A Method of Automatic Integration Test Case Generation from UML-based Scenario

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: - One key to success for high quality enterprise information systems development is to validate the customers' requirements sufficiently at the early stage. Scenarios are an effective means to an end because they make it possible to represent various situations of system usage. Most scenarios are defined by using a natural language or such a formal language as Unified Modeling Language (UML) and describe normal, alternative, and exceptional service flows from the point of view of system usage. As a result, scenarios make it easy for the customers to confirm their requirements intuitively because of the concreteness. On the other hand, based on the V-model, which is well-known software development process and denotes the correspondence of requirement analysis stage to integration test stage, it is desirable that the testers should test the system by using the test cases derived from the validated scenarios. We have proposed a UML-based requirements analysis (RA) model with automatic prototype system generation for enterprise Web application development. This paper proposes a way to efficiently create reliable test cases from the scenarios that have been validated by the customers using the prototype system which was generated by the RA model.

Key-Words: - Unified Modeling Language, Web Application, Scenario, Test Case for Integration Testing

1 Introduction

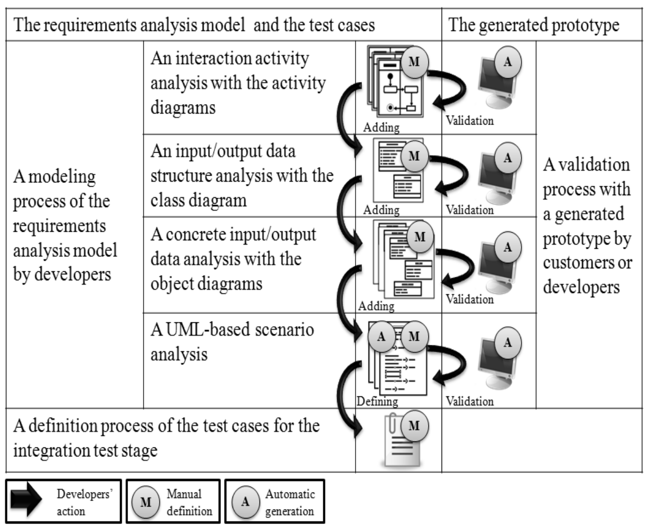
One key to success for high quality enterprise information systems development is to validate the customers' requirements sufficiently at the early stage. Scenarios are an effective means to an end because they make it possible to represent various situations of system usage.

Recently, Unified Modeling Language (UML) [1] has been widely recognized as an effective technique to rigorously define the software specification because many researchers have introduced UML models into their proposal as the core elements [2-4].

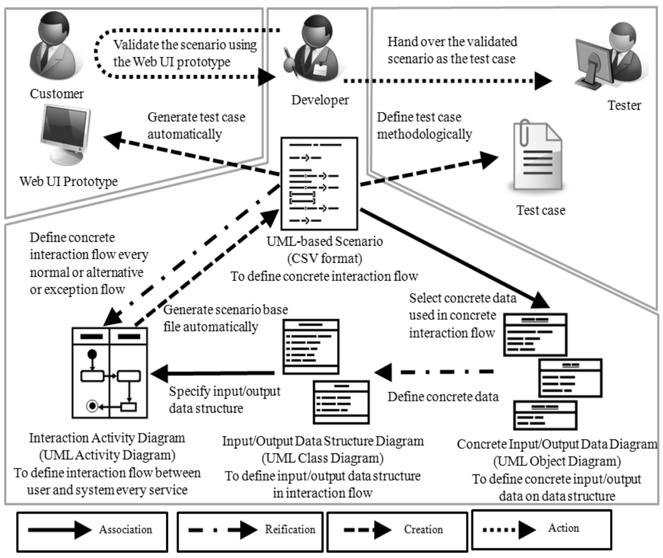
Most scenarios [5-9] are defined by using a natural language or a formal language such as UML and describe normal, alternative, and exceptional service flows from the point of view of system usage. As a result, scenarios make it easy for the customers to confirm their requirements intuitively because of the concreteness. On the other hand, based on the V-model, which is well-known software development process and denotes the correspondence of requirement analysis stage to integration test stage, it is desirable that the testers should test the system by using the test cases derived from the validated scenarios.

Several researchers [5-7] have made use of their scenarios to generate user Interface prototype automatically so that they can define and validate requirements specification efficiently. On the other hand, to decrease the time cost of test phase or to support exhaustive testing, several researchers have handled scenarios as the source to generate test cases [10-12] or test codes [12, 13] or test path [9].

We have proposed a UML-based requirements analysis (RA) method with automatic prototype system generation for enterprise Web application development [14-16]. In this paper, we propose a way to efficiently create reliable test cases from the scenarios that have been validated by the customers using the prototype system which was generated by the RA model The RA model consists of activity diagrams, class diagram, object diagrams and UML-based scenarios. The activity diagrams represent all flows which include normal, alternative and exceptional flows for every user authority and every service exhaustively. The class diagram represents data structures used in the flows of the activity diagrams. The object diagrams represent concrete


The requirements analysis model and the test cases
The generated prototype
A modeling process of the requirements analysis model by developers
An interaction activity analysis with the activity diagrams
An input/output data structure analysis with the class diagram
A concrete input/output data analysis with the object diagrams
A UML-based scenario analysis
A validation process with a generated prototype by customers or developers
A definition process of the test cases for the integration test stage
Adding
Defining
Validation
M
A
Developers' action
Manual definition
Automatic generation

The relation among each diagram, scenario and test case.


Customer
Validate the scenario using the Web UI prototype
Developer
Hand over the validated scenario as the test case
Tester
Generate test case automatically
Define test case methodologically
Web UI Prototype
Test case
UML-based Scenario (CSV format) To define concrete interaction flow
Define concrete interaction flow every normal or alternative or exception flow
Generate scenario base file automatically
Select concrete data used in concrete interaction flow
Specify input/output data structure
Define concrete data
Interaction Activity Diagram (UML Activity Diagram) To define interaction flow between user and system every service
Input/Output Data Structure Diagram (UML Class Diagram) To define input/output data structure in interaction flow
Concrete Input/Output Data Diagram (UML Object Diagram) To define concrete input/output data on data structure
Association
Reification
Creation
Action

The entire flow of the requirements analysis model and test case definition.

