Evaluation of a Use-Case-Driven Requirements Analysis Tool Employing Web UI Prototype Generation

SHINPEI OGATA
Course of Functional Control Systems,
Graduate School of Engineering
Shibaura Institute of Technology
307 Fukasaku, Minuma-ku Saitama-City,
Saitama 337-8570
JAPAN
m709101@sic.shibaura-it.ac.jp
http://www.sayo.se.shibaura-it.ac.jp/

SAEKO MATSUURA
Department of Electronic Information Systems,
College of System Engineering and Science
Shibaura Institute of Technology
307 Fukasaku, Minuma-ku Saitama-City,
Saitama 337-8570
JAPAN
matsuura@se.shibaura-it.ac.jp
http://www.sayo.se.shibaura-it.ac.jp/

Abstract: - It has been widely acknowledged that ambiguous or incomplete customer requirements are a major reason for the failure of software development projects. A key example of this is the inconsistency between implementation image and specifications provided in the early stages of software development. To address this, we propose a method for automatic generation of user interface prototypes for developing Web-based business applications, based on the requirements specifications defined in the Unified Modeling Language (UML). In this study, we compare the proposed method with traditional use case modeling to evaluate the effectiveness of the proposed method.

Key-Words: - Use Case, Prototyping, Web Application, Requirements Analysis, Unified Modeling Language

1 Introduction

It has been widely acknowledged that a major reason for the failure of software development projects is the ambiguity or incompleteness of customer requirements. Furthermore, data structures and data flows created while developing business systems are often quite complex. As customers are typically familiar with business rules such as data format and calculation procedure, they are capable of evaluating the adequacy of the requirements specification, thus reducing the degree of ambiguity and incompleteness associated with the requirements.

A use-case-driven requirement analysis is a typical, object-oriented approach for analysis of the system specification, which defines how the system should interact with users and other, external systems. A use case model generally consists of a use case diagram, created using Unified Modeling Language (UML) [1], and several use case templates [2, 4].

There are two advantages of employing use-case-driven requirements analysis. Firstly, the use case templates are written in natural language and tend to avoid the use of technical jargon pertaining to the implementation, so that customers are able to understand the behavioral aspect of users and system more easily. Secondly, a use case model is required to focus on modeling of interactions, thus, it contains a large number of core requirements, which, in turn, are the basis of user interface (UI), performance, data format, and business rule requirements. As a result, researchers in a number of fields, including requirements engineering, requirements elicitation, and requirements analysis, validation, and verification, acknowledge the effectiveness of the use case approach [5–9, 13, 14].

If a UML model precisely defines the customers' requirements, it can be treated as a requirements specification. However, there are three primary problems associated with precisely defining the interaction specifications in the form of requirements.

Firstly, although customers are capable of understanding non-technical terms, it is difficult for the customers to precisely understand the image of manipulating the system from these terms. For example, it is difficult for them to understand the means of input, or the concrete input/output data. Secondly, the use of natural language for describing the system leads to varied interpretation of the actual system image by the developers and customers; natural language is likely to create misunderstandings between the developers and customers. Thirdly, classes are generally derived from the natural language terms that are used in the use case templates; thus, it is often difficult for the developers to derive these classes because these terms can refer to various instances of a single concept.

To address these issues, we propose a method to automatically generate the UI prototype from a requirements analysis model (RA model) that has been defined in UML for Web-based business applications development [10, 16]. An RA model typically consists of activity diagrams, a class diagram, and object diagrams. The UI prototyping is a requirements analysis technique that allows one to easily understand a system image and to implement specific aspects the system, such as the UI [11].

In our proposed method, the first and second problems, identified above, are resolved by clarifying the correspondence between the RA model and the implementation image via a UI prototype. In addition, the third problem is resolved by explicitly relating a classifier of object nodes within the activity diagram, a class within the class diagram, and a classifier of instance specifications in the object diagram.

In this study, we evaluate the effectiveness of a requirements analysis method that uses our automatic prototype generation tool by comparing with traditional use case modeling. In the evaluation, we measure the execution time of each method and the resulting consistency between the RA model and implementation image.

Section 2 provides an overview of traditional use case modeling and its limitations, and section 3 provides an overview of the proposed method, comparing it with the traditional method. Section 4 describes the experiment used for the evaluation and its results. Section 5 identifies a series of considerations based upon the results obtained in section 4, and section 6 provides a discussion of related work. Finally, section 7 addresses conclusions and suggests avenues of future work.

2 Traditional Use Case Modeling

In order to develop a system to satisfy customer requirements, it is critical to first specify sufficient and precise requirements. In particular, functional requirements are core to implementation of the system. According to a software requirement specification standard [3], the set of possible functional requirements includes the following:

- Validity checks on inputs
- Definition of an exact sequence of operations
- Responses to abnormal situations
- Effects of parameters
- Relationship of outputs to inputs

Most of these requirements pertain to interactions between actors and a system because, ultimately, customers are only capable of directly viewing interactions via the system's UI. In order to execute functions correctly, interaction requirements include the standard sequence of user operations, data flows, and constraints on the data. The constraints are represented as branch conditions. To achieve the desired requirements, a use case is developed [2]. A use case is a definition of the interaction process, defined in terms of the system's behavior from the actor's perspective. An actor is a role within the system, and is classified as either an authority or an external system. An authority is a role of a user from system perspective.

