Toward the Application of Uncertainty Handling Methods to the Continuous Software Engineering

Hiromasa Inoki
Kyushu University
Fukuoka, Japan
inoki@f.ait.kyushu-u.ac.jp

Kenji Hisazumi
Kyushu University
Fukuoka, Japan
nel@slrc.kyushu-u.ac.jp

Takahiro Ando
Kyushu University
Fukuoka, Japan
ando.takahiro@f.ait.kyushu-u.ac.jp

Tsuneo Nakanishi
Fukuoka University
Fukuoka, Japan
tun@fukuoka-u.ac.jp

Akira Fukuda
Kyushu University
Fukuoka, Japan
fukuda@f.ait.kyushu-u.ac.jp

***Abstract*—Continuous software engineering has attracted attention because the scale of software is large and the frequency of changing specifications in accordance with the social situation and needs is high. In order to cope with uncertainties in continuous software development, a lifecycle-oriented architecture is required. It realizes the feedback between operational information and system design and ensuring the traceability of the operational information and the uncertainties. The framework for this uncertainty handling method has been proposed but the process of applying it to continuous software engineering is not defined. Therefore, we aim to clarify the detailed process and issues through a case study. We apply the method to the development of a parking lot recommendation system called "Free Parking System (FPS)".**

***Index Terms*—uncertainty, continuous software engineering, smart mobility**

I. Introduction

In recent software development, continuous software engineering(CSE) [1] has attracted attention because the scale of software is large and the frequency of changing specifications in accordance with the social situation and needs is high. However, CSE is currently focusing on methods for speeding up iterations, such as Continuous integration and Continuous Delivery, and does not apply a method for appropriately managing feedback in iterations.

A lifecycle-oriented architecture realizes the feedback between operational information and system design by analyzing which operational information should be collected to determine the uncertainties identified in the development stage, and ensuring the traceability of the operational information and the uncertainties. This makes it possible to cope with uncertainty and facilitate continuous software development.

Although a framework for this uncertainty handling method has been proposed, the process of applying it to actual continuous software engineering is not well defined. Therefore, in this paper, we aim to clarify the detailed process and issues when applying this method to actual continuous system development through case study. As a case study, we apply the method to the development of a parking lot recommendation system called "Free Parking System (FPS)".

The rest of paper is organized as follows: Section 2 introduces existing researches about the definition of uncertainty, related modeling notations, and the uncertainty handling method. In Section 3, we propose the process of applying the uncertainty handling method to continuous software engineering. Section 4 describes a case study conducted to demonstrate the proposed method. Section 5 describes some discussions, and Section 6 summarizes this paper.

II. Related Works

A. Continuous Software Engineering(CSE) [1]

CSE is a comprehensive term that describes several aspects of iterative software application development, including continuous integration, continuous delivery, continuous testing, and continuous deployment.

1) Continuous integration: Continuous integration refers specifically to the process of steadily adding new code commits to source code. Each team member to submit work as soon as it is finished and for a build to be conducted with each significant change. Usually, a certain baseline of automated unit and integration testing is performed to ensure that new code does not break the build. This way developers know as soon as they're done if their code will meet minimum standards and they can fix problems while the code is still fresh in their minds. An important advantage of continuous integration is that it provides developers with immediate feedback and status updates for the software they are working on.

2) Continuous delivery: Continuous delivery builds on continuous integration and as with continuous integration, each code commit is automatically tested at the time it is added. In addition to the automated unit and integration testing, a continuous delivery system will include functional tests, regression tests and possibly other tests, such as pre-generated acceptance tests. After passing the automated tests, the code changes are sent to a staging environment for deployment.

3) Continuous testing: Continuous testing adds manual testing to the continuous delivery model. With continuous

testing, the test group will constantly test the most up-to-date version of available code. Continuous testing generally adds manual exploratory tests and user acceptance testing. This approach to testing is different from traditional testing because the software under test is expected to change over time, independent of a defined test-release schedule.

4) Continuous deployment: Continuous deployment adds more automation to the process to the software development process. After passing all the automated delivery tests, each code commit is deployed into production as soon as it is available. Because changes are delivered to end-users quickly and without human intervention, continuous deployment can be seen as risky. It requires a high degree of confidence both in the existing application infrastructure and in the development team.