Fig.1 The Overview of Our Approach from the Requirements Analysis to Test Case Definition

data corresponding to the data structure of the class diagram. The UML-based scenarios represent a concrete flow by selecting and supplementing the appropriate part of exhaustive flows, data structures and concrete data extracted from the above-mentioned three kinds of models. Scenarios are represented as a UI prototype and a set of concrete data. There are three strong points in our approach. First, a non-functional Web UI prototype which is described in Hyper Text Markup Language (HTML) can be generated automatically from the RA model stepwise at each stage adding a kind of diagram or scenarios [14, 15] so that the developer can validate their model iteratively with the generated prototype from very early stage defining only one kind of diagrams. The advantage of the UI prototype [17, 18] is mainly to enable the validation of the service flow in the both view of UI transitions and input/output data represented on UI by customers. Secondly, concrete data specified in the object diagram can be reflected into the generated prototype [14, 15] so that the customers can validate the services with the prototype intuitively and easily. Thirdly, the test cases can be defined semi-automatically according to the RA model in which the customers have approved the contents of the services through the generated prototype. As a result, the tester can carry out the integration testing systematically by using the test cases derived from the validated scenarios.

2 Overview: from Requirements Analysis Process to Test Case Definition Process

As shown by the entire flow of the requirements analysis model and test case definition in Fig.1, there are five steps to define the RA model or the test case. At the former four steps, the developers define and validate and refine the RA model iteratively. To make it possible to validate the RA model effectively by introducing the prototyping into our approach, we have developed an automatic prototype generation tool which can generate a Web UI prototype automatically at each stage adding a kind of diagram or scenarios. Therefore, the developers can validate the RA model iteratively by such stepwise automatic prototype generation. In the latter step, as shown by the relation among each diagram, scenario and test case, the developers can define the test cases systematically from the UML-based scenario which is defined by using the validated RA model fully in the viewpoint of both behavior and data structure. It notes that the developers need to obtain a set of services which are equal to the concept of use case from requirements of the customers before the first step. The following sections illustrate these steps in detail with the RA model of an order management system.

3 The Modeling Process of the Requirements Analysis Model

The RA model consists of activity diagrams, a class diagram and object diagrams in UML 2.0, and UML-based scenarios. Three kinds of the UML diagrams notation in the RA model correspond completely with the original UML notation. Also, they are defined by Astah* [19] which is an analysis or design support tool and which supports UML modeling. The following sections describe the three kinds of diagrams and the UML-based scenario along with the RA model example of the order management system development for a confectionery.

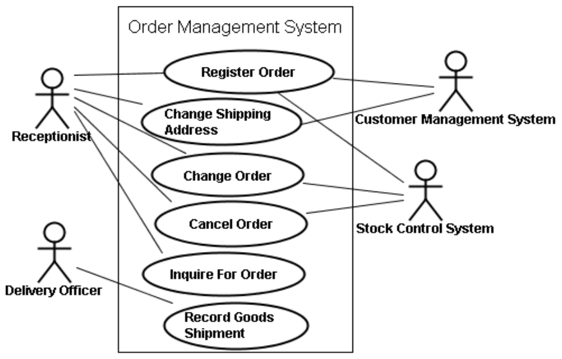


Fig.2 The Use Case Diagram of an Order Management System for a Confectionery

Fig.2 represents a use case diagram of the system. This system manages order information for confectionery. Also it handles customer information and stock information by interacting the Customer Management System and the Stock Control System. On the other hand, there are two kinds of user of the system. One is the Receptionist who receives a telephone call or an order mail to receive an order, the other is the Delivery Officer who records shipment of orders. As the overview of workflow, at the first step, the receptionist receives an order from a customer and registers the order into the system. Then, after the receptionist finished inputting the order, the system sends shipment sheets to the delivery officer of each distribution center. Finally, the delivery officer instructs the deliverer to ship the goods to the customer. Also, the delivery officer records goods shipment. If there is a request for changing or canceling orders or changing the shipment address of the customer, the receptionist conducts the work using the system. In this paper, the following sections illustrates about the service of “Change Shipping Address” in detail.

3.1 The Interaction Activity Analysis

The activity diagrams, which we call Interaction Activity Diagrams (IADs), define interaction between a user and a system for every user authority and every service. The IAD defines all normal, alternative and exceptional flows. The developer should define IAD according to the following rules, so that they can define the interaction clearly and enough.

(1) To define who performs each action, define partitions in an IAD which are named after a user authority or a system. And all nodes of the IAD should be defined within the partitions.
(2) To define the interaction as a sequence of the actions which is performed by a user and a system by turns, do not define concurrent flows over several partitions.
(3) To clarify responsibility of actions in each partition, define each action of a user as an input action to a system. Also, define each action of the system as an output action to the user or an internal logic action.
(4) To avoid defining a complex action, represent an action with a simple sentence. Therefore, we have specified the simple format “<verb> <noun>“ for an action. For example, an action which follows the format is “input password”. The <verb> of a user action defines input behavior toward a system. And the <noun> defines a target of the behavior such as an input item. On the other hand, in a system action, the <verb> defines output behavior toward the user or logic behavior such as CRUD (Create, Read, Update and Delete). And the <noun> defines a target of the behavior such as an output item or a message.
(5) To clarify input/output data provided to the user, define the object node in the middle of the control flow which crosses the boundary between a user partition and a system partition.
(6) To clarify the conditions on control flow branches, define guard conditions on the control flows just behind a decision node.
(7) To clarify pre and post conditions precisely when a service is executed, define a note, which its content starts with “pre-condition:”, as the pre-condition for an initial node. And define a note, which content starts with “post-condition:(normal)” or “post-condition:(exceptional)”, as the post-condition for an activity final node.
(8) To distinguish normal flows from exceptional flows, define a partition named “Interaction”. The control flow, which reaches the node in the “Interaction” partition just after a conditional

branch in the system or "Interaction" partition, are handled as the exceptional flows.