2.1 Use Case Model

The use case model represents interactions between actors and a system. It consists of a use case diagram, depicted in UML, and a use case template describing every possible use case. The typical process employed in generating a use case model is explained below.

Step 1. Definition of a use case diagram:
A use case diagram summarizes the system's intended behavior by describing use cases, actors, relationships between use cases and actors, and relationships between use cases.

Step 2. Definition of use case templates:
A use case template describes the details of the interactions that will take place for every use case. There are two basic principles to be followed while defining the templates. Firstly, the templates are not focused upon how to implement the system, but instead upon how to accomplish the business workflows of each authority. Secondly, the contents of the templates should be easily understood by the customers, as these are often used as a common point of reference for developers and customers to develop an understanding of the requirements. To achieve this, the contents of the templates must be defined using non-technical terminology and terminology that is employed by the customers within their business workflows. A use case template consists of a use case name, actors, preconditions, a basic flow, alternative flows, exceptional flows, and postconditions. The basic and alternative flows (normal flow) are the flows that accomplish the workflow under regular scenarios. On the other hand, exceptional flows are those used to recover from error conditions. The basic, alternative, and exceptional flows are defined as a sequence of steps and represent all interaction flows. A step represents the behavior of an actor or a system involved in the interactions. Though there is no one standard use case template, a number of methods such as Cockburn's template [4] or formal

grammars [6, 7] to define the templates have been proposed.

Step 3. Definition of a class diagram:
In the early stages of system formulation, the development team generates a class diagram in UML, based upon a set of data structures of conceptual levels. A class, in this sense, refers to a structured type having attributes, defined as data, and operations, defined as behaviors. In traditional use case modeling, classes are elicited based upon the range of data that can appear within the interaction flows, defined as part of the use case template.

2.2 Limitations of Use Case Modeling

There are three common issues faced while defining, sufficiently and unambiguously, the functional requirements of the system in terms of use case models.

Firstly, use case templates, targeted for comprehension by the customer, are often very ambiguous as a result of being simplified. The key points that must be conveyed to the customer by these templates, in our view, are the following:

(1) What customer business workflows do the various use cases address?
(2) Through what sequence of operations can the users execute the use cases?
(3) What alternative or exceptional flows the user might encounter because of the different viable inputs provided, either valid or erroneous?
(4) At each step in the flow, what data is a user able to input or confirm in the system? Data, in this case, includes the name, values, format, and structure.
(5) By what input means can the users input data into the system? For example, in a Web page, viable means of input include text fields, check boxes, radio buttons, etc.

These five aspects are inevitably required to ensure that the customers are provided with the correct support for executing their business workflows. Although the use case templates appear to address points (1), (2), and (3), they do not convincingly address points (4) and (5). In addition, if the templates define the details of their associated items, their contents become more complex. As a result, at the least, it becomes more difficult for the customers to interpret the templates.

Secondly, as mentioned previously, the gap that exists between the customer and the development team is one of the major causes of failure in software development projects. Moreover, the gap is largely the result of the specification being formed purely in terms of natural language. Each individual customer and developer independently formulates an arbitrary image of the ultimate goal for the implementation, based upon the natural language specification; this further widens the gap. Therefore, it is important that each of the above parties employ an implementation image to foster a precise understanding of the requirements specification.

Thirdly, whether developers can ensure the consistency between a use case model and a class diagram depends on their ability and experience. Generally, as described in section 2.1, classes are derived from terms, which represent data in use case templates. However, the developers often have a difficult time eliciting the optimal class definition because some of the terms in the use case templates may present various instances of the same class. As a result, it is a complex task to sufficiently and correctly derive the appropriate classes from these terms.

3 Use Case Modeling with Automatic Prototype Generation

3.1 Modeling Cycle in Requirements Analysis

In the proposed method [10, 16], a requirements analysis modeling cycle (RA model) is employed, as shown as Fig. 1. As the developers iterate through this cycle, they refine the RA model.

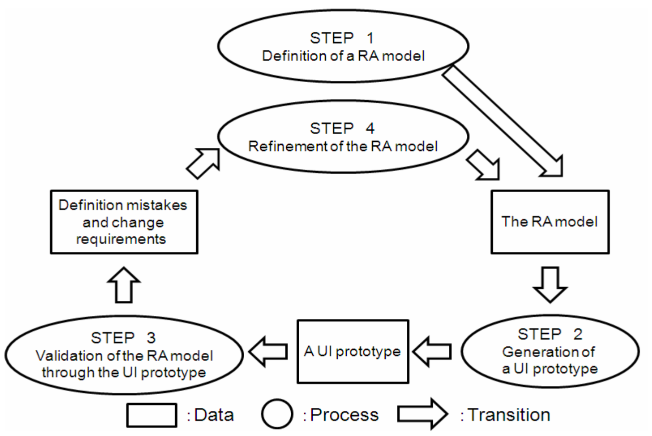

STEP 1
Definition of a RA model
STEP 4
Refinement of the RA model
Definition mistakes and change requirements
The RA model
STEP 3
Validation of the RA model through the UI prototype
A UI prototype
STEP 2
Generation of a UI prototype
: Data
: Process
: Transition