B. Definition of uncertainty

The paper [2] defined that uncertainty is the unresolved information needs that the developers identify and share, which is definitely needed for developing the target system. In addition, the uncertainty must be defined so that it can clearly judge whether or not the expected information is obtained.

C. D-Case [3]

D-Case(Dependability Case) is a notation for goal-oriented analysis, is used to make a clear Assurance Case. Basically, the following five symbols are used in D-Case.

- Goal: Proposition to discuss for the system
- Strategy: Discussion strategies for dividing goals into subgoals
- Solution: Evidences finally guaranteeing that a refined goal has been achieved
- Context: Informations to be considered in discussing the goal
- Undeveloped Goal: Goals that are not given evidence and can not be further refined
- Monitor: Operational information to guarantee the goal

The proposed method described later uses D-Case because D-Case is suitable for the method which aims to resolve uncertainties by using operational information.

D. Development Process for Realizing Feedback of Operational Information to Request and Design

The paper [4] proposed a development method for realizing feedback of operational information to request and design. In this method, the uncertainties found at the request and design stage are focused on, and they are fixed by using the operational information. By modifying the model based on the fixed result of uncertainty, realize the change of request and design smoothly. In the proposed method, this development method is used to realize feedback between operational information and requirements and design.

The procedure of the method consists of three steps.

STEP1: Making uncertainty table

The first step is making an uncertainty table based on the request specification. This table summarizes the options, influence ranges, dependency relationships, and fixed results on each uncertainty.

- Options: Solution candidates for each uncertainty
- Influence ranges: Parts of the model diagram that may change due to each uncertainty
- Dependency relationships: Dependence between uncertainties
- Fixed results: Name of fixed option(If it has not fixed, leave it blank)

STEP2: Analysis of operational information

- Making model of uncertainty:
  At the beginning of STEP2, the model of uncertainty is made by using notation such as GSN or D-Case. The goal of it is properly resolving of each uncertainty. Some goals can be given ”Solution” at design and development stage, but others cannot. Such a goal is given ”Monitor”, list of the operational informations which are necessary for achieve the goal.

- Making table of operational informations:
  The operational informations, described as ”Monitor”, are summarized in this step. The table shows what the corresponding uncertainty is and whether the function of collecting operational informations is implemented.

STEP3: Improvement plan analysis

After starting operation of the system, uncertainties are traced using the obtained operational information and consider fixing them based on the table of operational informations. When an uncertainty is fixed, feedback is realized by changing the corresponding part of the request and design model based on the uncertainty table.

III. Proposed Method

The current continuous software engineering does not apply an appropriate method of managing feedback in iterations of development. In order to smoothly feed back the obtained operational information to requirements and design, we propose the process of applying the uncertainty handling method to countinuous development. The process is shown in Fig. 1. First, create a D-Case about the application and find out uncertainties. At this time, traceability between operational information and requirements / design is secured based on uncertainty handling method. Then, classify each uncertainty depending on whether it is resolved in the simulation phase or the operation phase. When classification is over, start the simulation phase. Resolve the uncertainty that can be

fixed by simulation and improve the system. After resolving uncertainties by simulation and improving the system as much as possible, start the operation phase. Resolve the remaining uncertainties by operational information and further improve the system.

By repeating this process of resolving the uncertainties and improving the model using the operational information in each phase, it contributes to the continuous software engineering based on the operation.

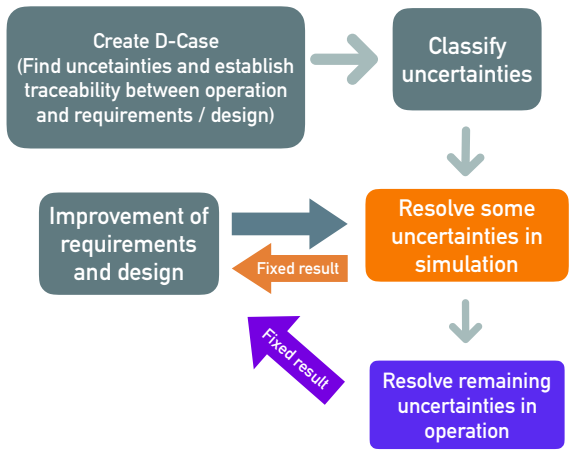

Fig. 1. The overall picture of the proposed method

When using this method, it is necessary to make D-Case. The procedure for making D-Case is as follows.

