Automatic Generation of SPIN Model Checking Code from UML Activity Diagram and Its Application to Web Application Design

Yutaka Yamada Katsumi Wasaki
Graduate School of Science and Technology, Shinshu University
4-17-1 Wakasato, Nagano, 380-8553 Japan

***Abstract*- The UML activity diagram is suitable for the expression of the work flow, and it expresses behavior at each stage of development from the analysis and the design to the programming. The approach models on the upstream design specification of software by the formal language and verifies it by the model checker, and it attracts attention. In this report, we propose the method of converting automatically the UML activity diagram into the SPIN model checking code PROMELA. We applied to the screen transition design of a Web application example for the evaluation of the proposal method. As a result, we obtained detection and the trace of the counter-example by the model checker, and found a latent bug under limited condition.**

I. INTRODUCTION

It is an important theme to ensure reliability/security of software in today's software development which is getting larger and more complicated. An approach to make a model of upstream design specifications of software in a formal language and use a checking tool to verify the behavior or detect a counterexample is receiving attention. By detecting software defects at the design stage by model checking, we can expect improvement in reliability/security of the software. In addition, when a defect of design in the upstream processes is detected in the subsequent implementation or test process, significant back-job such as review of the design and re-doing implementation is required. Hence, it is desired to verify validity of the design specifications satisfactorily in the upstream processes.

UML (Unified Modeling Language) [1] is often used for analyzing/designing software. UML is a notation for visually grasping models extensively utilizing diagrams. Therefore, models described in UML cannot be verified by the model checker. It is necessary to prepare a model for verification described in a formal language besides models for analysis/designing described in UML [2][3]. Review of specifications is frequently performed in the upstream processes of software development. To convert and derive a verification model from a UML model each time manually will require huge amount of cost and time. It is impractical in actual development.

In this paper, we focus to the UML activity diagram, and propose a method to automatically convert activity diagrams into PROMELA, a specifications description language of a model checking tool SPIN [4][5].

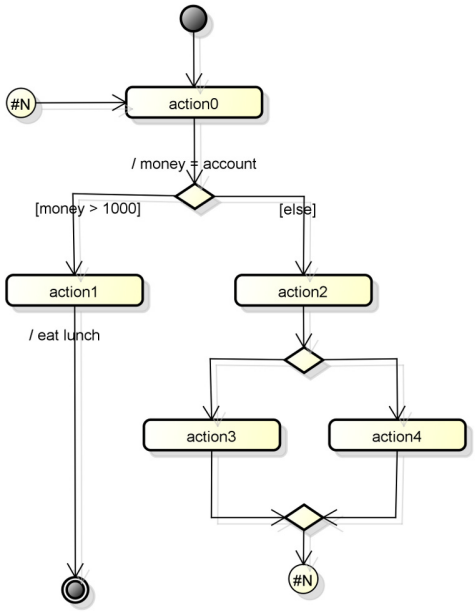

Fig. 1. Example of drawing activity diagram

Activity diagram is one of behavior diagrams separately defined in UML 2.0. It can express behaviors at each stage of development from analysis/design to implementation such as work flow or flow of functions in programming. Finally, we will present the results of applying the proposed method to the screen transition design of a Web application for evaluation purposes.

II. QUASI-FORMALIZATION BY UML ACTIVITY DIAGRAM

In UML 2.0, 13 types of diagrams are defined which are comprised of structure diagrams used for structure-based modeling and behavior diagrams used for behavior-based modeling. UML activity diagram belongs to behavior diagrams and can express the order of, conditions for and control of execution of processes and so on. Fig. 1 shows an example of activity diagram. Individual activities performed by the modeling objective are drawn as actions and the relationships of order of execution between actions are shown by the connecting lines of control flow. If the flow branches off according to conditions, decision node is used. For each branch of decision nodes, branching conditions are described in an element called a guard.

Activity diagram can express various process flows but there is no strict rule for describing actions and treatment of abstraction. Especially, when the natural language is used, implementation is different depending on the interpretation. This vagueness makes the auto convert difficult.

Our aim is to enhance efficiency of specifications verification in the upstream processes by automatically converting activity diagram into the PROMELA that is specifications description language of SPIN. Therefore, we targeted on formulation and simplification of automatic conversion by directly using PROMELA syntax for description of actions and guards. Since PROMELA has similar syntax to C language, for instance, variable substitution, four arithmetic operations and logical operation, software designers can easily use the language without need for relearning the syntax.

III. SPIN MODEL CHECKING TOOL

SPIN is a model checking tool developed and published under the leadership of G.J.Holzmann. It is one of automatic verification tools based on the model checking method which focuses on behavior specifications of parallel systems. Many examples of application of the tool are reported in the industry [5].

A. Overview of SPIN Model Checking

Fig. 2 shows the flow of model checking by a SPIN tool. SPIN generates a limited automaton (Buchi automaton) which handles infinite length words from a model described in PROMELA language and generates a program written in C language called a checker. The checker, when executed after it is compiled with C compiler, generates all state transitions to check if the model satisfies given characteristics.

With SPIN, assert and Linear Temporal Logic (LTL) can be used to express validity of models. Assert can specify counterexamples which should not be taken by the model at the moment by describing assert statement (assert(conditional expression)) in an arbitrary location in the process. LTL is a logical system where temporal operator which can express the concept of time is added to the propositional logic formula. With this, the characteristic that a process written in PROMELA should be operative in time series order can be specified in the form of a logical formula. The running checker will exhaustively search for all states which automaton can take and check if such states satisfy required conditions for validity.

