International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:6, No:9, 2012

An Automation of Check Focusing on CRUD for Requirements Analysis Model in UML

Shinpei Ogata, Yoshitaka Aoki, Hirotaka Okuda, Saeko Matsuura

***Abstract*—**A key to success of high quality software development is to define valid and feasible requirements specification. We have proposed a method of model-driven requirements analysis using Unified Modeling Language (UML). The main feature of our method is to automatically generate a Web user interface mock-up from UML requirements analysis model so that we can confirm validity of input/output data for each page and page transition on the system by directly operating the mock-up. This paper proposes a support method to check the validity of a data life cycle by using a model checking tool "UPPAAL" focusing on CRUD (Create, Read, Update and Delete). Exhaustive checking improves the quality of requirements analysis model which are validated by the customers through automatically generated mock-up. The effectiveness of our method is discussed by a case study of requirements modeling of two small projects which are a library management system and a supportive sales system for text books in a university.

***Keywords*—**CRUD, Model Checking, Model Driven Development, Requirements Analysis, Unified Modeling Language, UPPAAL.

I. Introduction

MODEL Driven Development [1,2,3,4] is a promising approach to develop high quality software products efficiently. Supporting tools such as a source code generator and several domain specific languages have been proposed [4]. However, to obtain high quality source codes, appropriate models that meet customer's requirements should be well defined at the requirements analysis phase which is a start point of the system development. At the requirements analysis phase, it is difficult to strictly define requirements analysis models (RA models) so that they can be translated into the source codes. This is because that the user requirements are often ambiguous, imprecise, insufficient and incomplete. To make the RA models precise, the developers should fully understand user requirements and define the problems that the customer is trying to solve as precisely as possible. Moreover, the requirements specification is the result of analysis so that it can offer correct and sufficient information to the following phases to generate the final product automatically.

S. Ogata is with the Shinshu University, Wakasato 4-17-1, Nagano Nagano 380-8553 Japan (e-mail: ogata@cs.shinshu-u.ac.jp).
Y. Aoki is with Shibaura Institute of Technology, Minuma-ku Fukasaku 307, Saitama Saitama 337-8570 Japan (e-mail: ma11081@shibaura-it.ac.jp).
H. Okuda is with Shibaura Institute of Technology, Minuma-ku Fukasaku 307, Saitama Saitama 337-8570 Japan (e-mail: ma11043@shibaura-it.ac.jp).
S. Matsuura is with Shibaura Institute of Technology, Minuma-ku Fukasaku 307, Saitama Saitama 337-8570 Japan (e-mail: matsuura@se.shibaura-it.ac.jp).

We have proposed a method of model-driven requirements analysis [5,6] using Unified Modeling Language (UML[7]). The main feature of our method is to automatically generate a Web user interface (UI) mock-up from UML RA model so that we can confirm validity of input/output data for each page and page transition on the system by directly operating the mock-up.

Models are effective in specifying the target system by the different aspects. However, the resultant integrated model often has some defects that are difficult to detect on each individual model such as omissions on entity data life cycle.

This paper proposes a support method to exhaustively check the validity of data lifecycle for the RA model in UML by using a model checking tool "UPPAAL" [8]. Exhaustive checking improves the quality of the RA model which are validated through automatically generated mock-up.

II. Problems in Applying Model Checking Techniques to Requirements Analysis

Model checking techniques allow us to exhaustively and efficiently check the model whether it satisfies the specifications in temporal logic formulas or not. Therefore, to introduce the model checking techniques into the interaction model is a promising trial because the interaction between a user and a system in an enterprise application is easy to become huge and complex.

Furthermore, the later the discovery of defects, the higher the cost of reworking becomes [16]. Therefore, it is important to discover the defects at an early stage of the development. However, there are some problems in applying model checking techniques to the requirements process.

A. The Lack of Supports of Intuitive Understanding of a Requirements Specification for Customers

Validation is very important for analysts to elaborate the requirements specification so that they can decrease the change of the specification at a late stage of the development. Therefore, it is necessary for analysts to shape the visualization of a requirements specification so that the analysts can make customers validate correctly and sufficiently the requirements specification.

However, it is not easy for the customers to understand a model and specification for model checking techniques because it has the formal expression which is unfamiliar to them. Therefore, it is difficult for them to understand correctly and to decide whether the requirements specification is valid or not by using model checking techniques.

Accordingly, it becomes an important issue that the method of combining with intuitive understanding support of the requirements specification, and utilizing model checking techniques is actualized.

In this paper, we try to solve above-mentioned issue by completely generating the model and specification for a model checking technique. To achieve this, we partly expand the notation of the RA model which employs the ability of automatic prototyping.

Prototyping [10] which creates a mock-up of a system such as user interface at an early stage of development is widely known as one of the effective methods to promote the validation. Accordingly, we have proposed a method to generate a mock-up of Web user interface [5, 6] from the RA model so that the customer can intuitively and easily validate the RA model of Web enterprise application through the mock-up. The RA model represents interaction between actors and a system in UML.

B. The Cost-Effectiveness of Model Management in Frequent Changes of Requirements

It is realistic to refine the requirements specification by iterative validation because it is rare for analysts and customers to completely understand the requirements from the beginning. Namely, we have to pay attention to manage the model as the specification because the change of the requirements and specification may often appear. Therefore, the following issues should be dealt with when we apply model checking techniques into requirements process.