Fig. 1 Modeling Cycle to Define the RA Model

Step 1. Defining the RA model:
Developers define the RA model, which addresses the functional requirements of the system, using Astah* [13], a UML modeling tool. The RA model includes activity diagrams, a class diagram, and object diagrams. The model represents both behavior and structure, which addresses the interaction as well as traditional use case modeling. Astah* can precisely delineate relationships among

a class (within a class diagram), the classifier of object nodes within an activity diagram, and the classifier of instance specifications within an object diagram. The details of the RA model are provided in section 3.2. The output of this process is the formulated RA model. Non-functional requirements, such as response time and reliability, are incorporated into the RA model as notes.

Step 2. Generation of a UI prototype:
In the proposed method, we provide a UI prototype generation tool, which accepts the RA model as an input source. A UI prototype generated using this tool is just one component of the set of implementation views comprising the RA model. The purpose of the UI prototype is not to confirm the layout of the UI, but to validate the aforementioned five points that the customer must understand, discussed in section 2.2. The output of this process is a set of Hyper Text Markup Language (HTML) pages that do not perform any true functions.

Step 3. Validation of the RA model through the UI prototype:
There are two ways of validating the RA model via the generated UI prototype. One method entails the developers' validation and checking of the RA model, in an effort to determine whether the model is missing any data types or if there are any inconsistencies in the data between the behavioral model and structural model. The second method requires that the customers validate the RA model in an effort to determine whether the business workflows can be achieved using the expected sequence of operations, input/output data, and means of input, for the data identified in the generated UI prototype. A number of errors of the RA model, including mistakes in the definition and changes to requirements, may clarify after this step.

Step 4. Refinement of the RA model:
The developers correct the RA model on the basis of the errors discovered during the validation performed in step 3. The output of this process is the refined RA model.

The developers perform steps 2 through 4, iteratively, until the customer is satisfied with the generated UI prototype representing the RA model.

There are two key benefits to using the proposed method:

Firstly, a UI prototype can be generated at each stage of the cycle that incorporates the definition of diagrams following Fig. 2. The details of this definition process are provided in section 3.2. This enables the developers to clearly and easily understand the contents of the UI prototypes represented by each type of diagram.

Secondly, to specify concrete data that is expected to appear in a given scenario, a group of object diagrams is used to represent a specific situation in a workflow. In this study, a scenario is defined as the depiction of a traversal path over a series of interaction flows, including concrete input/output data.

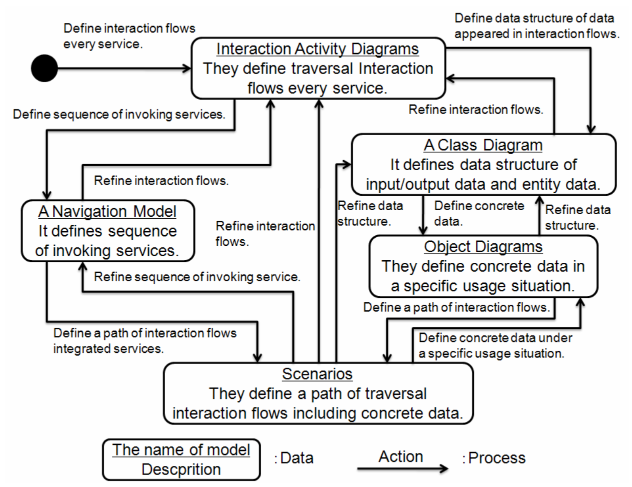

Define interaction flows every service.
Interaction Activity Diagrams
They define traversal Interaction flows every service.
Define data structure of data appeared in interaction flows.
Define sequence of invoking services.
Refine interaction flows.
A Class Diagram
It defines data structure of input/output data and entity data.
Refine interaction flows.
A Navigation Model
It defines sequence of invoking services.
Refine interaction flows.
Refine data structure.
Define concrete data.
Refine data structure.
Object Diagrams
They define concrete data in a specific usage situation.
Refine sequence of invoking service.
Define a path of interaction flows.
Define a path of interaction flows integrated services.
Define concrete data under a specific usage situation.
Scenarios
They define a path of traversal interaction flows including concrete data.
The name of model
Descprition
: Data
Action
: Process

Fig. 2 Definition Process of the RA Model

3.2 Definition and Validation of the RA Model

An overview of the definition process of the RA model is shown in Fig. 2. As initial input to the proposed method, a number of requirements documents are obtained from the customer and used by the development team to derive actors and initial use cases. The method by which each model is described is as follows:

Definition of Interaction Activity Diagrams:
An interaction activity diagram defines interactions between actors and a system. This is done using a UML activity diagram and a use case template. Note that the presented model does not extend the notation of the original UML activity diagram. We simply refer to this model as the interaction activity diagram in order to distinguish it from a navigation model of the RA model, which is also defined in the form of a UML activity diagram. An interaction activity diagram is defined for every service instance, which is typically more granular than a use case.