About the noun of the rule of (4), we have given a simple format to the noun to indentify clearly and uniquely an input/output item which belongs to a data structure represented by an object node (see Fig.3). Actually, it is possible to assume a class as input/output data structure because the object node on Astah* can assign a class as the base class. In Fig.3, an "object_node_name" is the name of an object node which corresponds to the root name of data structure defined by the base class. A "parent_item_name" and an "item_name" are the name of a structured item represented by an attribute of the base class. The "parent_item_name" implicates the name of the structured item which is in the middle-layer if the data structure has multi-layer.

Fig.4 represents the interaction flow of the service of "Change Shipping Address". As the explanation of the flow, at the first step, a receptionist executes the service by selecting "Change shipping address". Next, to identify the customer data that shipping address would be changed, the receptionist input the ID of the customer to the system and makes the system start search by selecting "search". Then, if the expected data was found, the receptionist inputs new shipping address for the customer data and selects "update". Next, the system asks the receptionist whether the edited data is correct. If the receptionist confirmed the data, he selects "update". Finally, after the system updated the internal data correctly, it finishes the service.

```
<noun> := object_node_name "::::" item | item
item := parent_item_name "::" item | item_name
object_node_name := string
parent_item_name := string
item_name := string
string := ("[^();,]")+
```

Fig.3 The Formal Grammar of the <noun>

3.2 The Input/Output Data Structure Analysis

The class diagram, which we call Input/Output Data Structure Diagram (IODSD), defines data structure based on input/output data represented as object nodes in the IAD. Therefore, the IODSD is expected to define essential data which must be handled on UI. Since the data handling on UI include generally a lot of entity data which are referred by the boundary, it can regards a defined input/output data structure as a class candidate and a defined input/output item as a class or attribute candidate. Since some of them are not entity data which means the persistent data, but data handling on UI only, the developers can elicit these candidates for not only entity classes but also boundary classes. Just after the stage assigning classes to the object nodes, the IODSD may include the same candidates among the resultant class and attribute candidates. The developers elaborate an appropriate class diagram to integrate the candidates properly. Fig.5 represents an example of the IODSD.

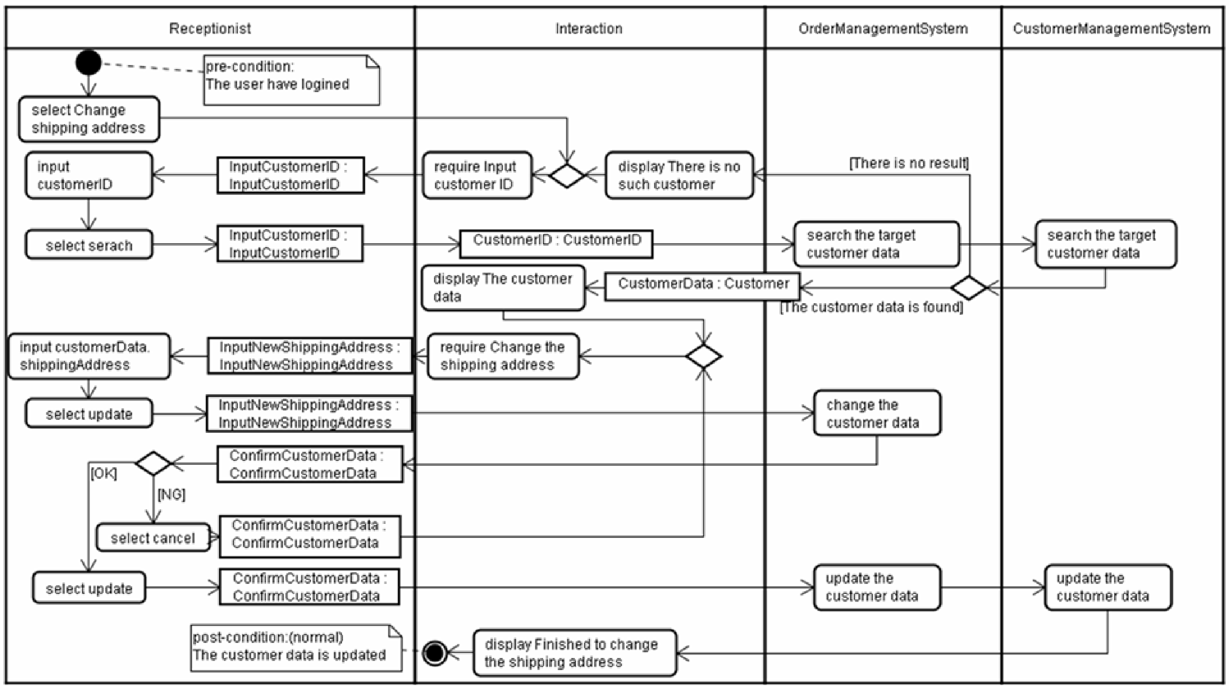
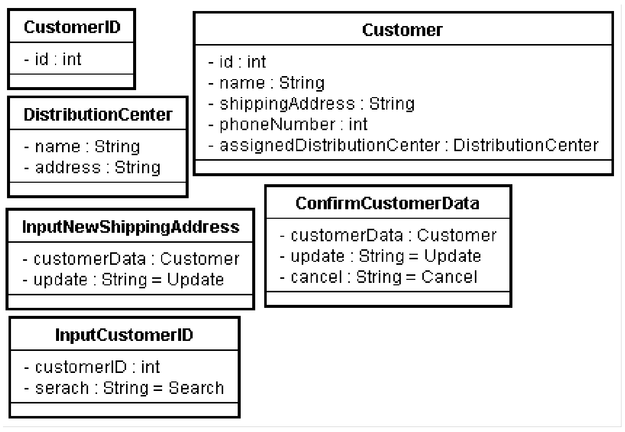
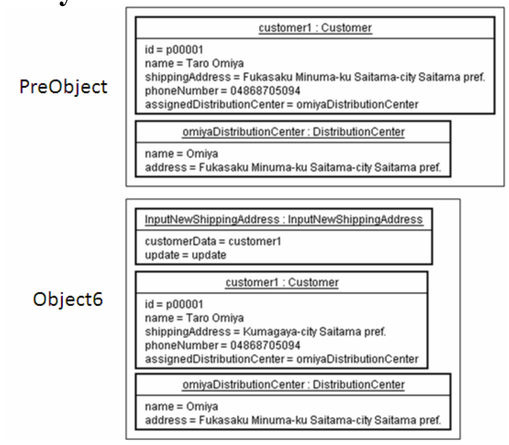