(a) The cost of managing the requirements specification, the model and specification for the model checking techniques in parallel because of frequent change of the requirements is high.
(b) To validly and precisely create the model and specification for the model checking techniques according to ambiguous and incomplete requirements needs a high degree of skill.
(c) It is difficult to reuse the model and specification for the model checking techniques from other development if the model and specification are specialized to a specific domain.

We try to solve above-mentioned issues along the following plans.

For the issue of (a), we try to improve by automatically generating the model and specification for model checking techniques from the RA model so that the analysts do not need to directly manage these model and specification.

For the issue of (b), we try to solve it by automatically generating the model and specification for model checking techniques as same as above-mentioned. This implies that our method does not require the analysts to write the model and specification including the knowledge of the model checking techniques. To actualize this, we propose templates of the model and specification used by the generation.

For the issue of (c), we keep the generality of the templates high so that the analysts can use the templates regardless of depending on domains. One of such aspects whose generality is high is the lifecycle of data in CRUD.

CRUD is widely known as a fundamental unit of database operation. Furthermore, a CRUD table [12] is a major example showing that the concept of CRUD is utilized at an early stage of a software development.

The concept of CRUD is divided into two levels as follows so that we can enhance the generality of CRUD.

On the 1st level, it checks the validity of data lifecycle focusing on the existence of data. For example, it checks whether "the data always have to be created or read when the data is updated." The generality of this aspect is high because it is not depending on a specific domain. This aspect seems too simple and natural so that it does not need to ensure. However, we assume that to ensure the validity of this aspect is difficult because the management of the RA model tends to become complex and unclear by making a lot of analysts share the work in a large project. In this paper, we focus on the aspect of the 1st level.

On the 2nd level, it checks the validity of data lifecycle focusing on how to change the data before and after CRUD. This aspect is easy to depend on a specific domain by according to a business rule, a law, etc. Therefore, the generality of this aspect is not high enough. In this paper, we do not handle the aspect of the 2nd level because the aspect of the 1st level should be ensured at the first.

III. Requirements Analysis Model in UML and Automatic Mock-Up Generation

At requirements analysis phase, developers extract requirements for a system from customers and generally specified them by defining semiformal documents. Recently, many developers have been getting to use UML, so that requirements specifications can be defined more formally. We have proposed a method of model-driven requirements analysis using UML.

We analyze functional requirements of services as well as service analysis. Especially, because what customers essentially want to do obviously appear within the interaction between a user and a system, our method proposes to clearly model the interaction.

To put it concretely, we specify business process as a service from the following four viewpoints.

- Based on the business rules, what kinds of input data and the conditions are required in order to execute a service correctly?
- To observe the business rule, what kinds of conditions should be required in case of not executing the service? Moreover, how the system should treat these exceptional cases?
- According to these conditions, what kinds of behaviors are required in order to execute the service?
- What kinds of data are outputted by these behaviors?

Based on the above mentioned four viewpoints, both business flow and business entity data which are required to execute the target business are defined by activity diagrams and a class diagram in UML.

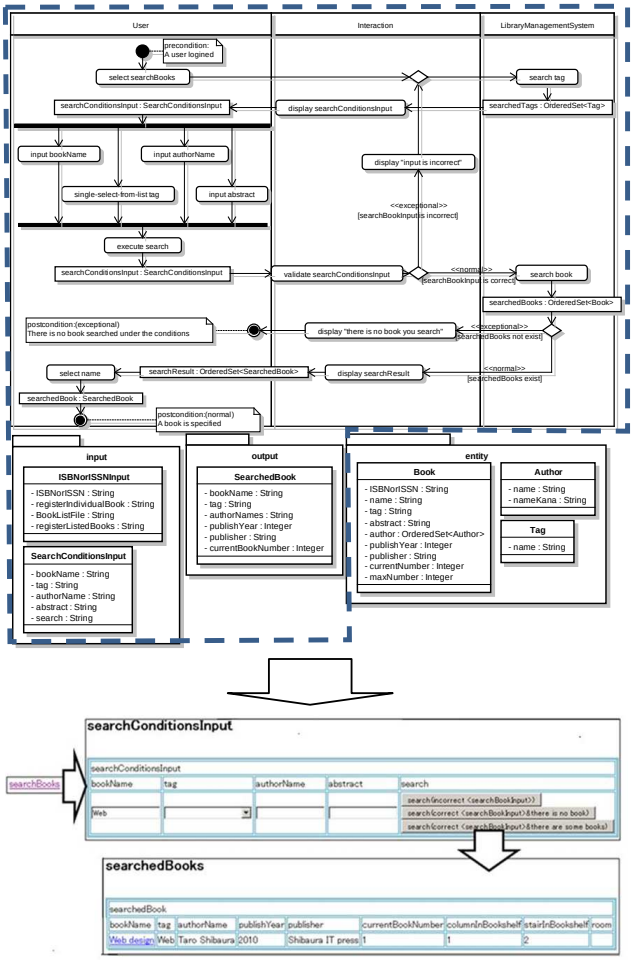

Fig. 1 UI mock-up generation from RA model

An activity diagram specifies not only normal and exceptional action flows but also data flows which are related with these actions. An action is defined by an action node and data is defined by an object node being classified by a class which is defined in a class diagram.

Accordingly, these two kinds of diagrams enable us to specify business flow in connection with the data. This is one of the features of our method on how to use activity diagram and class diagram. Especially, the interaction between a user and a system includes requisite various flows and data on user input, conditions, output to execute a service correctly.