The interaction activity diagram follows the two aforementioned principles of use case template design: its definition focuses upon accomplishing business workflows, and it only employs non-technical terminology.

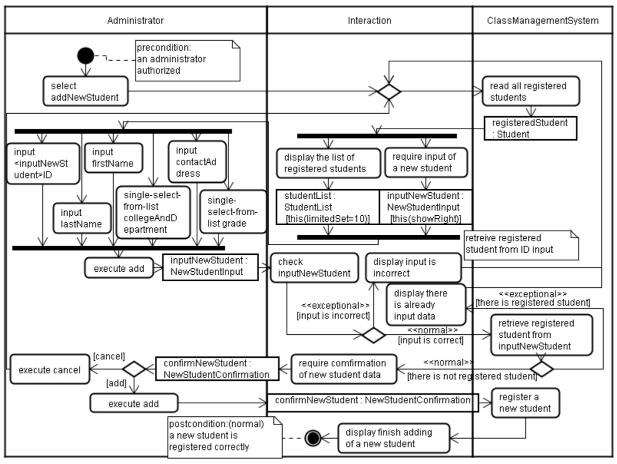

Administrator
Interaction
ClassManagementSystem
precondition: an administrator authorized
select addNewStudent
read all registered students
registeredStudent : Student
input <inputNewStudent>ID
input firstName
input contactAddress
input lastName
single-select-from-list collegeAndDepartment
single-select-from-list grade
display the list of registered students
require input of a new student
studentList : StudentList [this(limitedSet=10)]
inputNewStudent : NewStudentInput [this(showRight)]
retreive registered student from ID input
execute add
inputNewStudent : NewStudentInput
check inputNewStudent
display input is incorrect
display there is already input data
<<exceptional>> [there is registered student]
<<exceptional>> [input is incorrect]
<<normal>> [input is correct]
retrieve registered student from inputNewStudent
[cancel]
execute cancel
confirmNewStudent : NewStudentConfirmation
require comfirmation of new student data
<<normal>> [there is not registered student]
[add]
execute add
confirmNewStudent : NewStudentConfirmation
register a new student
postcondition:(normal) a new student is registered correctly
display finish adding of a new student

Fig. 3 Interaction Activity Diagram of "Add New Student" Service

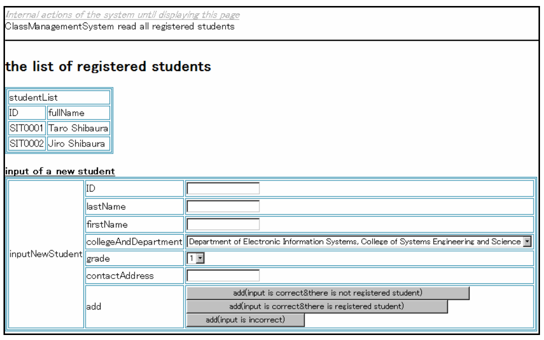
Fig. 3 depicts an interaction activity diagram that implements the "Add New Student" service of a class management system. This service enables the registration of a new student by providing confirmation of registration status. The system checks for the presence of duplicate data between new and existing students. Student data consists of an ID, last name, first name, college, department, grade, and contact address. We now define the five items discussed in section 2.2.

The sequence of operations that users can employ to execute services is represented as a series of actions, conducted by the actors or system, which are represented by partitions. In Fig. 3, the "Administrator" partition represents the authority. Other partitions represent different roles of the system. An initial node maintains pre-conditions to initiate the service, as a note. Final activity nodes maintain postconditions, satisfied immediately after termination of the service, again, as a note.

The alternative or exceptional flows that a user is capable of traversing are represented as conditional branches, using decision nodes. The conditions are defined at control flows as guard conditions, immediately following a decision node. To identify normal or exceptional flows, each control flow defines a stereotype that is <<normal>> or <<exceptional>>.

The data that a user is able to input or confirm is represented as an object node, which is directed toward an authority partition. For example, the "inputNewStudent" object node corresponds to data that is made visible for a user. The data structure is specified by defining a classifier of the object nodes.

The means by which users are able to input data are represented by the verb of actions at an authority partition. Fig. 4 depicts a page of a UI prototype that has been generated from the interaction activity diagrams, a class diagram, and object diagrams of the previously mentioned class management system. This diagram corresponds to the flow from the "read all registered students" action to the "check inputNewStudent" action, depicted in Fig. 3. For example, the "input lastName" action converts data into a text format. The "single-select-from-list grade" action converts data into a list box format. Lastly, the "execute add" action converts data into a button format.