1) ”The application operates properly” is set as the top ”Goal.”
2) Divide Goal into detailed sub goals according to the ”Strategy”(how to divide the goal). The prerequisite information is connected with each goal as ”Contexts” when describing the goal.
3) Repeat the operation of 2 until a goal cannot be divided anymore.
4) When a goal that can not be divided anymore is found, information that guarantees the goal is described as ”Monitor” and connected with the goal. For a goal that does not have Monitor, add a mark indicating ”Undeveloped.”

Monitors include information necessary for resolution of uncertainties that are unknown at the requirement analysis, design, and implementation phase. They are divided according to which phase each information is obtained. In order to recognize the division, each Monitor is classified visually by assigning unique color it. This coloring method makes it easier for developers to know what information to note when simulating or operating.

Fig. 2 shows an example of creating a D-Case. The D-Case set ”The system is grasping the vacancy of the parking lot” as top ”Goal.” ”The system gets sufficient data from application users and sensors” is connected with the goal as ”Context” because it is prerequisite information. The goal is divided into two goals according to the ”Strategy” which says ”Divide by conditions.” Each goal is divided again into two goals according to strategy ”Divide by elements.” Since further division cannot be found, information to guarantee each of these four goals is described in ”Monitor” and connected. At last, each Monitor is classified by assigning color it. The information obtained in the simulation phase is colored with orange and in the operation phase is colored in blue.

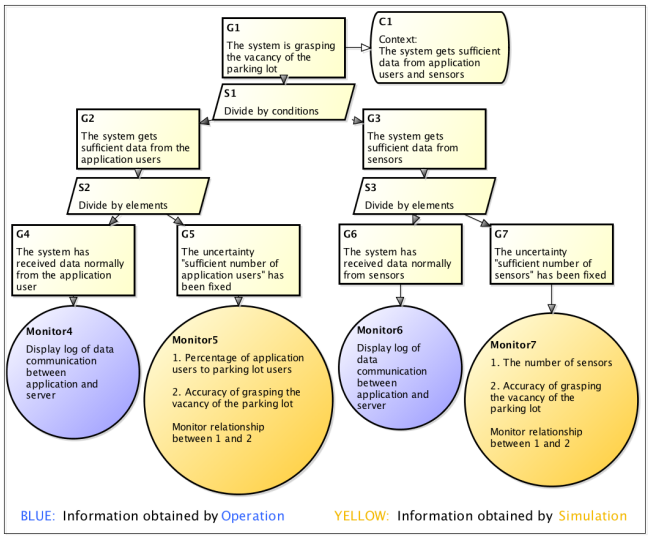

Fig. 2. D-Case example for parking recommendation application

IV. Case Study

For the case study, we use parking recommendation application which is using a system named ”Free Parking System (FPS)” [5]. This system is a parking recommendation system that targets a busy parking lot like a free curbside parking. The system has three major features as follows.

1) It consists of a smartphone application and a server and does not necessarily need a sensor.
2) Priority is given to shortening the total travel time for the entire user, not to shortening the travel time of individuals.
3) It also considers the existence of drivers who do not use the system.

We apply uncertainty handling method to this system. The created D-Case model is shown in Fig. 7. ”G” means Goal, ”S” means Strategy, and ”C” means Context in the figure. In the applying, we set ”The parking assignment application operates properly” as top goal(G1) at first, and divided it into subgoals according to Strategy ”Divide by function(S1).” After repeat of this procedure, we connected Monitor to the goal which is judged that it cannot divide any further. If it doesn’t need Monitor, a diamond mark which means ”Undeveloped” has been attached. Each Monitor describes the information necessary to achieve the goal, and it is colored orange or blue. The informations described in orange Monitors can be

confirmed at the simulation phase, and in blue Monitors can be confirmed at the operation phase.