The second feature is that an activity diagram has three kinds of partitions being named User, Interaction, and System. This is because that these partitions enable us to easily recognize the following activities; user input activities, interaction activities between a user and a system which are caused by the conditions to execute a service, and the resulted output.

The third feature is that we use an object diagram to define concrete data for each activity, because concrete valid data make it easy for us to confirm business process.

The fourth feature is that a mock-up which consists of Web pages written in HTML is automatically generated from these three kinds of diagrams. Fig. 1 shows an image of mock-up generation. The mock-up which is a kind of final product model enables the customers to confirm plainly and easily the requisite business flows in connection with the data. The generated mock-up describes the required target system except user interface appearance and internal business logic processing. Moreover, the mock-up enables the developer to confirm and understand the correspondence between his/her models and the final system. The developer defines three kinds of diagrams along requirements analysis from such different viewpoints as action flows, data flows and the structure, and the concrete values. The automatically generated mock-up enables him/her to easily understand the consistency between his/her models and the target system. To be able to fully understand the correspondence between each diagram and the target system, a mock-up can be generated whenever the developer want to confirm at the requirement analysis phase. The requirement analysis model is defined by using the astah[11] of a modeling tool.

IV. Proposal of How to Automate the Checking of the Validity of Requirements Analysis Model

Our proposal targets to the development of the interactive Web system that deals with entities as the core of the system such as enterprise application. And the phase to apply our proposal is requirements analysis from the viewpoint of checking the feasibility of functional requirements. Also, checking the validity of the RA model focusing on CRUD is needed to step to fundamental design phase.

We explain our method from the following three aspects after depicting the outline of our proposal.

- Architecture of our CASE tool to support the checking for the validity of the RA model focusing on CRUD.
- Notation to identify CRUD actions in the RA model.
- A Template of the model in UPPAAL which is a model checking tool, in order to support the automation of use of UPPAAL.

At the first, we define a glossary which identifies misleading terms. The RA model is the UML model we have proposed. A UPPAAL model is the model of the system in the format needed by UPPAAL. A UPPAAL specification is the specification to check the validity of the UPPAAL model, which is represented as formulas of CTL [9] (Computational Tree Logic) format.

A. Outline of Proposal

Unfeasible definitions of business logic in the RA model cause critical reworking in the later stage of the development process even if the RA model allows the analysts to capture desirable interaction more precisely and validly.

Accordingly, we propose a method to automatically check the validity of the lifecycle of the data which change in state by CRUD so that such unfeasible business logic can be automatically detected from defined interaction at an early stage of the development process.

In this context, valid data lifecycle means that CRUD actions are properly performed depending on the situation in which the data to perform CRUD exist or not. In valid data lifecycle, for example, certain data is always created or read before the business logic updates its data. Benefits by focusing on the data lifecycle on CRUD according to the existence of data are explained as follows.

Firstly, analysis techniques of CRUD is reusable enough because the techniques such as a CRUD matrix [] were utilized over wide variety of business domains in the past. In essential, we try to utilize techniques whose generality are high so that we can reuse the techniques without essential change of them depending on the kind of business domain.

Secondly, Focusing on the existence of data allows us to decompose the complexity of functional requirements and to concentrate to define and understand fundamental of the data lifecycle.

This decomposition is very important to alleviate difficulties of the process in which we derive the rigorous and correct model from ambiguous and incomplete requirements. It is meaningful to define detailed contracts such as pre/post-conditions including concrete side effects but this definition from the beginning brings us thorny confusion and workload when we refine or maintain these contracts. The definition focusing on the existence of data can become the basis to detail such contracts.

B. Architecture of a CASE Tool to Automatically Generate the UPPAAL model and UPPAAL specification

Fig. 2 shows the architecture of our CASE tool which enables us to automate the use of a model checking tool "UPPAAL". To be precise, a user of the tool needs to understand the output of the UPPAAL. The tool is developed with Java, C# and the astah API which can get the information of the RA model from the astah file.

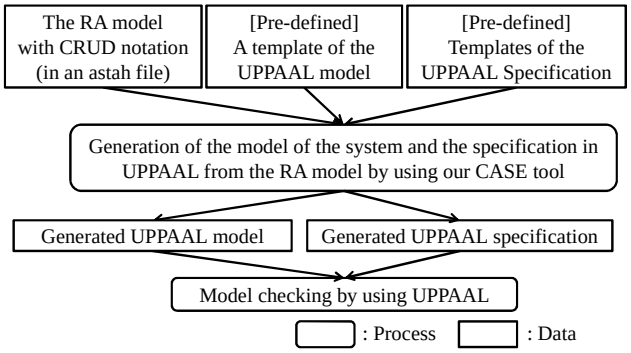

Fig. 2 Architecture of proposal

TABLE I
ASSOCIATIONS BETWEEN A CRUD TYPE AND VERBS

Type	Verbs
C	create, generate
R	read, get, search
U	update, add, insert, change
D	delete

Our CASE tool requires two inputs of a user. One is each name of *System* partitions. The other is the RA model whose notation is extended for CRUD. Also, both of a template of the UPPAAL model and templates of UPPAAL specification are needed by our CASE tool. And, the user does not need to explicitly input these templates. The template of the UPPAAL model is a thing to depict data lifecycle on CRUD, which is called by the UPPAAL model generated from the RA model when we perform model checking by using UPPAAL. The templates of the UPPAAL specification are things to check the validity of data lifecycle, which are automatically arranged as corresponding to the RA model by our CASE tool.