Receptionist
Interaction
OrderManagementSystem
CustomerManagementSystem
pre-condition: The user have logined
select Change shipping address
input customerID
InputCustomerID : InputCustomerID
require Input customer ID
display There is no such customer
[There is no result]
select serach
InputCustomerID : InputCustomerID
CustomerID : CustomerID
search the target customer data
search the target customer data
display The customer data
CustomerData : Customer
[The customer data is found]
input customerData. shippingAddress
InputNewShippingAddress : InputNewShippingAddress
require Change the shipping address
select update
InputNewShippingAddress : InputNewShippingAddress
change the customer data
ConfirmCustomerData : ConfirmCustomerData
[OK]
[NG]
select cancel
ConfirmCustomerData : ConfirmCustomerData
select update
ConfirmCustomerData : ConfirmCustomerData
update the customer data
update the customer data
post-condition:(normal) The customer data is updated
display Finished to change the shipping address

Fig.4 The Interaction Activity Diagram of a Service of "Change Shipping


CustomerID
- id : int
DistributionCenter
- name : String
- address : String
Customer
- id : int
- name : String
- shippingAddress : String
- phoneNumber : int
- assignedDistributionCenter : DistributionCenter
InputNewShippingAddress
- customerData : Customer
- update : String = Update
ConfirmCustomerData
- customerData : Customer
- update : String = Update
- cancel : String = Cancel
InputCustomerID
- customerID : int
- serach : String = Search
Fig.5 The Input/Output Data Structure Diagram Related to the Service of "Change Shipping Address"

3.3 The Concrete Input/Output Data Analysis


PreObject
customer1 : Customer
id = p00001
name = Taro Omiya
shippingAddress = Fukasaku Minuma-ku Saitama-city Saitama pref.
phoneNumber = 04868705094
assignedDistributionCenter = omiyaDistributionCenter
omiyaDistributionCenter : DistributionCenter
name = Omiya
address = Fukasaku Minuma-ku Saitama-city Saitama pref.
Object6
InputNewShippingAddress : InputNewShippingAddress
customerData = customer1
update = update
customer1 : Customer
id = p00001
name = Taro Omiya
shippingAddress = Kumagaya-city Saitama pref.
phoneNumber = 04868705094
assignedDistributionCenter = omiyaDistributionCenter
omiyaDistributionCenter : DistributionCenter
name = Omiya
address = Fukasaku Minuma-ku Saitama-city Saitama pref.
Fig.6 The Concrete Input/Output Data Diagrams Related to the Service of "Change Shipping Address"

The object diagrams, we called Concrete Input/Output Data Diagrams (CIODDs), define concrete data of the input/output data structure defined by an IODSD. By defining the concrete data needed when the usage situation of a service such as a normal or alternative or exceptional flow is presumed, the developers can validate the RA model in the viewpoint of both data structure and behavior. For example, a data structure which can't describe required data or a flow which can't represent required data is improper model. However, these problems are difficult to be detected from abstract data structure such as a class diagram and make the possibility of creating misunderstandings about the data constraints between the developers without a specification about concrete data. Therefore, we have adopted the CIODD into the RA model specifically. In our approach, to define multiple concrete data to one slot, comma as the separator between the data is inserted. Also, to define a concrete data into a slot which has a class type, the object name of instance specifications which has the class type is defined into the slot. Fig.6 represents examples of the CIODD.

