Capturing and Analyzing Requirement

In case of Software and Applying to Hardware

Akira Kawaguchi

Solution Design Laboratory Co.,Ltd.
Yokohama City Kanagawa Pref. 221-0062
Tel:03-3443-9720
Fax:03-3443-9721
e-mail:a-kawaguchi@msf.biglobe.ne.jp

Abstract - It introduces the technology of requirement capture and requirement analysis that uses the UML in the software world, and considers applying to hardware development.

I. Introduction

In the world of software, requirement for system is becoming large-scale and complicate every year. So in order to capture requirement correctly, it is popular to use UML for describing specification of system. As one of important feature of the UML, there is description of requirement using use cases. Furthermore, in order the maximum to utilize requirement description by use cases, the development process which proceeds every use cases is applied. With the combination of use cases and development process, while feeding back the capture and the realization of requirement mutually, it becomes possible to advance development. Namely the mechanism which can manage system development itself by use cases, has been prepared.

On one hand, regarding the hardware, requirement is becoming large-scale and complicate. So same as the software, it is expected that the process that can specify requirement correctly and can realize it under managing. In this paper, It is introduced that capture and analysis process of the requirement which uses the UML in the software, and we consider the application to hardware development.

II. Description of requirement

To begin with, Requirement is defined that a condition or capability to which a system must conform. In addition, it is roughly classified to functional requirement and non-functional requirement [1]. Here, the Requirement in the software is shown in every the characteristic. It examines whether with the hardware and the software there is a difference in requirement.

A. *Functional requirement*

Physical constraint is not considered and specifies the actions that the system must perform. For example, behavior of input/output of the system is specified. Inputting what, it outputs what and so on.

B. Non-functional requirement

Other than functional requirement, the system must conform, For example, performance, interface, physical design constraint, design constraint concerning architecture and implementation constraint et cetera.

* Performance requirement
  Speed and the throughput that the system should fill up, response time, memory amount used and the like is specified.

* Interface requirement
  Interface between system and its outside item which system must interact is specified. In addition, the format, timing and the other factors regarding the interaction are made clear.

* Design constraint
  This is the requirement that constricts the design of the system. For example, constraint keeps extensibility and constraint keeps maintainability. And the platform dependency et cetera that relate to the reuse of the legacy system or the essential part.

This way trying looking around requirement, the requirement that is hardware peculiar does not exist, in the same way the requirement that is peculiar to the software

does not exist.

Especially functional requirement originally does not consider physical constraint, so it is possible to handle without distinction between hardware and software.

Therefore, at the time of requirement describing, it is not necessary to distinguish hardware from software, can describe with identical method.

C. Description of requirement using use cases

In the world of the software, the mechanism of the requirement description, use case is utilized. One use case specifies the consecutive actions that include the variable part. These actions are executed by the system and produce concrete value for specific actor. The actor shows the user of the system and the person and those that interact with the system.

In another expression, a use case bundles requirement that brings the individual concrete value to specific actor.

Fig.1 shows Use case Diagram that expresses relation among use case and actor.

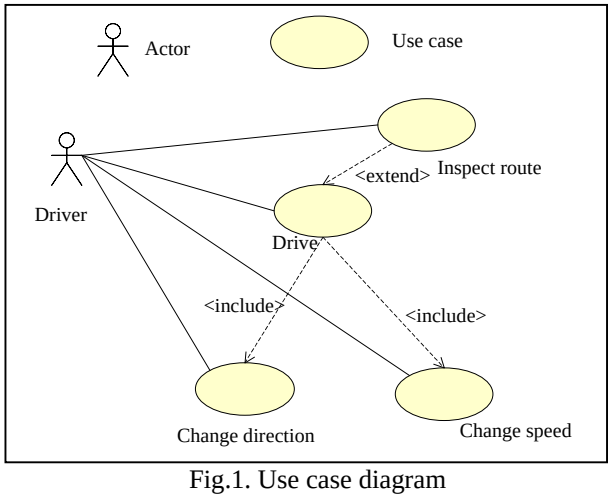

Fig.1. Use case diagram

D. Use case and development process

Requirement for system , especially functional requirement specifies the actions which the system has to perform. In actual development, the actions that the system should execute are bundled with a certain scope, and that bundled actions are developed at same turn. This is called incremental development, and it is thought it is practical approach.

The advantage of this development process is approximately as mentioned below. The cost that depends on one increment can be controlled. If certain iteration was the failure, it is sufficient that the cost increase of one batch. Risk of schedule delay can be lightened, early stage of development risk can be identified. User requirements being to be able to refine in the consecutive iterations, it is easy to be adapted to the modification of requirement.

When this kind of incremental development is done, it becomes important to show concretely the result, which is achieved with every iteration in the form that is easy to be understood by the customer or the user.