Then, our CASE tool outputs UPPPAAL model and UPPAAL specification. The user does not really need to define these artifacts but needs to operate UPPAAL in order to confirm the results of model checking.

From next sections, we explain about the CRUD notation, above mentioned templates and how to confirm the results of model checking.

C. Notation of CRUD

We propose the notation of CRUD to identify the data lifecycle on CRUD in the RA model. Accordingly, CRUD actions and entity data are needed to be identified. To realize such identification, we extend the notation of the RA model which handles a part of classes as entity and a part of actions as business logic. The notation is extended based on UML notation so that original one is not violated. This extension is conducted by two ways. One is by the stereotypes which is UML standard extension. The other is by limitations of the terms which represent actions, object nodes and the guard condition of specific branched flows. This limitation is quite simple and natural to represent the data lifecycle on CRUD.

We have ever proposed a simple format for the actions in order to avoid misreading of them by developers. Concretely, the format is represented as "behavior (as verb) object (as noun)." An example of the format is "create book."

In this paper, we propose a new categorization of verbs, an interpretation of the relationship between actions and object nodes and new stereotypes in order to identify elements related to CRUD in the RA model.

Fig. 3 shows an example of the activity diagram with CRUD notation. This activity diagram depicts a typical login service. The CRUD notation is applied only to the *System* partition because entities are handled by business logic only.

Table I shows the category of verbs corresponding to the CRUD type. The "search User from inputtedAuthentification" action, for example, is categorized to the type of "R".

Then, the interpretation of the relationship between actions and object nodes is explained. The result of above-mentioned "R" action is depicted as the object node immediately after this action. The same interpretation is applied to the "C." In the case of actions of "U" or "D", the object of each action has to be defined as corresponding to the name of the object node as the target.

For example, if we need to update "searchedUser" in Fig. 3, its action will be represented as "update searchedUser."

Table II shows the stereotypes we propose. In general, an action of "R" often provides the null value. The oversight of this makes analysts create invalid data lifecycle. Therefore, we propose the stereotypes to avoid such oversights. The "nullable" stereotype implies the possibility to return the null value. For example, an action without the "nullable" stereotype implies that some values exclusive of the null value are inevitably returned. In this case, the data needed as the result has to be created by an action of "C" before such read action. In contrast, if a read action has the "nullable" stereotype, the analysts should define the proper activity for the null value after such action.

Furthermore, there is one of the kinds of conditional branch in order to check whether the value is null or not. It is necessary to identify such condition in order to determine whether above-mentioned "proper activity" is valid or not. Accordingly, we propose the identifier of such condition for the guard condition of branched flow. When the data represented as an object node has one or more values, the guard condition ends with "exist." When the data has the null value, the guard condition ends with "not exist." For example, the guard conditions such as "searchedUser not exist" and "searchedUser exist" in Fig. 3 imply that the flows are branched depending on the existence of the "searchedUser" data.

TABLE II
STEREOTYPES RELATED TO CRUD

Element Type	Stereotype	Description
Action of "R"	nullable	The read action which has this stereotype implies that the null value may be returned.
Class	constant	The data of the class which has this stereotype implies that the data is already created by other external systems or else.

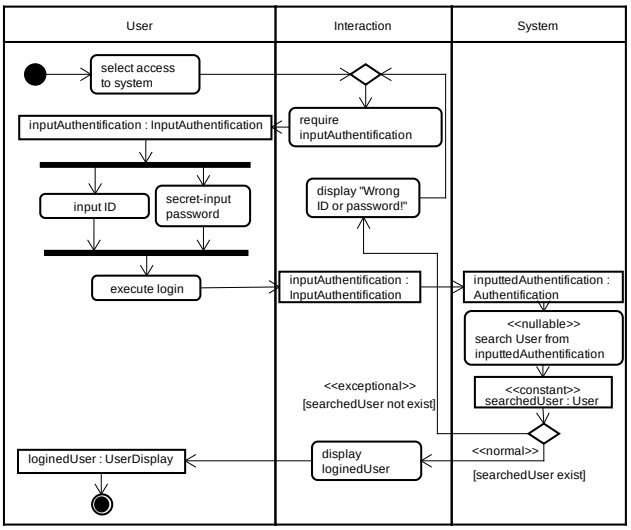

Fig. 3 An activity diagram of login service

D.The Template of the UPPAAL Model and Generation of the UPPAAL Model

The UPPAAL model is expressed as finite state machine i.e. the state of data can be recorded based on the data lifecycle focusing on CRUD. The purpose of the template of the UPPAAL model is to represent the data lifecycle in such state as the finite state machine.

The template includes three types of finite state machines. Firstly, it represents the change in state by performing CRUD actions shown as Fig. 4. The state in this machine transits by messaging from the state machines corresponding to the service shown as Fig. 7. The state machine in Fig. 7 is not in our pre-defined templates but can be generated from the activity diagram in Fig. 1 by using our CASE tool. For example, the message of "r_objnul[3][1]!" in Fig. 7 which corresponds to the read action of "search tag" in Fig. 1 is sent to the state machine in Fig. 4. In response to this message, the state machine in the Fig. 4 transits from "START" to "Pre_Read." Finally, the state in the Fig. 7 reaches "START" with the flag which records that the data was read. The indexes of "[3][1]" are the things to identify each data. For example, "[3][1]" implies the first data of the "Tag" class whose identifier is 3.