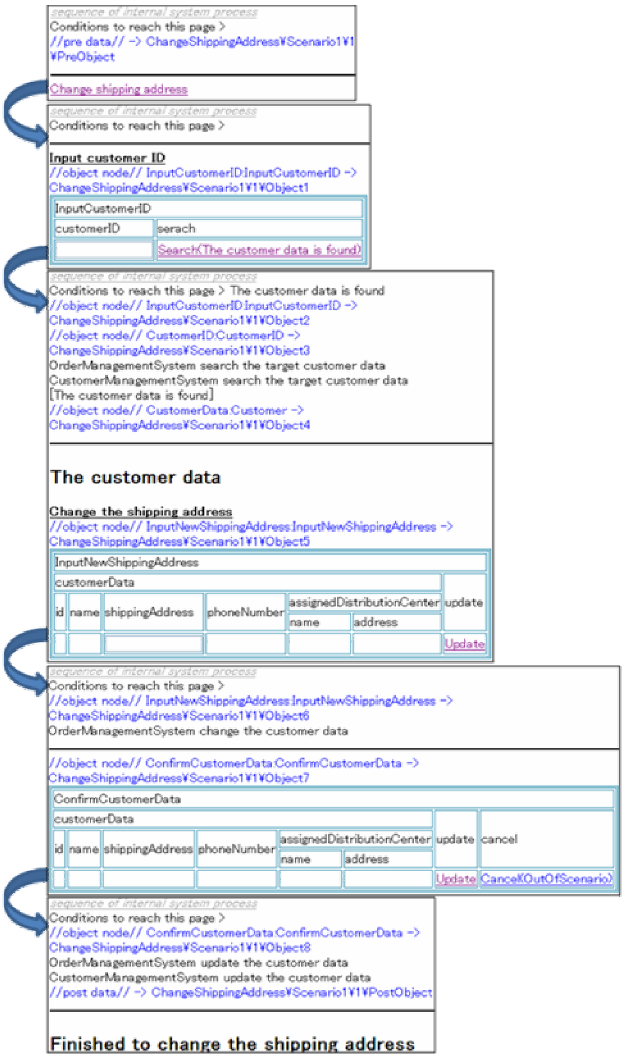
3.4 The UML-based Scenario Analysis

In the above-mentioned three steps, the developers have defined three kinds diagram of the RA model; the IAD represents the exhaustive interaction flow; the IODSD represents the input/output data structure needed in the interaction flow; the CIODD represents the concrete input/output data, which can be used to validate the RA model in the viewpoint of the validation of data structure. Then, the UML-based scenarios define concrete paths to validate the exhaustive interaction flow in the viewpoint of behavior, using concrete input/output data. Concretely, the UML-based scenarios are defined for every normal, alternative and exceptional flow. Also, they define the prepared and predicted data which are the internal system data before and after the service execution to clarify the usage situation of the system. These scenarios are represented as a UI prototype and a set of concrete data. It notes that the Transition Condition is a set of guard expressions which specifies a flow path through the "Interaction" and system partition.

(a) A service name is the name of the service which corresponds to the name of an IAD.
(b) A scenario name is the name of a scenario for the above-mentioned service. The format of scenario name is "scenario<number>", and is numbered automatically by the prototype generation tool.
(c) Prepared Data named "PreObject" is the internal system data needed to execute the scenario, which defines the data related only to the scenario.
(d) Predicted Data named "PostObject" is the internal system data after the scenario execution, which defines the data related only to the scenario.
(e) Scenario steps consist of the sequence of UIT trigger actions and transition conditions and object nodes. The UIT trigger actions and transition conditions specify a concrete normal or alternative or exceptional flow path of an IAD. On the other hand, the object nodes specify a concrete data using in the scenario such as an input/output data or internal system data. The developers do not need to concern about how to define these steps excluding how

to specify concrete data to each object node because these are represented as the view of UI prototype.

Our automatic prototype generation tool can generate automatically a UI prototype which includes above-mentioned elements. the above (a) and (b) are generated automatically; the path of above (c) to define a CIODD into Astah* model is represented by the generated UI prototype as "//pre data//"; the path of above (c) to define a CIODD into Astah* model is represented by the generated UI prototype as "//post data//"; the above (e) defines all candidates, which are the elements to specify a flow path uniquely or the object nodes, for each service. To completely define the scenario, the developer defines the concrete data according to the usage situation of the scenario. Concretely, the developer defines the CIODDs as Prepared Data, Predicted Data and Object<number> which are represented by the generated UI prototype.


sequence of internal system process
Conditions to reach this page >
//pre data// -> ChangeShippingAddress¥Scenario1¥1¥PreObject

Change shipping address

sequence of internal system process
Conditions to reach this page >

Input customer ID
//object node// InputCustomerID:InputCustomerID -> ChangeShippingAddress¥Scenario1¥1¥Object1

InputCustomerID	
customerID	serach
	Search(The customer data is found)

sequence of internal system process
Conditions to reach this page > The customer data is found
//object node// InputCustomerID:InputCustomerID -> ChangeShippingAddress¥Scenario1¥1¥Object2
//object node// CustomerID:CustomerID -> ChangeShippingAddress¥Scenario1¥1¥Object3
OrderManagementSystem search the target customer data
CustomerManagementSystem search the target customer data
[The customer data is found]
//object node// CustomerData:Customer -> ChangeShippingAddress¥Scenario1¥1¥Object4

The customer data

Change the shipping address
//object node// InputNewShippingAddress:InputNewShippingAddress -> ChangeShippingAddress¥Scenario1¥1¥Object5

InputNewShippingAddress						
customerData						update
id	name	shippingAddress	phoneNumber	assignedDistributionCenter		
				name	address	
						Update

sequence of internal system process
Conditions to reach this page >
//object node// InputNewShippingAddress:InputNewShippingAddress -> ChangeShippingAddress¥Scenario1¥1¥Object6
OrderManagementSystem change the customer data

//object node// ConfirmCustomerData:ConfirmCustomerData -> ChangeShippingAddress¥Scenario1¥1¥Object7

ConfirmCustomerData							
customerData						update	cancel
id	name	shippingAddress	phoneNumber	assignedDistributionCenter			
				name	address		
						Update	Cancel(OutOfScenario)

sequence of internal system process
Conditions to reach this page >
//object node// ConfirmCustomerData:ConfirmCustomerData -> ChangeShippingAddress¥Scenario1¥1¥Object8
OrderManagementSystem update the customer data
CustomerManagementSystem update the customer data
//post data// -> ChangeShippingAddress¥Scenario1¥1¥PostObject

Finished to change the shipping address

Fig.7 A Normal Scenario of the Service of "Change Shipping Address"

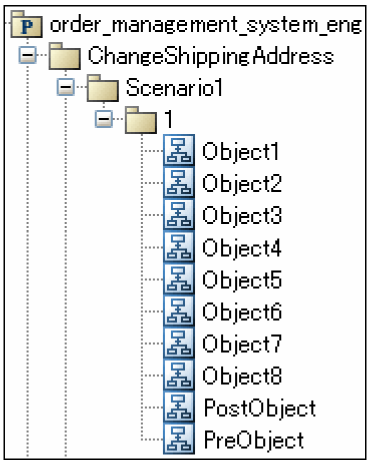


Fig.8. The Structure Tree of Object Diagrams

Fig.7 represents the normal scenario of the service of "Change Shipping Address" which was edited according to the above-mentioned steps.

This scenario used only the data defined in Fig.4, Fig.5, Fig.6 and Fig.8 except for the scenario name. In particular, the CIODDs on the scenario must be defined in the Astah* Structure Tree View with the structure which consists of the service name, the scenario name and the CIODD as shown in Fig.8. Concretely, the service name package includes the scenario name packages, and the scenario name package includes the CIODDs.