FPS does not require a sensor, but we think that the accuracy of the parking assignment can be further improved by using several sensors. Therefore, ”There is sufficient data from the sensor(G18)” exists in the subgoal to achieve the goal ”The system grasps the vacancy information of the parking lot(G9).”

As a result of creating the D-case as described above, we were able to discover some uncertainties included in FPS. Among them, this paper focuses on ”sufficient number of sensors(G22).” We resolved the uncertainty through simulation using SUMO (Simulation of Urban MObility) [6].

A. Simulation Settings

The simulation was carried out using map data of Ito Campus of Kyushu University. The three parking garages were set as three stories each. We set the number of vehicles visiting campus based on traffic survey [7] every hour from 7 to 18. The number of vehicles is shown in Fig. 3. Each car was set to go to one of the three parking garages, and the parking assignment was done in the floor unit.

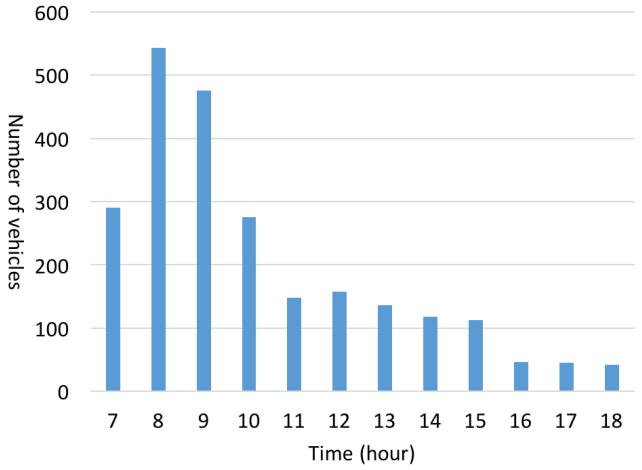

Fig. 3. Number of vehicles visiting Ito campus

B. Evaluation: sufficient number of application users

By changing the proportion of vehicles to which the parking recommendation application is applied, ”Percentage of application users in the whole driver” is changed, and ”Percentage of parking assignment error to application users” per hour is confirmed through simulation. ”Parking assignment error” means that the application user can not park as planned. There are two kinds of errors floor error and building error. The floor error makes users change the floor. The building error makes users change the parking garage. These have different importance, but this time we treat them as the same error. The result is shown in Fig. 4. ”Parking ratio” means what percentage of the parking lot is parked. Looking at the solid line representing the simulation result at each user percentage, the percentage of parking assignment error is the highest at 9 a.m. in most user percentages. For the sake of clarity, the graph focusing on only the percentage of parking assignment error at 9 a.m. is shown in Fig. 5 . Looking at 9 a.m., we can see that when the percentage of users exceeds 70% of all drivers (red line in graphs), the error rate falls below 50%. From this, we concluded that ”70% of all drivers” is appropriate for ”sufficient number of application users.”

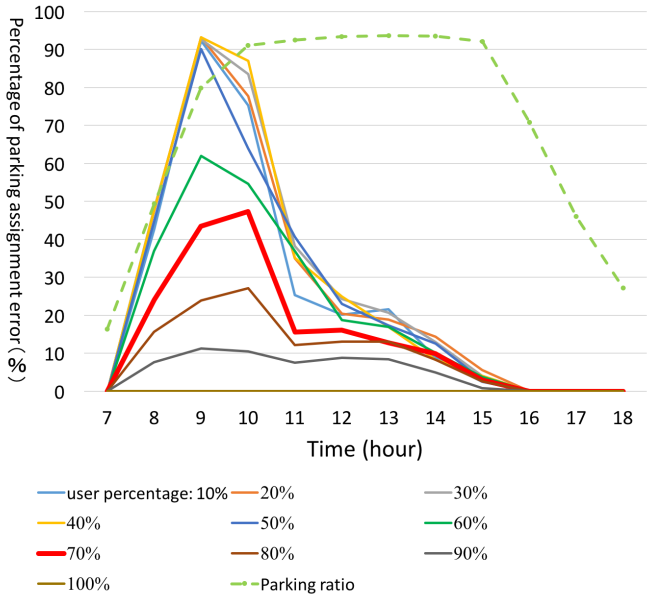

Fig. 4. The relationship between user percentage and parking assignment error rate