Secondly, it represents the existence of data for each class shown as Fig. 5. The state in this machine reaches to "EXECUTE" when data of a certain class is created. Similarly, there are the state machines for the existence of each data on creation and on read.

Thirdly, it represents the possibility of whether the return value is null or not, depending on the "nullable" stereotype shown as Fig. 6. This machine randomly decides whether each return value of read actions is null or not; but the return value for data of the class which is never created is inevitably null.

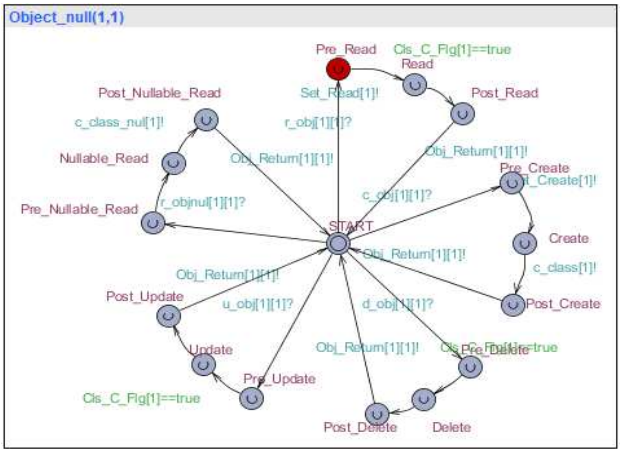

Fig. 4 The state machine on CRUD in the template

To avoid state explosion as much as possible, we shapes how to generate the UPPAAL model such as Fig. 7 as follows.

- The action nodes which have no relation with CRUD are removed through the generation process.
- Meta type which UPPAAL does not count state of variable of is used as much as possible.

- The routine to forcibly break out of the infinite loop is added through the generation process.
- The sequent transitions over different machines by utilizing synchronous messages are actualized in order to reduce the combinations of state that increase by allowing concurrent transitions. Concretely, each machine is created so that it always waits for the return message from another one immediately after sending the message to it.

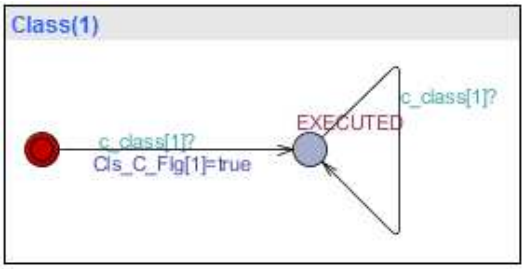

Fig. 5 The state machine of the existence of data in the template

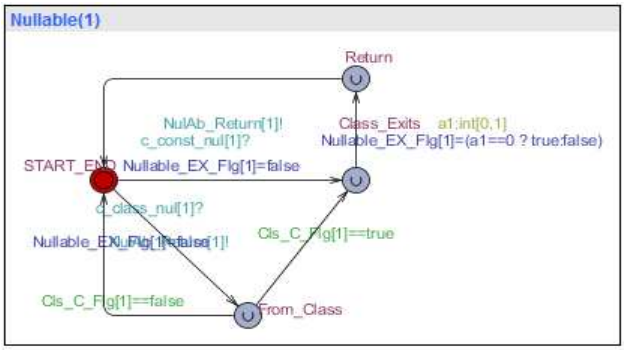

Fig. 6 The state machine in the template to determine whether the result of a read action is the null value or not

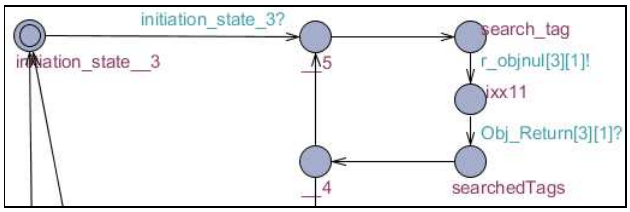

Fig. 7 The state machine which is generated from the activity diagram in the RA model

E. The Templates of the UPPAAL Specification and Generation of the UPPAAL Specifications

We propose UPPAAL specifications which check whether the data lifecycle of each entity is valid or not from the viewpoint of the existence of data. Concrete problems which can be detected by using generated UPPAAL specifications are explained as follows.

Oversight of the possibility of "null": The analyst should pay attention to correctly suppose the "null" value so as not to introduce invalid data lifecycle. For example, the data to be read must exist when the read action without the "nullable" stereotype is performed. We propose templates of the UPPAAL specification such as "A[] Object_null(*m*,*n*).Pre_Read imply Class(*m*).EXECUTED" so that analysts can check the validity of data lifecycle in above-mentioned situation. The parameters of *m* and *n* are the identifier of classes and object node respectively explained at IV.D, which are automatically decided by our CASE tool according to the RA model.

Invalid data lifecycle: To create or read data is needed in order to achieve updating or deleting its data. Although it is a very simple principle, it seems that it is difficult to keep the completeness of data lifecycle when the project makes a lot of analysts share the work and when the requirements are frequently changed. Accordingly, we propose templates of the UPPAAL specification to ensure "the data are created or read at some services or external systems before the action for updating or deleting is performed." Exactly, the read action has to provide one or more data but not the null value in this sense. An example of the template is "A[] Object_null(*m*,*n*).Pre_Update imply Create(*m*).EXECUTED or Read(*m*).EXECUTED."