The scenario in Fig.7 specifies the flow that the receptionist modifies the shipping address of a customer from "Fukasaku Minuma-ku Saitama-city Saitama pref." into "Kumagaya-city Saitama pref.". In the object diagrams named "Object<number>", the sets of concrete data which depict the data flow of the above-mentioned scenario are defined.

About the resultant instance specifications of the CIODDs in a scenario, these can be categorized automatically as the following Table 1.

Table 1 The Categorization of the Instance Specification on a Scenario

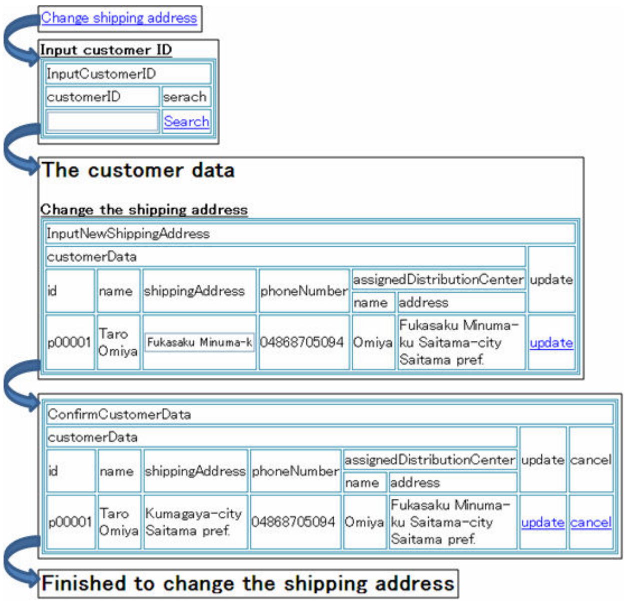
Categories of instance specification	Kinds of CIODD on the scenario		
	Prepared data	Predicted data	Object nodes of scenario steps
Input/output data on UI only	DA	DA	A
Created system data	DA	A	NR
Remaining system data	A	A	NR
Updated system data	A	A*	NR
Deleted system data	A	DA	NR

A = Appeared, DA = DisAppeared, NR = NoRelated,
A* = Appeared (For an instance specification, the object name and type remained, but the slots are updated)

4 The Scenario-based Prototype Generation

In our approach, the scenario-based prototype can be generated automatically from the RA model defined until former steps. Originally, in our approach, the prototype can be generated at each stage adding a kind of diagram [14, 15]. In this paper, the scenario-based prototype only is explained. Fig.9 shows the sample of a Web UI prototype which is generated from the RA model.

In spite of the non-functional prototype, the prototype which is generated from the RA model that includes the scenario can represent the change of output data as if the prototype had a function. Then, using such scenario-based prototype, the developers can validate the RA model which meets the customers' requirements by discussing with the customers.


Change shipping address

Input customer ID

InputCustomerID	
customerID	serach
	Search

The customer data

Change the shipping address

InputNewShippingAddress

customerData						update
id	name	shippingAddress	phoneNumber	assignedDistributionCenter		
				name	address	
p00001	Taro Omiya	Fukasaku Minuma-k	04868705094	Omiya	Fukasaku Minuma-ku Saitama-city Saitama pref.	update

ConfirmCustomerData

customerData						update	cancel
id	name	shippingAddress	phoneNumber	assignedDistributionCenter			
				name	address		
p00001	Taro Omiya	Kumagaya-city Saitama pref.	04868705094	Omiya	Fukasaku Minuma-ku Saitama-city Saitama pref.	update	cancel

Finished to change the shipping address

Fig.9 The Scenario-based Prototype of the Normal Scenario of "Change Shipping Address"

5 The Definition process of the Test Case for the Integration Test Stage

Until former steps, the UML-based scenarios have been defined for every normal or alternative or exceptional flow and have specified concrete input/output data for each object node of the scenario steps and for the prepared and predicted data. Thus, these scenarios are similar to test cases exceedingly in the viewpoint of not only specifying the concrete flow path but also containing the concrete data. Furthermore, these scenarios are expected to be validated by the customers and developers using the generated prototype in the validation process. Since such validated scenario must be achieved by the system correctly and completely, the system must be tested according to the validated scenarios. Therefore, it is important for the developer and tester to create the test cases which correspond to the validated scenarios without any inference in the definition process of them. In this paper, we have attempted to create systematically the test cases for the integration test stage using the RA model.