Internal actions of the system until displaying this page
ClassManagementSystem read all registered students

the list of registered students

studentList	
ID	fullName
SIT0001	Taro Shibaura
SIT0002	Jiro Shibaura

input of a new student

inputNewStudent	ID	
	lastName	
	firstName	
	collegeAndDepartment	Department of Electronic Information Systems, College of Systems Engineering and Science
	grade	1
	contactAddress	
	add	add(input is correct&there is not registered student) add(input is correct&there is registered student) add(input is incorrect)

Fig. 4 A Page of Generated UI Prototype

We have realized both the definition of the use case in the form of a UML-based RA model, as well as validation of the UI prototype, which clearly corresponds with the RA model. Thus, in spite of defining technically detailed interaction flows, the customers can easily validate the RA model via a generated UI prototype. The gap between the understanding of the customers and the developers is thus reduced by the generation of a UI prototype that presents an implementation image of the system. The consistency between the behavioral model and structural model are guaranteed by the delineation of relationships between UML diagrams and a class, a classifier of an object node, and a classifier of an instance specification.

Definition of a Class Diagram:
A class diagram defines the structure of data in terms of a class at the conceptual level. An understanding of the extreme ranges of data, presenting the boundary of a system, is ultimately necessary to implement a business workflow. As such, classes are defined as the structure of object nodes within an interaction activity diagram. These classes may be categorized into one of two types: boundary candidates, a structure that is only required at the boundary between an authority and a system, or an entity candidate, a structure required for permanent system data. Fig. 5 depicts the classes involved in an interaction activity diagram, such as Fig. 3. The "Student" class is an entity candidate, while the other classes are boundary candidates.

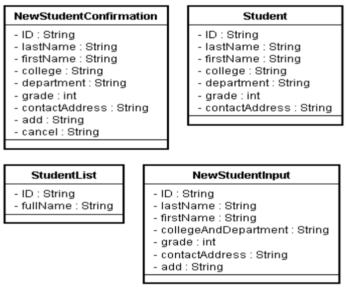
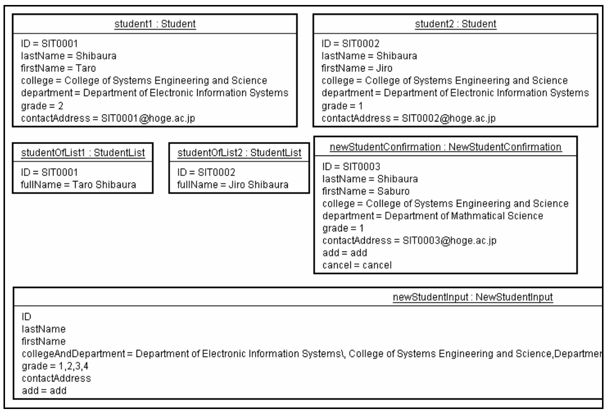


Fig. 5 Class Diagram for "Add a New Student" Service

When a UI prototype is generated from the RA model at this point, the classes of object nodes appearing at the boundary of the system are converted into tables that depict the data structure shown in Fig. 4.

Definition of Object Diagrams:
The system information that is most easily understood by customers is concrete data. As such, concrete data should be used to validate the requirements effectively. The object diagram defines a set of instance specifications of classes, under a specific system usage scenario. The instance specifications define concrete input/output data on the boundary of the system. Fig. 6 represents the object diagram pertaining to the registration of the student having the ID "SIT0003." If multiple values are defined in one slot, the developer inserts commas between each value. Moreover, if a class instance value is defined within a slot, this implies that it indicates the name of the target instance specification.

When a UI prototype is generated from the interaction activity diagrams, the class diagram, and the object diagrams, the instance specifications are converted into concrete data for the corresponding class depicted in Fig. 4.


student1 : Student
ID = SIT0001
lastName = Shibaura
firstName = Taro
college = College of Systems Engineering and Science
department = Department of Electronic Information Systems
grade = 2
contactAddress = SIT0001@hoge.ac.jp
student2 : Student
ID = SIT0002
lastName = Shibaura
firstName = Jiro
college = College of Systems Engineering and Science
department = Department of Electronic Information Systems
grade = 1
contactAddress = SIT0002@hoge.ac.jp
studentOfList1 : StudentList
ID = SIT0001
fullName = Taro Shibaura
studentOfList2 : StudentList
ID = SIT0002
fullName = Jiro Shibaura
newStudentConfirmation : NewStudentConfirmation
ID = SIT0003
lastName = Shibaura
firstName = Saburo
college = College of Systems Engineering and Science
department = Department of Mathmatical Science
grade = 1
contactAddress = SIT0003@hoge.ac.jp
add = add
cancel = cancel
newStudentInput : NewStudentInput
ID
lastName
firstName
collegeAndDepartment = Department of Electronic Information Systems\, College of Systems Engineering and Science,Departmer
grade = 1,2,3,4
contactAddress
add = add
Fig. 6 Object Diagram for "Add a New Student" Service

Definition of Scenarios:
The developers must be able to provide the customer with a guarantee of complete, proper functionality under specific scenarios, with respect to business workflows. To achieve this, the developers can define scenarios as follows: A scenario is a path of traversal over a series of interaction flows, defined in an interaction activity diagram. The scenario also defines concrete data that corresponds to the object nodes that lie on the path. The customers can confirm the scenario formulation using Selenium IDE [17] to the UI prototype automatically generated.