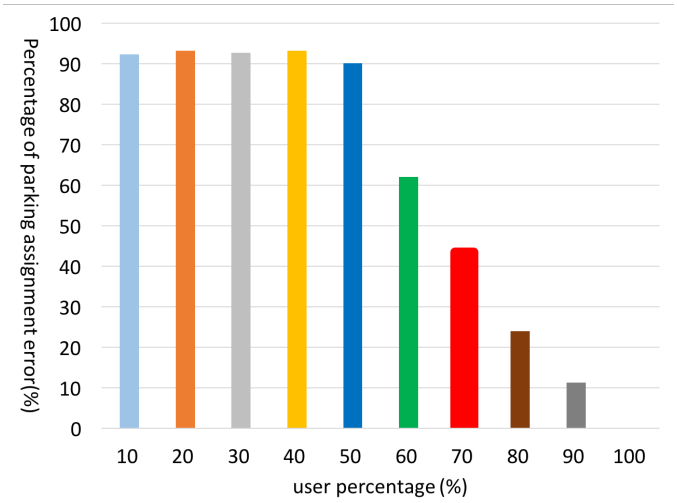

Fig. 5. The relationship between user percentage and parking assignment error rate at 9 a.m.

C. Evaluation: sufficient number of sensors

”Number of sensors installed in the parking garage” was changed, and ”Percentage of parking assignment error to application users” every hour was confirmed through simulation.

”Parking assignment error” means that the application user can not park as planned. There are two kinds of errors: floor error and building error. The floor error makes users change the floor. The building error makes users change the parking garage. These have different importance, but this time we treat them as the same error.

In the simulation, sensors were installed to three parking garages in four ways: only on each 1st floor, each 1st and 2nd floor, all the floors, and no sensors. Application user rate was 30% in this simulation.

The result is shown in Fig. 6. When the sensor was installed only on the first floor, the percentage of parking assignment error rapidly increases at 9 a.m. On the other hand, when the sensor was installed on the first and second floor, the percentage rapidly increases at 10 a.m. This result is because cars park from the first floor in order, it is considered that the percentage of parking assignment error is increasing at the time when parking starts on the floor where sensors are not installed. Regarding the maximum rate of the parking assignment errors, the parking assignment error rate is less than 50% when installing sensors on the 1st and 2nd floor. From this result, we concluded that ”install sensors on the 1st and 2nd floor” is appropriate for ”sufficient number of sensors.”

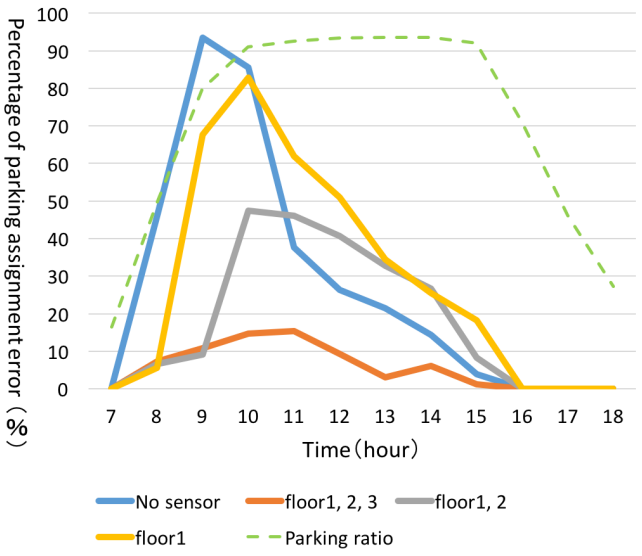

Fig. 6. The relationship between the floors on which the sensor is installed and parking assignment error rate

V. Discussion

A. Case study achievements

In this case study, the D-Case diagram clarified uncertainties involved in system development. In addition, we clarified in the Monitor what operational information is necessary to resolve each uncertainty. The color coding of Monitor indicates the timing to focus on each Monitor. Then, the simulation was performed, and the uncertainty could be resolved smoothly using the operational information obtained by the simulation.