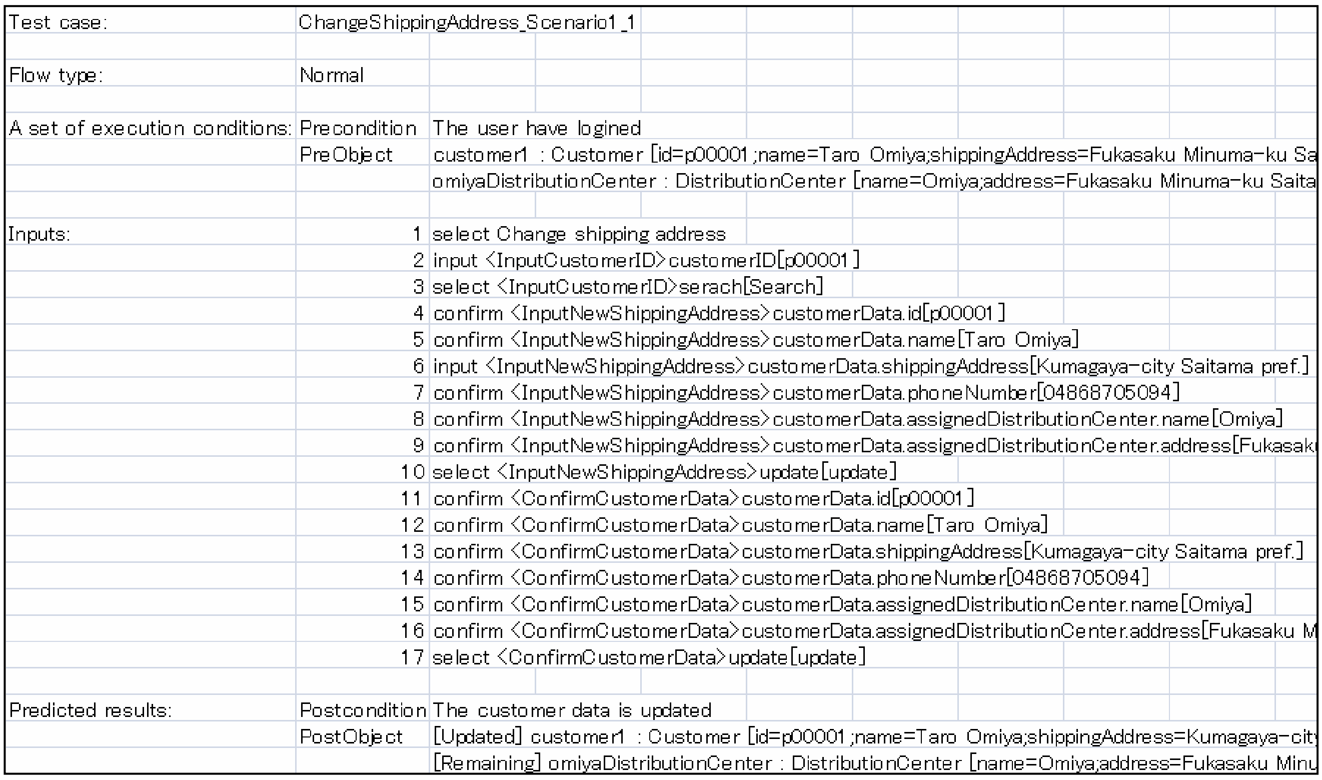
According to [20], the test case is documentation specifying inputs, predicted results, and a set of execution conditions for a test item. Therefore, we make the UML-based scenario correspond to the test case as follows.

(1) The Inputs correspond to the action sequence of a user in an IAD along a specified normal or exceptional flow path. Furthermore, the concrete data input by the user are assumed as the instance specifications which are categorized into the "input/output data on UI only" in Table 1.

(2) The Predicted Results correspond to two kinds of the resultant object in the RA. One is the generated prototype so that the tester can confirm whether predicted data is represented on the UI by executing the scenario. The other is a set of the difference data which is expected to be changed (such as created or updated or deleted) by the system operation and user manipulation and which is discovered as the result of comparing the predicted data with the prepared data so that the tester can confirm what is changed among the internal system data by executing the scenario. Also, the predicted results must satisfy the post-condition in the service if the flow type of test case is normal flow.

(3) The set of Execution Conditions correspond to the prepared data of the UML-based scenario which enumerates the required data to execute the scenario. Also, the set of execution conditions must satisfy the pre-condition in the service.

There is an important thing to define the test cases. That is, the defined test cases are whether they cover testing for exhaustive flows. In our approach, in related to above (1), the test cases covering exhaustive flow can be defined systematically since the IAD defines exhaustive flows every service. Fig.10 represents an example of the test case corresponding to the UML-based scenario in Fig.7.



Test case:	ChangeShippingAddress_Scenario1_1	
Flow type:	Normal	
A set of execution conditions:	Precondition	The user have logined
	PreObject	customer1 : Customer [id=p00001;name=Taro Omiya;shippingAddress=Fukasaku Minuma-ku Sa
		omiyaDistributionCenter : DistributionCenter [name=Omiya;address=Fukasaku Minuma-ku Saita
Inputs:	1	select Change shipping address
	2	input <InputCustomerID>customerID[p00001]
	3	select <InputCustomerID>serach[Search]
	4	confirm <InputNewShippingAddress>customerData.id[p00001]
	5	confirm <InputNewShippingAddress>customerData.name[Taro Omiya]
	6	input <InputNewShippingAddress>customerData.shippingAddress[Kumagaya-city Saitama pref.]
	7	confirm <InputNewShippingAddress>customerData.phoneNumber[04868705094]
	8	confirm <InputNewShippingAddress>customerData.assignedDistributionCenter.name[Omiya]
	9	confirm <InputNewShippingAddress>customerData.assignedDistributionCenter.address[Fukasak
	10	select <InputNewShippingAddress>update[update]
	11	confirm <ConfirmCustomerData>customerData.id[p00001]
	12	confirm <ConfirmCustomerData>customerData.name[Taro Omiya]
	13	confirm <ConfirmCustomerData>customerData.shippingAddress[Kumagaya-city Saitama pref.]
	14	confirm <ConfirmCustomerData>customerData.phoneNumber[04868705094]
	15	confirm <ConfirmCustomerData>customerData.assignedDistributionCenter.name[Omiya]
	16	confirm <ConfirmCustomerData>customerData.assignedDistributionCenter.address[Fukasaku M
	17	select <ConfirmCustomerData>update[update]
Predicted results:	Postcondition	The customer data is updated
	PostObject	[Updated] customer1 : Customer [id=p00001;name=Taro Omiya;shippingAddress=Kumagaya-city
		[Remaining] omiyaDistributionCenter : DistributionCenter [name=Omiya;address=Fukasaku Minu

Fig.10 The Test Case corresponding to the Normal Scenario

We have defined a format of a test case as Fig.10. The contents of test case in Fig. 10 are generated following the scenario in Fig. 7. The content of the "Test case" includes the service name and the scenario name of the scenario. The content of the "Flow type" becomes "Normal" because the flow path of the scenario has no control flow fulfilling the (8) of the list in section 3.1. The "A set of execution conditions" consists of the "Precondition" which is defined according to the "pre-condition" of the IAD and the set of "PreObject" which is defined according to the prepared data in the scenario. The "Inputs" represents the sequence of the user's action in the IAD according to the scenario steps. About the part of "<concrete data name>", the concrete data is defined according to the instance specifications which are categorized into "input/output data on UI only" and which are in adequate position for the actions. Also, the "Predicted results" consist of the "Postcondition" which is defined according to the "post-condition" of the IAD and the set of "PostObject" which is defined according to the predicted data in the scenario. To clarify the difference between the predicted data and the prepared data in the scenario, the data condition represented as "[<condition>]" (<condition> := "created" | "updated" | "deleted" | "remaining") in Fig.10 is generated following the rule of Table 1.