Definition of a Navigation Model:
Finally, the interaction flows, defined in each interaction activity diagram, are integrated. To do this, the services should be validated from a service integration perspective. A navigation model defines the necessary sequence to invoke a service. It is defined accordingly in the form of a UML activity diagram. In the navigation model, an action corresponds to a service and a partition corresponds to an authority. When a UI prototype is generated, integrating services according to the navigation model, the actions, representing before and after states, are integrated if postconditions of the previous action equate to the preconditions of the next action.

4 An Experiment for Evaluation of the Proposed Method

4.1 Purpose of Experiment
To evaluate the effectiveness of the proposed method, we compared the model it produces with the model produced using use case modeling. In this experiment, two perspectives are taken with respect to evaluation: the time taken to complete requirements analysis until customer satisfaction has been attained, and the quality of the RA model in terms of its consistency with the implementation image.

4.2 Target Applications and the Development Process
The developers for this project are three students completing four-year undergraduate degrees at our university, having basic UML knowledge and experience in developing Web-based business applications using the traditional use case method. The developers are hereafter referred to as A, B, and C. The customers are two, second year graduate students at our university. The customers are

hereafter referred to as D and E. One of target applications is a library management system required by D, hereafter referred to as LMS. The other is a schedule sharing system required by E, hereafter referred to as SSS.

The developers first analyze the requirements individually, having been given instructions not to share the information obtained from their independent analyses. They then formulate an RA model and implementation image as part of their analysis. Those developers applying the traditional method must create the use case model, the class diagram, and the implementation image, which is formulated in an arbitrary manner. In contrast, those developers applying the proposed method must create the interaction activity diagrams, the class diagram, and implementation image, in order to adapt the different model types to the traditional method. The proposed method allows the developers to substitute the UI prototype, generated from the RA model, for implementation image, and to create the remainder of the abovementioned model arbitrarily.

When the customer and the developer validate the RA model, they discuss it using only the implementation image, as a customer will typically not be capable of interpreting the model directly. Moreover, the analysis is deemed to be complete only when the implementation image meets with the customer's approval.

We retain two forms of data from the experiment: the time required to complete the modeling, prototyping, and validation process with the customer, and the sets of RA models and implementation images, captured at each stage of the process used in identifying definitions that require correction.

4.3 Evaluation Viewpoints of the RA model

There are two aspects that are measured in assessing the consistency between the RA model and implementation image: the consistency between the two with respect to the name of the visible data, and with respect to the sequence of operations. In the proposed method, these consistencies are fundamentally guaranteed as a result of the process itself. Usually, we only measure these inconsistencies for the results of the traditional method.

4.4 Experiment Result

Table 1 represents the time required to complete the modeling and prototyping, individually. For the proposed method, although the prototyping takes negligible time, the analysis of system SSS by developer C takes a significant amount of time, because of bugs in the UI prototype generation tool. It should also be noted that, with the proposed method, the time required to complete the modeling also incorporates the time required to learn how to define the RA model.

Table 1 Time Taken to Model and Prototype

		A		B		C	
		M	P	M	P	M	P
LMS	Subtracted time [Hour]	19.4	0	20.6	0	15.4	14.1
	Total time [Hour]	19.4		20.6		29.5	
SSS	Subtracted time [Hour]	15.4	23.8	24.0	33.4	86.3	1.1
	Total time [Hour]	39.2		57.4		87.4	

M: Modeling, P: Prototyping,
□: Proposed method, ▨: Traditional method

Table 2 shows the rate of inconsistency in the results of the traditional method. The number of pages is measured, targeting all use cases. The volume of data having inconsistency refers to variation between the implementation image and the use case templates.

Table 2 Rate of Inconsistency in Traditional method

	A-SSS	B-SSS	C-LMS
(a)Amount of pages	117	190	74
(b)Volume of data in all pages	1202	1740	379
(c)Volume of data in the use case model	438	1018	189
(d)Volume of inconsistent data between pages and the model	764	722	190
(e)Rate of inconsistency (d) / (b) [%]	63.5	41.4	50.1

Table 3 shows the number of classes and attributes in each method.

Table 3 Number of Classes and Attributes

	A		B		C	
	LMS	SSS	LMS	SSS	LMS	SSS
Amount of classes	13	4	18	5	4	98
Amount of attributes	61	35	89	37	9	383

Table 4 shows the number of the pages created manually or generated automatically. The pace of manual page creation is estimated on the basis of the number of pages and the time required to create the prototype.

Table 5 shows the number of validations, and the time required to conduct each validation.