At each time iteration ends, it is verified whether the result of that iteration conforms to the intention of the customer and the user of that system. This way, It is confirmed whether system development is in progress according to plan.

With this incremental development, use case is effective as the scope bundles the actions that it should develop with iteration of one time. Use case has specified the consecutive actions which produces concrete value for a certain actor. Therefore if incremental development has done using use case as its scope, the result is expected to become concrete value for user and the like which is actor. In other words, if one use case is realized, it means that one concrete value is offered to the user and the like outside the system.

This way, use case becomes the tool in order to organize requirement in management easy shape at the time of incremental development process. By the fact that the use case is used as the scope of development, the following kind of merit is added to the merit of incremental development.

Customer, system user, and developer closely examine each use case in first stage of development, attache priority to the value that is actualized by the respective use case. Following to this priority, deliberate development becomes possible.

It meaning that one value is actualized with iteration of one time, order side of system development such as user and customer is easy to grasp the progress and the contents of development. Because of this, risk of schedule delay can be lightened.

By the fact that early of development, risk is estimated in every use case, risk can be grasped.

E. In order to capture requirement as use case

There are many requirements in a system. In order to capture a certain range from a lot of requirement as one use case, it is necessary to make the scope of the use case clear. On description below it shows the point to notice in order to capture requirements as a use case following to the clear scope.

First, designate the consecutive actions, which give value to one actor outside the system as the bundle. In other words, the scope of use case is limited by the fact that the object that gives value is restricted to the individual actor.

Furthermore, by making the content of value more concrete, the scope of use case is limited more. For example, [the system, car] [the actor, driver] is thought. First, vis-a-vis the value that it can move favorite to the place, you can think the use case that " drive ". If this furthermore is disassembled in concrete value, " it can move favorite at speed ", " it can move favorite to direction" and so on it becomes. As the use case which gives these values, " change direction of the car ", " change speed of the car", and so on is extracted. This way, by making the value that the use case gives to the actor concretely, the scope of the use case becomes clearness and is limited to the appropriate range.

In case of incremental development using use case, lowest step size to proceed becomes one use case. " Drive " the way, when it develops with the use case to which the scope is too large, the merit of incremental development fades. By the fact that it disassembles to concrete value, it becomes important to limit the scope of the use case to the appropriate range.

In addition, to also complexity of the value that is actualized with one use case should be noted. For example, we assume the value that is actualized with a certain use case, is actualized with combination of other plural values. In this case, new use cases that actualize the respective value should be specified, and original use case includes those new use cases. The relationship of this is called Include [2]. Complexity of the value that is actualized with one use case can be controlled using this kind of mechanism.

F. The requirement which use case describes

Because the use case bundles the consecutive actions that produce the concrete value for outside the system, description of functional requirement becomes prime object. In other words, in regard to functional requirement, it can structure naturally with use case.

In regard to non- functional requirement, other than of performance requirement is circumstances of system realization, is not the object that the use case should describe directly. But, it can specify most non- functional requests, vis-a-vis the individual use case. Because of this, in many case it is possible to handle with the respective use case description. Non-functional requirement which is common, to the plural, or all use cases is described in the document which is independent of individual use case, is handled as supplemental requirement. These are referred through the whole development process.

G. Applying hardware

As above, requirement description by use case is the technology that can be applied without distinction of the hardware and the software. But in case of ordinary software it is general that environment for realization already exists as a platform. Because of this, relatively it is easy to do requirement description, analysis, design and implementation incrementally every use case.

However, when it is the system whose hardware development is necessary, in many case it is not practical to iterate to implementation in every use case. Hardware implementation needs more cost and time than software in every iteration of one time, and use case has relatively small scope, so iteration number becomes many. As a result cost and time increases. This not only the hardware system, is similar even in the domain where it is called the generally known embedded software. Therefore, in the system that includes hardware development, when it does incremental development using use case, it is necessary to do some device. It mentions later in regard to this problem.

III. Requirement analysis

As shown in the foregoing paragraph, assuming, individual requirement had be captured by the use case it is not appropriate that way to enter into actual design.

Because, each use case, in order to conform to the incremental development process, bundles requirements that can be realized within one iteration. Therefore if the individual use case could be described completely, requirement and its structure for the whole system cannot be captured. Then, it is necessary to extract function and the behavior to realize the overall system by looking around the most of use cases with the concept that is orthogonal to use case. Furthermore, while refining those function and behavior, it structures simultaneously. This is Requirement analysis.

In addition, it can grasp interference, concurrency, and conflicts among use cases by this activity, these characteristics are not found with just use case. Furthermore, it becomes possible to add the structure that considers flexibility and reusability for the modification of requirement.