We can get numerous UPPAAL specifications shown as Fig. 8 at low cost by generating the specifications based on above-mentioned templates.

After the generation of the UPPAAL model and UPPAAL specifications, the analysts can confirm the result of check and the counterexamples by using UPPAAL without defining any model and specification.

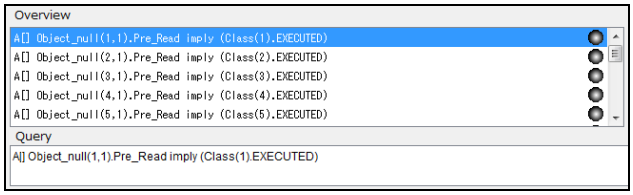

Fig. 8 The UPPAAL specifications in the UPPAAL verifier

V. Preliminary Evaluation

To evaluate the effectiveness of our method, we conducted preliminary evaluation through small case studies. Then we have evaluated the effectiveness of our method by comparing each data obtained by two kinds of methods.

Firstly, we have compared the difference of the time taken by manual review and by using our method because we expect to decrease the time by using our method.

Secondly, we have measured the rate of recall and precision on detected defects to pre-defined defects because we expect to improve the correctness of detecting defects than manual review. Finally, we discuss the effectiveness and correctness of our method.

A. Case Study

We apply our method into two small projects which are the development of library management system (LMS) and of a supportive sales system for text books (STB) in a university. At the first, the RA model of each project was manually created. The RA model of LMS was defined by one analyst.

The RA model of STB was defined as an exercise of the class of software engineering by three graduate students. These systems are assumed as Web enterprise application.

TABLE III
SCALE OF LMS

Use cases	Number of actions			
	User	Interaction	System	**Total**
Borrow books	3	6	6	15
Return books	8	7	4	19
Confirm history of borrowing books	3	5	1	9
Search books	7	5	2	14
Register books	25	17	17	59
Total	46	40	30	116

TABLE IV
SCALE OF STB

Use cases	Number of actions			
	User	Interaction	System	**Total**
Browse results of questionnaire	3	3	3	9
Make and edit a text book purchase list	25	12	28	65
Reserve receipts	11	10	8	29
Make and edit a purchase plan list	32	20	15	67
Browse reservation receipts	5	5	4	14
Process purchase	6	9	9	24
Total	82	59	67	208

TABLE V
THE NUMBER OF DEFECTS PUTTING IN THE RA MODEL

Defect types	LMS	STB
Oversight of the possibility of "null"	5	4
Invalid data lifecycle	2	6
Total	7	10

These analysts did not have the experience of using UPPAAL enough. At least, one analyst was not able to correctly write the UPPAAL model and UPPAAL specification.

The other was able to manage to correctly write the UPPAAL model and UPPAAL specification according to its tutorial but did not have the experience of using UPPAAL in any software development.

There is each customer for these systems. Then, they validated the RA model through the mock-up which was generated form its model.

Table III and IV show the scale of these projects. The kinds of actors in LMS are one. In STB, these are three. The kinds of entities in LMS are 6. In STB, these are 8.

B. Steps of Evaluation

We have conducted the evaluation along the following steps. Two analysts as participants were decided in by selecting one analyst from each project.

Each analyst put some defects shown as table V into his own RA model. Concretely, they put defects by removing "nullable" or "constant" stereotypes or CRUD actions from valid RA model. Then, they exchanged the RA model each other.

Each analyst manually discovered the defects and recorded the time at each time when he discovered a defect. As how to record the defects they suspected, they give a note to each suspect action in astah.

After the manual review, each analyst automatically detects the defects by our method. Also, they recorded the time in the same way of the manual review.

In this evaluation, we let the analysts find the defects by focusing on only the existence of data of which the generality is high so that the point of view of checking can be kept fair between the manual review and our method. Each analyst was not able to sufficiently understand the model he reviewed because he did not relate to the project which created its model. However, such situation often appears in the large scale development in which it makes a lot of analysts share the work. Therefore, it is natural for them to review the model by focusing on the viewpoint which has high generality. On the other hand, we did not make them refine the RA model because they were not able to exactly understand the requirements as the background of the RA model.

C. Result and Consideration

The analysts were able to detail the RA model without the modification of a part of the RA model which is transformed to the UI mock-up. Therefore, our checking method was able to be completely combined with the analysis method which supports the validation by generating the UI mock-up.

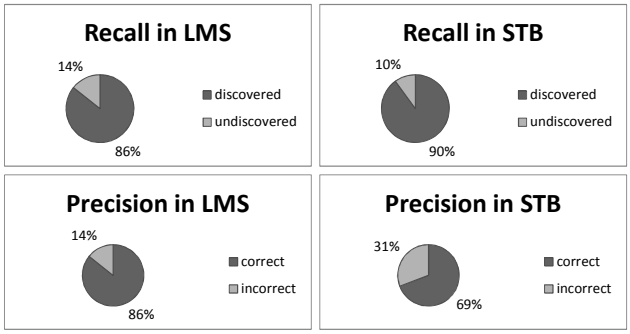

Fig. 9 The rate of recall and precision in manual review

Fig. 9 shows the rate of recall and precision of discovered defects in manual review. On the other hand, the recall and precision in our method is 100%.