B. Continuous verification of the proposed method

Two simulation-related uncertainties were solved through case study, but this is not sufficient to verify the application of the method to continuous software engineering, so it is necessary to continue the case study, including operation phase. Specifically, we will confirm what is uncertain again due to changes in the simulation environment settings such as the number of users and building settings, and clarify what information is necessary for that.

C. Automate processes by clarification of criteria

Each Monitor in D-Case is color-coded according to when the operation information can be obtained. However, this classification criterion is still ambiguous, so its clarification is necessary. If this clarification is realized, it will be possible to automate classification. The resolving criteria for uncertainties are also needed because it probably enables automation of the improvement process of requirements and design based on operation information. Specifically, information such as the conditions for automatically determining uncertainty, priority, and weight will be added to the model by connecting to Monitor. In addition, it will provide a means to acquire information described in Monitor. Uncertainty will be determined by the information described in Monitor and the information connected to it.

D. Responding to changes in uncertainty and its fixed result

It is difficult to truly determine uncertainties in continuous software engineering. The uncertainties that have been determined once may become indeterminate again or it may often be better to change the ”Fixed results.” Furthermore, the number of uncertainties may increase or decrease. In order to cope with these problems, tools for managing uncertainties will be needed. We intend to model uncertainty cancellation criteria in D-case notation, evaluate them based on continuous testing and obtained Monitor information, and make decisions again.

E. Uncertainty about design

Uncertainty about design is also an issue. We have analyzed the uncertainty about the requirement that ”The parking recommendation application operates properly,” but do not consider the uncertainty about the design. It is also necessary to verify design uncertainty and cases where uncertainty in requirements affects the design.

VI. Conclusion

This paper has summarized the definition of uncertainty and uncertainty handling method, and proposed the application of the method to continuous software engineering. As a case study, the uncertainty handling method was applied to the development of the parking lot recommendation system called FPS. Through this case study, we clarified uncertainty by making D-Case diagram, classified Monitor based on the timing when operational information of Monitor is obtained, resolved uncertainty by simulation, and showed the process

of applying the uncertainty handling method for continuous software engineering.

Future works of this research are to continue applying the method to continuous software engineering, clarify the criteria of Monitor classification and uncertainty determination, cope with changes of uncertainties and their fixed result, and consider design uncertainty.

ACKNOWLEDGMENT

This work is supported by JSPS KAKENHI Grant No.15H05708.

REFERENCES

[1] Bosch, J.: Continuous Software Engineering, Switzerland, Springer, 2014.
[2] Nakanishi, T., Ma, L., Hisazumi, K., and Fukuda, A.: A Framework to Manage Uncertainty in System Development, IPSJ SIG Technical Report, pp.1-6, 2014.
[3] Kelly, T., and Weaver, R. ,: The Goal Structuring Notation A Safety Argument Notation, Proceedings of the dependable systems and networks 2004 workshop on assurance cases, 2004.
[4] Ishibashi, S., Hisazumi, K., Nakanishi, T., and Fukuda, A.: Establishing Traceability between Requirements, Design and operational information in Lifecycle-Oriented Architecture, Advanced Applied Informatics (IIAI-AAI 2016), 5th IIAI International Congress on. IEEE, 2016.
[5] Hakeem, A., Gehani, N., Ding, X., Curtmola, R., and Borcea, C.: On-The-Fly Curbside Parking Assignment, MobiCASE'16 Proceedings of the 8th EAI International Conference on Mobile Computing, Applications and Services, pp.1-10, 2016.
[6] German Aerospace Center, Institute of Transportation Systems: Simulation of Urban MObility, http://sumo.dlr.de/index.html, accessed 2018-02.
[7] Ministry of Land, Infrastructure and Transport: Heisei 27th Nationwide Road/Street Traffic Situation Survey General Traffic Survey Total Schedule, http://www.mlit.go.jp/road/census/h27/, accessed 2018-04.
[8] Chen, C., Hisazumi, K., Katahira, M., Nishihara, Y., Kawai, A., Nakanishi, T., and Fukuda, A.,: Requirement, Deployment, and Design Model Including Uncertainty, Embedded Systems Symposium 2013, pp.75-80, 2013.
[9] Bishop, P., and Bloomfield, R.,: A methodology for safety case development, Industrial Perspectives of Safety-Critical Systems. Springer London, pp.194-203, 1998.
[10] Kelly, T.,: Arguing Safety, a Systematic Approach to Managing Safety Cases, PhD Thesis, Department of Computer Science, University of York, 1998.