A. Activity of requirement analysis

In order to know the structure of entire requirement, analyst takes a second look at requirement over the whole system using the conceptual object model (analysis model) and assigns function and behavior of the system in combination of object. Analysis model contains analysis class and analysis package.

Simultaneously, analyst considers the structure inside the system that includes the common resource.

Furthermore, the structure that considers flexibility and the reusability for the modification of requirement is added. This structure should be kept till design and implementation phase.

Furthermore, with the analysis model, analyst are not conscious of realization environment, analyst tries to investigate the functional level that is required to the system.

Being to become the argument that does not depend on the environment of realization, it does not handle non-functional request. Non-functional requirement handles mainly with design and implementation phase.

B. Analysis class

Analysis class localizes use case description, especially the functional part that is expressed with natural language, in conceptual object. It becomes more structural expression. As for analysis class, three types of Boundary, Control and Entity class are used depend on the function that is mapped to each object.

Fig.2 shows the example of analysis class.

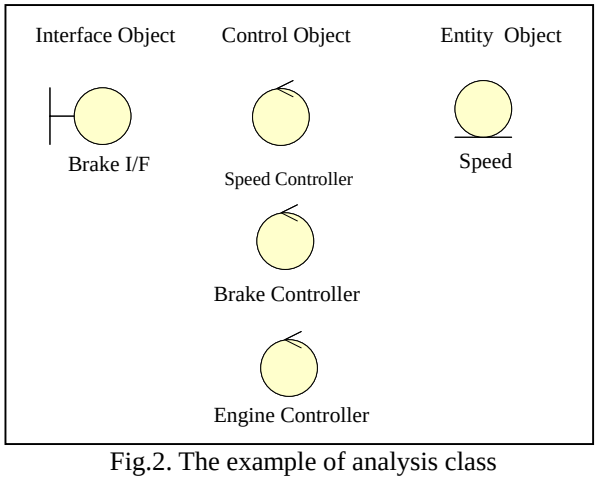

Fig.2. The example of analysis class

* Control object

Control flow inside the use case such as coordination, sequence, and transaction, etc. is localized in control object, and assigns work to other object (entity and boundary). Behavior of one use case is usually localized in one control object. Relating to the plural use cases is rare case.

* Entity object

It has long-term life, or is often permanent information is localized in entity object. Often, it shows logical data structure, it is useful to understanding of the information where the system depends.

* Boundary object

Interaction between System and its actor (for example user, external system, and so on) is localized in this object. Boundary object models a part of system that depend on actor, for example communication interface or user interface, like that.

C. Use case realization by analysis class.

At first stage functional behavior of the system is assigned in analysis class of three types of Boundary, Control and Entity, and it is expressed by those collaborations. It is necessary to analyze how to combine the objects that offer some kind of function to realize respective action that is included in each use case. Functional behavior of the system is re-expressed by the relationship of analysis objects. This is called use case realization.

Fig.3 shows the example of use case realization.

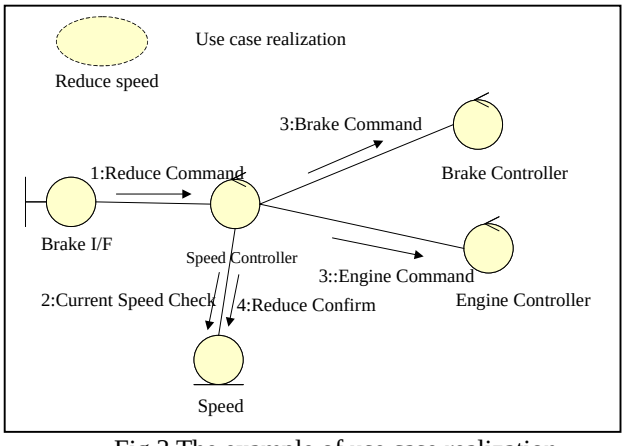

Fig.3 The example of use case realization.

Goal of the use case realization that uses analysis class takes a second look at almost of use cases, it is to inventory the functions which is necessary for the system without omission. Simultaneously it is required to grasp internal structure and to refine it. For example, extracting conflict and function that can be shared among use cases, mapping functions to analysis classes with considering reusability and extensibility.

On the basis of the result of these activities, architecture base line construction of the system which uses analysis package and the like which it mentions later is done.

D. Architecture base line construction using analysis package.

Among actually development, as for the case that can develop the whole system with one organization it is not many. Usually, plural organizations respectively develop the charge part (it calls subsystem) that is initially assigned, finally, integrating those it is general to develop one system. As a primary plan of this subsystem decomposition, the architecture base line construction by the analysis package is done.