This reason is that the defects put were adjusted so that our method can discover all defects. We consider that this setting does not affect the result about whether the analysts can correctly and exhaustively discover all defects or not. On the other hand, this setting is inadequate if the ability of our method for detecting the definition which will become the defects potentially is evaluated.

As a preliminary evaluation, we focused on whether the analysts can correctly and exhaustively discover all defects that our method focuses on than manual review because we wanted to evaluate the potential of our method for efficiency, easiness and effectiveness.

As the result, the rate of recall and precision in the manual review was decreased because it was perhaps difficult for the modelers to correctly imagine the situation which violates the validity of data lifecycle even if the projects were small scale.

The activity diagrams can explicitly represent object nodes but the data lifecycle of concrete data level cannot be visualized on the diagram. What is worse, the activity diagrams were a little complex so that the analysts can not completely and correctly trace flow in manual because the services can be variously used by users. For example, the user of LMS can arbitrarily perform the service of "borrow books" regardless of that the "register books" is called or not. Any books which the user wants to borrow may not exist if the "register books" is not called. The modelers were required to manually imagine such situations and had to specify the defects.

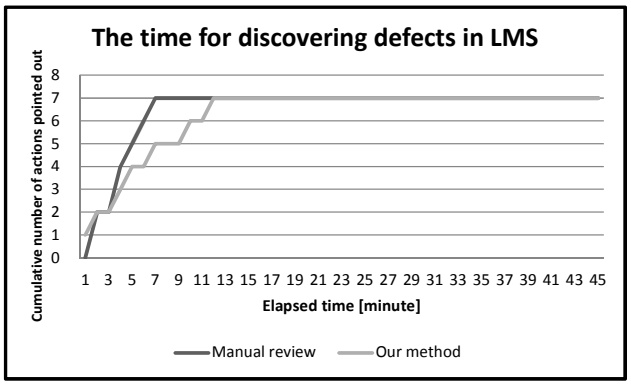

Fig. 10 The time for discovering defects in LMS

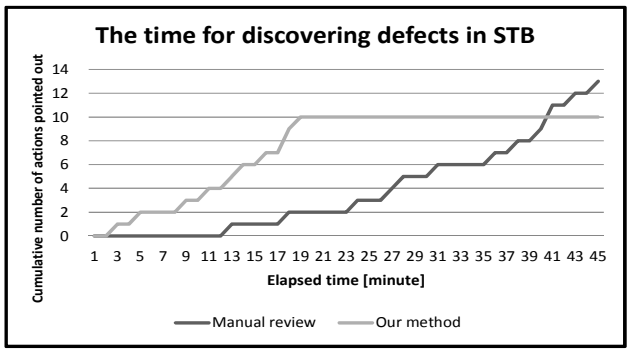

Fig. 11 The time for discovering defects in STB.

Fig. 10 and 11 show the time spent by discovering defects by each way. Cumulative number of actions pointed out implies the cumulative number of points by the analyst or of points by using UPPAAL. The elapsed time implies the time elapsed after starting of checking. The time of detecting a defect by using our method implies the time from generating the UPPAAL model and UPPAAL specifications to pointing out the incorrect action. Such time in the manual review is the time from manually searching the defect to pointing out the suspect action.

The time of the manual review was earlier than our method in LMS. On the other hand, the time of our method was earlier than the manual review in STB.

According to this result, our method can efficiently detect defects as increasing the scales of the RA model. However, for too small project, the overhead of generating the UPPAAL model and UPPAAL specifications and of utilizing of UPPAAL using the generated model and specifications was high than the manual review.

The analyst of STB said that "In manual review, I had to point out the defects after I grasped the data lifecycle of a certain entity through the entire services by focusing on the entity. Therefore, I spent a lot of time in some cases for pointing out a defect." In our method, the analysts did not need to grasp the entire services because the model checking tool exhaustively checks the RA model. Furthermore, the analysts needed less time for detecting defects because the UPPAAL model and UPPAAL specifications were able to be automatically and completely generated.

As a problem of our method, the analysts needed to iteratively use our CASE tool in order to completely remove defects because the model checking tool can depict only one counterexample for each UPPAAL specification even if there are the one or more defects which can be detected by the same UPPAAL specification at one time. Therefore, we try to improve the architecture of the CASE tool which can round trip support so that the analyst can efficiently utilize our method.

Another problem is the scale of this case study. To show the significant effectiveness of our approach is needed to apply it into larger projects. However, we showed an effectiveness of our method from the aspects of the time for discovering defects, the rate of recall and precision and the analyst's opinions.

VI. Related Work

A. Variation of Checking Aspects

CRUD is widely known as the useful viewpoint in order to effectively clarify and check the specification. CRUD table [12] often is used for analysis. CRUD table can simply visualize the relation between the behavior and data so that analysts are easy to understand its relation. For example, the behavior means services, functions or actions. Also, the data means classes which includes entities, or attributes.

However, CRUD table cannot capture the relation between the behavior and data on the flow. Therefore, we cannot use the CRUD table to check the validity of the relation from the aspect of action sequence. Also, CRUD table shows the CRUD operation if its operation appeared in the behavior at least one time. However, its operation may not be performed through a certain path as a result of user's operation. We cannot understand whether such situation exists or not, by only using CRUD table.