Table 4 Number of Pages and Pace of Page Creation

	A	B	C
(f)Number of pages created manually	117 (SSS)	190 (SSS)	74 (LMS)
(g)Amount of time taken by manual creation of the pages [Hour]	23.8 (SSS)	33.4 (SSS)	14.1 (LMS)
(h)Pace of one page creation (g) / (f) [Minute / Page]	9.5 (SSS)	10.5 (SSS)	10 (LMS)
(i)Number of pages generated automatically	60 (LMS)	119 (LMS)	558 (SSS)

Table 5 Number of Validations and Time Required

		A	B	C
LMS	Validation times [times]	2	3	4
	Total time [minute]	67	55	105
SSS	Validation times [times]	4	6	7
	Total time [minute]	315	539	514

5 Considerations

5.1 Effectiveness of the Proposed Method

Referring to Table 1, the total time taken to perform the requirements analysis using the proposed method is less than with the traditional method, on the whole, in spite of the developers' having to learn how to define the RA model. Referring to individual data, the total time taken by C-SSS is greater than that taken by A-SSS and B-SSS. The primary reason for this was that developer C defined the RA model in detail. The number of classes and attributes, presented in Table 3, and the number of pages generated automatically, presented in Table 4, prove this fact. C-SSS produced approximately five times more than A-SSS, and about three times more than B-SSS. The time taken to conduct manual prototyping, on the scale of C-SSS, can be estimated at approximately 90 hours, simply by multiplying the pace of page creation by the number of pages generated automatically for C-SSS. Therefore, it appears that the proposed method decreases the total time required for requirements analysis, when compared with the traditional method, when the scale of the system is taken into consideration.

The UI prototype, containing concrete data or representing integrated services, was generated using the proposed method.

Although the UI prototype created manually, via the traditional method, is represented as an electronic document, in Excel or Word, it has no specification that clarifies the flow of UI transitions. In the proposed method, the flow of UI transition is specified by way of the interaction activity diagrams and the navigation model. In addition, the traditional method resulted in significant inconsistencies between the UI prototype and the RA model, as shown in Table 2.

Furthermore, although the UI prototype created via the traditional method contains concrete data, the RA model defines no specification of concrete data. In general, with respect to the defining of concrete data, the proposed method, which specifies concrete data quite clearly, is of greater use to the developer.

Referring to Table 5, there is no measurable difference between the two approaches, in terms of achieving customer approval during the validation process. As such, we suggest that the proposed method is at least equivalent to the traditional method in this regard.

5.2 Problems with the Proposed Method

There are two major problems with the proposed method.

Firstly, there are too many classes, as shown in Table 3. This problem is a result of a scattered set of data structures, which require integration.

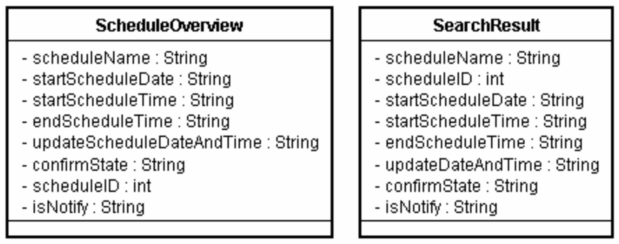

ScheduleOverview
- scheduleName : String
- startScheduleDate : String
- startScheduleTime : String
- endScheduleTime : String
- updateScheduleDateAndTime : String
- confirmState : String
- scheduleID : int
- isNotify : String
SearchResult
- scheduleName : String
- scheduleID : int
- startScheduleDate : String
- startScheduleTime : String
- endScheduleTime : String
- updateDateAndTime : String
- confirmState : String
- isNotify : String

Fig. 7 Classes Including the Same Data Structure

Fig. 7 depicts examples of classes that should be integrated. According to the data captured pertaining to the RA model, the "ScheduleOverview" class was defined first, followed by the "SearchResult" class, which has a different structure. However, a subsequent modification to the classes, post validation, resulted in their having a common structure. These redundant classes should therefore be integrated at some point in the future.

Secondly, it is difficult for the developer to precisely specify the branch conditions that are required in the alternative and exceptional flows, in both the proposed and traditional method. For example, developer A defined "There is no incorrectness for all input" as a branch condition. Thus, it is not possible to precisely interpret several of the branch conditions. Conversely, however, if all conditions of the RA model are completely and precisely defined, the model becomes too complicated to read.

In order to resolve these issues, a method is required that allows the developer to clearly and correctly define data properties that are necessary for the deliver of a service. In order to address the issue of redundancy, as a policy, these properties would become a baseline from which to derive the set of data requiring integration.

Therefore, we suggest resolving these problems by introducing the use of the Object Constraint Language (OCL), which is a formal language for object-oriented development. This will allow a developer to define data properties clearly and precisely.

6 Related Work

Somé [5, 6] has proposed a formal grammar of use case templates, in order to support requirements engineering with respect to requirements elicitation, clarification, composition, and simulation. Employing this grammar results in an improved consistency between a behavioral model, such as a use case model, and a structural model, such as a class diagram. In addition, this grammar can simulate use cases, employing the use case model, and the classes employing a domain model. A key characteristic of a use case simulation is the implementation- independent simulation, using only those steps that are defined in the model. The developer simulates the use cases by selecting actor's behaviors defined in the use case templates and by selecting the system's conditions, ultimately deciding the path of branched flows. In this simulation, no concrete data and no means of input are represented. Jayaraman [9] proposes a simulation of use cases using a use case chart for the validation of use cases.