The analysis package is the tool to organize object of Control, Entity and Boundary that are defined in use case realization in the group which is easy to manage each object. At the point of view that is orthogonal to use case, you look around all functions that are required to the system, and

reconstruct in an optimum group with analysis package. Objects whose relationship is deep are grouped to the same analysis package, and relationship among analysis packages should to be lightened. In addition, functions of the objects which grouping in same package have to belong to identical domain, namely in order to be developed by identical organization, it is important point for grouping. With this, analysis package becomes primary candidate of the subsystem that each developing team who belongs wrong organization and has respectively wrong domain knowledge can design and implementation independently and concurrently.

This way, the architecture base line that consists of use case, analysis class, and analysis package that are primary candidate of subsystem, and interface between the analysis packages is drawn up. Fig.4 shows a part of architecture base line.

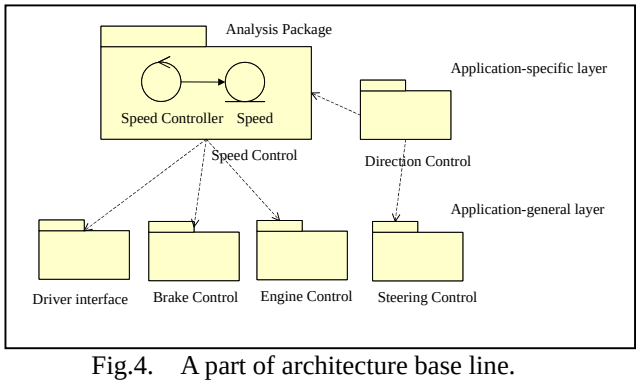

Fig.4. A part of architecture base line.

Actually subsystem decomposition is done on basis of architecture base line, adding the various factors. For example, human resource, domain knowledge skill, know-how, intellectual property, etc.

E. Applying requirement analysis to hardware development

As for requirement analysis, functional requirement is principal basically, so there is no distinction of hardware and software. In addition, the architecture base line that is the result of requirement analysis is effective regardless of the hardware and the software.

Way it shows with requirement description, In incremental development, when it is the system needs hardware development, in many case it is not practical to iterate to implementation in every use case, because of cost and time for hardware implementation.

Then, It is effective to iterate in every use case from the requirement description to requirement analysis that can be applied without distinction of hardware and software and to construct the architecture base line.

After this phase, design, implementation, there is a part that cannot be covered with just software development process, such as Co-design for hardware and software.

In regard to design and implementation, it is thought applying the technology that is individually optimum with the architecture base line as an input, as practical solution.

As for this, there is approach that separates function and behavior of the system that is captured with requirement description and requirement analysis phase from the realization that is constructed with design and implementation phase. In the world of the software, this is called MDA (the Model Driven Architecture) and watched [3].

Fig.5 shows the summary of incremental development process using use case for hardware and embedded software.

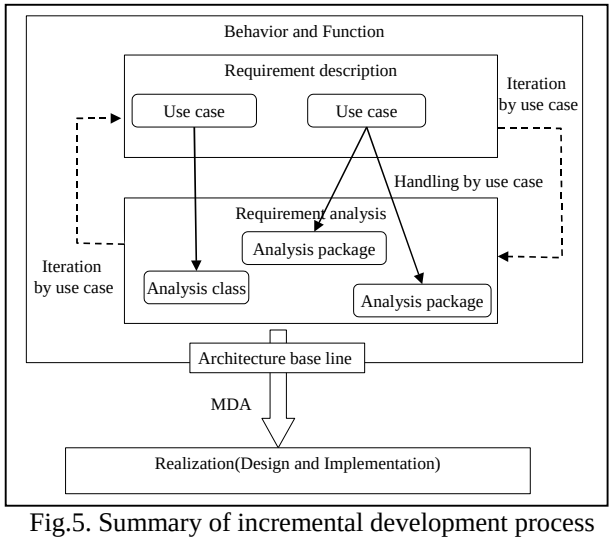

Fig.5. Summary of incremental development process using use case for hardware and embedded software.

For example, when new realization appears such as C based design or the application of the large capacity FPGA in the LSI, it is possible to reuse the model of function and behavior, because that model exists independent of realization. Mapping from the model to realization is thought to automate using some suitable tools.

IV. Conclusion

Above, it introduced the technology of requirement description and requirement analysis that uses the UML in the software world, and considered applying to hardware development.

It is thought the effect is shown sufficiently in hardware development, in regard to requirement description and requirement analysis that is not influenced by realization environment. In addition, the MDA (Model Driven Architecture) approach probably becomes one guide for

mapping the model of function and behavior that are captured with requirement description and requirement analysis to realization environment.

References

[1] Ivar Jacobson, Grady Booch, James Rumbaugh, The Unified Software Development Process, Addison-Wesley, 1999
[2] Grady Booch, James Rumbaugh, Ivar Jacobson, The Unified Modeling Language User Guide, Addison-Wesley, 1999
[3] http://www.omgj.org/technology/mda/