In our method, the CRUD operation is represented in the interaction flow which is depicted in the RA model. Therefore, the RA model can resolve above-mentioned problems. Furthermore, by using model checking techniques, we can exhaustively and efficiently find more problems than what we can find by grasping at CRUD table only.

B. The degree of abstraction of the Specifications

In the method proposed by Li et al. [13], the specifications are created as specialized to a specific domain. In this way, the specifications is useful for its domain but cannot be reused for other domains. The specifications generated in our method can be reused to various domains because we focused on the aspect whose generality is high.

Such aspect is the data lifecycle focusing on the existence of data on CRUD. In the method proposed by Sciascio[14], the specifications used in a model checking tool can be exhaustively created as combination which can be considered according to existing specification. Such specifications are useful from the aspect of the support of regression testing. However, it seems that it is difficult to apply it to an early stage of the development in which requirements frequently change because the software may not exist and the specification may not always be valid and correct. In our method, the proposed UPPAAL specifications ensure absolute correctness by deriving based on fundamental principle of the data lifecycle from the aspect of the existence of data. Furthermore, we can combine our method with the analysis method which can supports validation by generating the UI mock-up.

C. Controlling State Explosion

There are a lot of challenges to control the state explosion. One of how to control is the abstraction of the model of the system and the specifications. In this context, the model of the system implies requirements specification, design specification, program, etc.

Corbett et al. [15] control the state explosion by slicing techniques and the engine for performing abstraction. Concretely, they perform abstraction as follows. Firstly, unrelated components for the specification are removed. Secondly, data abstraction is performed. Finally, they conduct to limit the components which are used in checking. On the other hand, the analysts are required the task such as selecting the model of the system and so on.

In our method, the UPPAAL model is degenerated as leaving only the sequence of CRUD actions in the *System* partition when the UPPAAL model is generated from the RA model. Furthermore, the analyst does not need the knowledge of abstraction of the model because he does not manually adjust the UPPAAL model.

VII. Conclusion

In this paper, we proposed a support method to check the validity of the data lifecycle focusing on CRUD by combining UML and UPPAAL. One of main features of our method is that we can receive support not only of verification but also of validation by using the RA model. The other is that our CASE tool can completely generate the UPPAAL model and UPPAAL specifications from the RA model. As the result of preliminary evaluation, the effectiveness of our method in applying it to larger project was confirmed. Concretely, the time for discovering defects was reduced. Also, the analysts could exactly detect the defects at low cost.

As the future work, we plan to evaluate our method by applying it to larger projects of the development of enterprise application. Also, we improve the CASE tool for iterative usage. Furthermore, we consider how to actualize more steps in “stepwise” support e.g. we focus on the attributes of a class, so that we can more rigorously and particularly check the model.

References

[1] Paulo, Rogerio; Carvalho, Adriano,Towards model-driven design of substation automation systems, 8th International Conference and Exhibition on CIRED, pp.1-5, 2005.
[2] Monteiro, R.; Araujo, J.; Amaral, V.; Patricio, P., Mdgore: Towards Model-Driven and Goal-Oriented Requirements Engineering , 18th IEEE International Requirements Engineering Conference , pp. 405-406, 2010.
[3] Forward, A.; Badreddin, O.; Lethbridge, T.C. Towards combining model driven with prototype driven system development, 21st IEEE International Symposium on Rapid System Prototyping (RSP), pp.1-7,2010.
[4] Rational Software Modeler, http://www-06.ibm.com/software/jp/rational /products/design/rsm/.
[5] S. Ogata, and S. Matsuura, “A UML-based Requirements Analysis with Automatic Prototype System Generation,” Communication of SIWN, Vol.3, Jun. 2008, pp.166-172.
[6] S. Ogata. and S. Matsuura, “A Method of Automatic Integration Test Case Generation from UML-based Scenario,” WSEAS TRANSACTIONS on INFORMATION SCIENCE and APPLICATIONS, Issue 4, Vol.7, Apr 2010, pp.598-607 .
[7] UML, http://www.uml.org/
[8] UPPAAL, http://www.uppaal.com/, 2010.
[9] Thomas A. Henzinger. Symbolic model checking for real-time systems. Information and Computation, 1994, 111:193-244.
[10] ACM SIGSOFT, Special Issue on Rapid Prototyping, ACM SIGSOFT Software Engineering Notes, Vol.7, No.5, 1982.
[11] astah*, http://www.change-vision.com/
[12] van den Brink, H.; van der Leek, R.; Visser, J., Quality Assessment for Embedded SQL, Proc. of Seventh IEEE International Working Conference on Source Code Analysis and Manipulation, 2007 (SCAM 2007), pp.163-170, 2007.
[13] Li, H., Krishnamurthi, S. and Fisler, K.: Verifying cross-cutting features as open systems, in international conference on Foundation of Software Engineering ,2002
[14] Sciascio, E. D., Donini, F. M., Mongiello, M., and Piscitelli, G.: Web Applications Design and Maintenance Using Symbolic Model Checking, Proc. of the 7th European Conference on Software Maintenance and Reengineering (CSMR 2003), 2003, pp. 63?72.
[15] Corbett, J., Dwyer, M., Hatcliff, J., Laubach, S., Pasareanu, C., Robby and Zheng, H.: Bandera: extracting _nite-state models from Java source code, Proc. the 22nd Int'l Conf. on on Softw. Eng. (ICSE 2000), pp.439-448 (2000).
[16] Wiegers, K. E., Software Requirements, Microsoft Press, 2003.