We have proposed a method to define concrete data, which is then reflected by the generated UI prototype so that customers are able to easily understand the RA model. Moreover, we have proposed how to generate a UI prototype that represents the concrete means of input. For example, with regard to a service delivering search refinement, information regarding the dynamic change of data between page transitions is critical to developing an understanding of the behavior. Also, concrete means of input are quite important, as described in section 2.2.

Lin et al. [7] have proposed a formal grammar for use case templates and a method of converting use case templates into activity diagrams. This method represents a trend in the field toward defining interaction flows as activity diagrams.

Störrle [13] has proposed a method to convert activity diagrams into color petrinets in order to perform validation of the data flows in the activity diagrams. In addition, Eshuis [14] has proposed a method to validate the workflows defined in the activity diagrams. This method is particularly suited to dealing with event driven behavior, data, loop, and real time.

These methods [13, 14] focus on the validation of the reachability of flows defined in the activity diagrams, under specific scenarios. In contrast, our proposed method focuses on validation of the operational flows defined in the RA model with the intent of determining what data the users are capable of entering and confirming on the boundaries of the system (extreme cases). As such, our proposed method does not directly challenge the effectiveness of the methods described in [13, 14]. There remains the possibility of integrating these methods through the appropriate combination of the concrete syntax of natural language, with respect to conditions and states of data. This is a direct benefit of having selected UML activity diagram as our approach to notation.

7 Conclusion

The major contributions of our proposed method, validated through experiments, may be summarized in three points. Firstly, concrete data is needed to easily formulate an understanding of the customer requirements and, as such, is specified clearly using the proposed method. Secondly, our method can decrease the time required to conduct requirements analysis, regardless of the additional learning time that is required. Thirdly, unlike usage of the traditional method, where it is difficult to guarantee consistency of the data and flow between the RA model and the prototype, usage of the proposed method does guarantee such consistency.

In terms of future work, we believe that more consideration can be given to defining the conditions representing business rules, using a formal language such as OCL. Moreover, we suggest that additional consideration should be paid to defining a methodology for prevention and elimination of duplicate classes.

References:

[1] UML: http://www.uml.org/

[2] Jacobson, I., Christerson, M., Jonsson, P., and Övergaard. G., Object-oriented software engineering: A usecase driven approach, Addison-Wesley Publishing, 1992.

[3] IEEE Computer Society, IEEE Recommended Practice for Software Requirements Specifications, IEEE Std 830–1998, 1998.

[4] Cockburn, A., Writing Effective Use Cases, Addison-Wesley Publishing, 2000.

[5] Somé, S.S., Use Case based Requirements

Validation with Scenarios, Proc. of the 13th IEEE International Conference on Requirements Engineering, 2005, pp.465–466.

[6] Somé, S.S., Supporting use case based requirements engineering, Information and Software Technology, Vol.48, No.1, 2006, pp.43–58.

[7] Lin, X., Wang, C., Chu, W.C., and Shih, C., A Use Case Model and its Transformation to Activity Diagram, Proc. of the 18th International Conference on Software Engineering and Knowledge Engineering, 2006, pp.556–561.

[8] Elkoutbi, M., Khriss, I. and Keller, R.K., Automated Prototyping of User Interfaces Based on UML Scenarios, Journal of Automated Software Engineering, Vol.13, No.1, 2006, pp.5–40.

[9] Jayaraman, P.K., Whittle, J., UCSIM: A Tool for Simulating Use Case Scenarios, Companion to the Proc. of the 29th International Conference on Software Engineering, 2007, pp.43–44.

[10] Ogata, S. and Matsuura, S., A UML-based Requirements Analysis with Automatic Prototype System Generation, Communication of SIWN, Vol.3, 2008, pp.166–172.

[11] ACM SIGSOFT, Special Issue on Rapid Prototyping, ACM SIGSOFT Software Engineering Notes, Vol.7, No.5, 1982.

[12] Astah*: http://www.change-vision.com/

[13] Störrle, H., Semantics and Verification of Data Flow in UML 2.0 Activities, Electronic Notes in Theoretical Computer Science, Vol.127, No.4, 2005, pp.35–52.

[14] Eshuis, R. and Wieringa, R., Tool support for verifying UML activity diagrams, IEEE Transaction on Software Engineering, Vol.30, No.7, 2004, pp.437–447.

[15] Sommerville, P Sawyer, Requirements Engineering: A Good Practice Guide, John Wiley & Sons, 1997.

[16] Ogata, S. and Matsuura, S., Scenario-based automatic prototype generation, The 32nd Annual IEEE International COMPSAC, 2008, pp.492–493.

[17] SeleniumIDE: http://seleniumhq.org/projects/ide/