under dynamic situations.

5 Conclusion

A game theoretic decision-making method using a MAPE-K loop was proposed in this study. The method consists of four processes: monitoring, action, detection and decision processes. The monitoring process collects new devices to construct an IoT environment, and reads the sensor-devices to detect environmental changes. The monitored data are transferred to the detection process, which is used to detect a requirement violation. The decision process receives violations of the requirements and generates a candidate strategy using the game-theoretic decision-making method. The candidate strategies are evaluated using an evaluation method, and the most optimal solution is selected for adapting to environmental changes. The last process is the action process, which executes the optimal solution from the decision process. An empirical evaluation was conducted to evaluate the proposed approach, and the results show that the proposed framework can be used to extract candidate strategies and the optimal solution within a reasonable amount of time. In addition, the results show that the proposed method can be applied at runtime.

In the future, The proposed approach will be extended for application in physical IoT environments. Especially, the proposed approach will be applied in smart home and smart greenhouse. A modeling method that automatically generates data from a sensor for automation of an IoT system will be studied. In addition, a reinforcement learning based decision-making method will be conducted.

Acknowledgements This work was supported by Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No. 2016-0-00498, User behavior pattern analysis based authentication and anomaly detection within the system using deep learning techniques)

References

Algur SP, Kumar NP (2013) Novel user centric, game theory based bandwidth allocation mechanism in wimax. Hum Centric Compu Inf Sci 3(1):20

Azimi I, Anzanpour A, Rahmani AM, Pahikkala T, Levorato M, Liljeberg P, Dutt N (2017) Hich: Hierarchical fog-assisted computing architecture for healthcare iot. ACM Trans Embedded Comput Syst (TECS) 16(5s):174

Azzedin F, Yahaya M (2016) Modeling bittorrent choking algorithm using game theory. Future Gener Comput Syst 55:255–265

Balasubramaniam S, Jagannath R (2015) A service oriented IOT using cluster controlled decision making. In: Advance Computing Conference (IACC), 2015 IEEE International, IEEE, pp 558–563

Bhatia M, Sood SK (2017) Game theoretic decision making in iot-assisted activity monitoring of defence personnel. Multimedia Tools Appl 76(21):21911–21935

Choi J, Ahn S (2019) Scalable service placement in the fog computing environment for the iot-based smart city. J Inf Process Syst 15:2

Filieri A, Tamburrelli G (2013) Probabilistic verification at runtime for self-adaptive systems. In: Assurances for Self-Adaptive Systems, Springer, pp 30–59

Filieri A, Ghezzi C, Tamburrelli G (2011) Run-time efficient probabilistic model checking. In: Proceedings of the 33rd international conference on software engineering, ACM, pp 341–350

Filieri A, Tamburrelli G, Ghezzi C (2016) Supporting self-adaptation via quantitative verification and sensitivity analysis at run time. IEEE Trans Softw Eng 1:75–99

Garlan D, Cheng SW, Huang AC, Schmerl B, Steenkiste P (2004) Rainbow: Architecture-based self-adaptation with reusable infrastructure. Computer 37(10):46–54

Henry J, Tang S, Hanneghan M, Carter C (2018) A framework for the integration of serious games and the internet of things (iot). In: 2018 IEEE 6th international conference on serious games and applications for health (SeGAH), IEEE, pp 1–8

Hughes D (2018) Self adaptive software systems are essential for the internet of things. In: 2018 IEEE/ACM 13th International symposium on software engineering for adaptive and self-managing systems (SEAMS), pp 21–21

Jeong YS, Park JH (2019) Iot and smart city technology: challenges, opportunities, and solutions. J Inf Process Syst 15:2

Jo D, Kim GJ (2019) IOT+ AR: pervasive and augmented environments for ”digi-log” shopping experience. Hum Centric Comput Inf Sci 9(1):1

Jo H, Yoon YI (2018) Intelligent smart home energy efficiency model using artificial tensorflow engine. Hum Centric Comput Inf Sci 8(1):9

Kaur J, Kaur K (2017) A fuzzy approach for an iot-based automated employee performance appraisal. Comput Mater Continua 53(1):23–36

Kephart JO, Chess DM (2003) The vision of autonomic computing. Computer 1:41–50

Kim H, Lee E, Baik DK (2017) Self-adaptive software simulation: a lighting control system for multiple devices. In: Asian Simulation Conference, Springer, pp 380–391

Knauss A, Damian D, Franch X, Rook A, Müller HA, Thomo A (2016) Acon: A learning-based approach to deal with uncertainty in contextual requirements at runtime. Inf Softw Technol 70:85–99