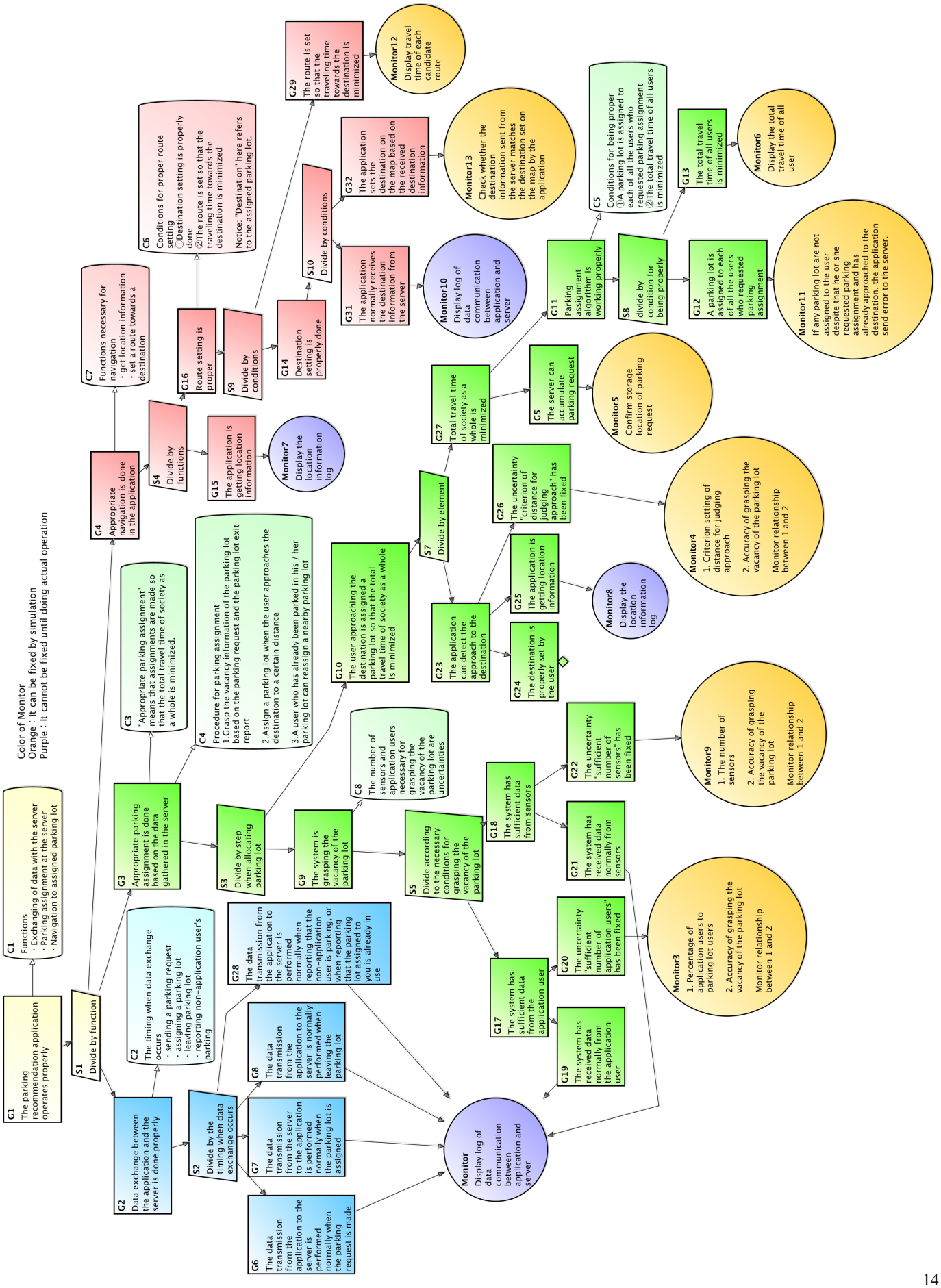

Fig. 7. D-Case model for parking assignment application