However, a few points cannot be defined systematically as is. It is to define the concrete data into the contents of "Inputs". In an IAD, a user inputs to the data outputted from the system. However, the concrete data should reflect into the contents of "Inputs" is defined to the object node which is sent to the system. Therefore, the input item which is sent to the system should correspond to the output item which is reached from the system properly. For example, this problem can be resolved by defining the correspondence between the input item and the output item, using tagged value.

6 Related Work

Several researchers [9, 10-13] have proposed to generate specification or code for test stage from mainly UML behavioral model such as activity diagram and/or sequence diagram and/or state machine diagram. Some of these works [9, 12] introduce contracts, which are pre-/post-conditions for executing services, into the use cases or the messages of a sequence diagram. Then, the contracts are defined by the constraint language such as Object Constraint Language (OCL) or OCL like languages.

These works [9, 10-13] have strongly focused on measuring the test coverage based on the idea of generating test cases from the behavioral model which represents exhaustive flows or the sets of scenario which consists of normal and exceptional flows. On the other hand, in our approach, it is important that the UML-based scenarios have defined satisfactory and appropriate service for the customers. After validating the scenarios through generated UI prototype, the test cases are semi-automatically defined by the concrete data derived from the validated scenarios. Also, from the viewpoint of test coverage, an IAD (Interaction Activity Diagram) exhaustively defines all flows for each service, so that the testers can get all test case templates for the service. Furthermore, the works [9, 10-13] do not specify how to confirm the predicted results of the scenario. In our approach, the predicted results can be specified via following two ways. The scenario-based prototype makes it possible to confirm it by the data represented on UI. The CIODD (Concrete Input/Output Data Diagram) of the UML-based scenario makes it possible to confirm the internal system data which would be changed partially by the scenario execution.

7 Conclusion

In this paper, we have proposed the definition process of reliable test cases from the scenarios that have been validated by the customers using the prototype system which was generated by the RA model. As RA model includes the specification of concrete data, the test cases can be systematically defined by concrete input data and the expected result data in the scenario. Therefore, the developers can hand over the validated scenario as the test specification model to the tester. Main task of the tester is to select all adequate test data to exhaustively generated scenario templates.

As the future work, we introduce the order of priority into the flows every conditional branch in the IAD to clarify the flows which the developers want the testers to test preponderantly.

References:

[1] Object Management Group: http://www.omg.org/

[2] Han, K.H., Yoo, S.K. and Kim, B., Qualitative and Quantitative Analysis of Workflows Based on the UML Activity Diagram and Petri Net, *WSEAS TRANSACTIONS on INFORMATION SCIENCE and APPLICATIONS*, Issue 7, Volume 6, 2009, pp.1249-1258.

[3] Machado, L., Filho, O. and Ribeiro, J., UWER: an extension to a Web Engineering methodology for Rich Internet Applications, *WSEAS TRANSACTIONS on INFORMATION SCIENCE and APPLICATIONS*, Issue 4, Volume 6, 2009, pp.601-610.

[4] Teilans, A., Merkuryev, Y. And Grinbergs, A., Simulation of UML models using ARENA, *Proc. of the 6th WSEAS International Conference on SYSTEM SCIENCE and SIMULATION in ENGINEERING*, 2007, pp.191-195

[5] Mibe, R., Kawai, K., Takeuchi, T., et al., A Method for Requirement Acquisition for Web Applications by Usecase Driven Prototyping, *Journal of Information Processing*, Vol.49, No.4, 2008, pp.1669-1679. (in Japanese)

[6] Díaz, J. S., López, O. P. and Fons, J. J., From User Requirements to User Interfaces: A Methodological Approach, *The 13th CAiSE, LNCS 2068*, 2001, pp.60-75.

[7] Elkoutbi, M., Khriss, I., 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.

[8] Jakimi, A., Sabraoui, A., Salah, A. and Elkoutbi, M., Using UML Scenarios in B2B Systems, *Proc. of ICCCE'08*, 2008, pp.964-968.

[9] N Raza, A Nadeem and M Z Z Iqbal, An Automated Approach to System Testing based on Scenarios and Operation Contracts, *Proc. of the 7th QSIC*, 2007, pp.256-261.

[10] Sarma, M. and Mall, R., System Testing using UML Models, *Proc. of the 16th IEEE ATS,* 2007, pp.155-158.

[11] Sarma, M., kundu, D. and Mall, R., Automatic Test Case Generation from UML Sequence Diagrams, *Proc. of the 15th ADCOM*, 2007, pp.60-67.

[12] Nebut, C., Fleurey, F., Traon, Y. L., et al., Automatic Test Generation: A Use Case Driven Approach, IEEE Trans. on Software Engineering, Vol.32, No.3, 2006, pp.140-155.

[13] Huang, C. and Chen, H. Y., A Tool to Support Automated Testing for Web Application Scenario, *Proc. of 2006 IEEE ICSMC*, 2006, pp.2179-2184.

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

[15] Ogata, S. and Matsuura, S., Scenario-based Automatic Prototype Generation, *Proc. of the 32nd Annual IEEE International COMPSAC*, 2008, pp.492-493.

[16] Ogata, S. and Matsuura, S., An Evaluation of a

Use Case Driven Requirements Analysis Using Web UI Prototype Generation Tool, *Proc. of the 9th WSEAS International Conference on APPLIED COMPUTER SCIENCE*, 2009, pp.235-240.

[17] Onishi, A. and Go, K., *Rquirements Engineering*, Kyoritsu Publishing Company, 2002. (in Japanese)

[18] Sommerville, I. and Sawyer, P., *Requirements Engineering: A Good Practice Guide*, John Wiley & Sons, 1997.

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

[20] IEEE Computer Society, *829-2008 IEEE Standard for Software and System Test Documentation*, 2008.