Kumari V, Chakravarthy S (2016) Cooperative privacy game: a novel strategy for preserving privacy in data publishing. Hum Centric Comput Inf Sci 6(1):12

Kutsuna H, Fujita S (2011) A fair and efficient congestion avoidance scheme based on the minority game. J Inf Process Syst 7(3):531–542

Laddaga R, Robertson P (2004) Self adaptive software: A position paper. In: SELF-STAR: International Workshop on Self-* Properties in Complex Information Systems, Citeseer, vol 31, p 19

Lee E, Baik DK (2015) A verification technique for self-adaptive software by using model-checking. In: Proceedings on the International Conference on Artificial Intelligence (ICAI), The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), p 395

Lee E, Kim YG, Seo YD, Seol K, Baik DK (2017) Runtime verification method for self-adaptive software using reachability of transition system model. In: Proceedings of the symposium on applied computing, ACM, pp 65–68

Lee E, Kim YG, Seo YD, Seol K, Baik DK (2018) Ringa: design and verification of finite state machine for self-adaptive software at runtime. Inf Softw Technol 93:200–222

Lee J, Lee E, Baik DK (2016) Simulation and performance evaluation of the self-adaptive light control system. J Korea Soc Simul 25(2):63–74

Mohammadi V, Rahmani AM, Darwesh AM, Sahafi A (2019) Trust-based recommendation systems in internet of things: a systematic literature review. Hum Centric Comput Inf Sci 9(1):21. https://doi.org/10.1186/s13673-019-0183-8

Muccini H, Spalazzese R, Moghaddam MT, Sharaf M (2018) Self-adaptive iot architectures: an emergency handling case study. In: Proceedings of the 12th European conference on software architecture: companion proceedings, ACM, p 19

Nash J (1951) Non-cooperative games. Ann Math 1951:286–295

Nash JF et al (1950) Equilibrium points in n-person games. Proc Natl Acad Sci 36(1):48–49

Nisan N, Roughgarden T, Tardos E, Vazirani VV (2007) Algorithmic game theory, vol 1. Cambridge University Press, Cambridge

Ouechtati H, Azzouna NB, Said LB (2018) Towards a self-adaptive access control middleware for the internet of things. In: 2018 international conference on information networking (ICOIN), IEEE, pp 545–550

Park DM, Kim SK, Seo YS (2019) S-mote: smart home framework for common household appliances in IOT network. J Inf Process Syst 15:2

Rayes A, Samer S (2017) Internet of things–from hype to reality. The road to Digitization River Publisher Series in Communications, Denmark, p 49

Ribeiro AdRL, de Almeida FM, Moreno ED, Montesco CA (2016) A management architectural pattern for adaptation system in internet of things. In: 2016 international wireless communications and mobile computing conference (IWCMC), IEEE, pp 576–581

Salehie M, Tahvildari L (2009) Self-adaptive software: landscape and research challenges. ACM Trans auton Adapt Syst (TAAS) 4(2):14

Semasinghe P, Maghsudi S, Hossain E (2017) Game theoretic mechanisms for resource management in massive wireless iot systems. IEEE Commun Mag 55(2):121–127

Seo YD, Kim YG, Lee E, Seol KS, Baik DK (2018) Design of a smart greenhouse system based on mape-k and iso/iec-11179. In: Consumer Electronics (ICCE), 2018 IEEE International Conference on, IEEE, pp 1–2

Shoham Y (2008) Computer science and game theory. Commun ACM 51(8):74–79

Straffin PD (1993) Game theory and strategy, vol 36. MAA

Sylla AN, Louvel M, Rutten E, Delaval G (2017) Design framework for reliable multiple autonomic loops in smart environments. In: 2017 international conference on cloud and autonomic computing (ICCAC), IEEE, pp 131–142

Tallabaci G, Souza VES (2013) Engineering adaptation with zanshin: an experience report. In: Proceedings of the 8th international symposium on software engineering for adaptive and self-managing systems, IEEE Press, pp 93–102

Tao X, Li G, Sun D, Cai H (2017) A game-theoretic model and analysis of data exchange protocols for internet of things in clouds. Future Gener Comput Syst 76:582–589

Zhang W, Zhou R, Zou Y (2017) Self-adaptive and bidirectional dynamic subset selection algorithm for digital image correlation. J Inf Process Syst 13(2):305–320

Zheng J, Cai Y, Chen X, Li R, Zhang H (2015) Optimal base station sleeping in green cellular networks: a distributed cooperative framework based on game theory. IEEE Trans Wirel Commun 14(8):4391–4406