B. PROMELA Language

PROMELA expresses the target system as a collection of processes which perform channel-based communication. Processes executed in parallel express behavior specification of the target system basically by the concept of the automaton. The basic concept of description the behavior of a process is the Guarded Commands. Fig. 3 shows an example of description the guarded commands. This is suitable for expressing nondeterministic and can be used for modeling

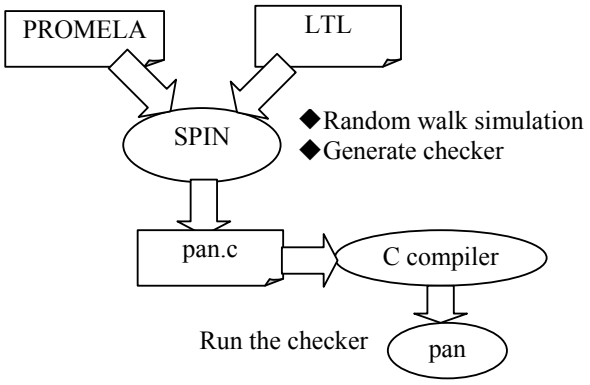

Fig. 2. Flow of SPIN model checking

such an objective as a communication system involving nondeterministic selection.

```
if
  :: Guard 1 → sentences 1
  :: Guard 2 → sentences 2
            …
  :: Guard N → sentences N
fi;
```

Fig. 3. Guard command

When the conditions for guards are satisfied, the statements comprising an action are executed. If the statements are described so that the guards become always true (or if the guards are omitted), nondeterministic selection from all possible choices will be performed.

PROMELA is provided with a scheme called channel for verifying communications between processes. Such problems as loss of message and deadlock etc. can be detected by defining a channel through which communication passes and describing a process so that it sends/receives messages through the channel. The channel can specify a buffer size; declaring zero size will cause synchronous communication to be performed and a size more than zero will cause asynchronous communication.

IV. AUTOMATIC CONVERSION FROM ACTIVITY DIAGRAM TO PROMELA

Models drawn in an activity diagram is automatically converted into PROMELA. While activity diagram expresses the process target data with the object, we also included class diagram in the conversion target since object type and structure are defined with class diagram.

In an activity diagram, nondeterministic is used for selective behaviors by the user which cannot be drawn in advance. Writing a guard in decision nodes which branch according to the user's selection is avoided in PROMELA so that if statement which becomes always true is generated.

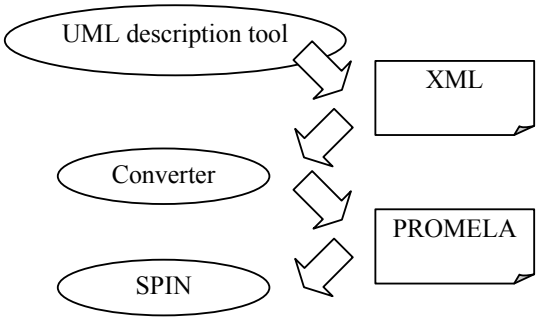

Fig. 4. Flow of automatic conversion

A. Automatic Conversion Procedures

Fig. 4 shows the flow of automatic conversion from UML into a PROMELA model. We implemented the converter using PHP language.

First step is to draw the verification target model with UML activity diagram and the structure of the object treated by the activity with a class diagram. We used astah* Professional[6] of Change Vision as the UML description tool. This tool can output described UML to the XML file. The XML file has diagram components such as node and transition etc. of activity diagram stored as tags [7].

The converter program mainly consists of four components: XML purser, activity purser, class purser and PROMELA formatter. When the converter has received an XML file, it develops the tree structure of XML on the memory by XML purser. Then, class purser extracts elements of the class diagram to create a list of data of the class. Similarly, activity purser extracts elements of the activity diagram and classifies them into object, action node, transition and channel etc. to create treatable intermediate data. When there are multiple activity diagrams drawn, intermediate data are created for each. Lastly, PROMELA formatter outputs necessary information in PROMELA format from the intermediate data using the action node as the frame. If there are multiple activities, process is generated for each.

B. Conversion Specifier Added to Elements of UML Diagram

We examined treatment and notation of instruction information of PROMELA which cannot satisfactorily be expressed with elements of UML in order to realize automatic generation of PROMELA by the converter. In our work, we utilized the "tagged values" of UML for such expression. The "tagged value" is one of extension mechanisms of UML which enables to define arbitrary information for a model element in the {tag=value} format. Table I shows a list of "tagged values" defined for conversion of PROMELA.

We have associated message transmission to a channel with the signal transmission action and message reception from a channel with the event reception action as a method to express a channel on activity diagrams. In addition, we expressed the channel and flow of message in an intuitive manner by connecting signal transmission action and event reception action with a hyper link (transmission → reception).

TABLE I
TAGGED VALUE DEFINED FOR PROMELA CONVERSION

Tag to add info to overall activity diagram	
spin.process	Process name (default value is P1, P2, ...)
spin.number_of_processes	No. of processes (default value is 1)
Tag to add info to action	
spin.label	Label in the process (omissible). Assumed to be used for describing LTL.
Tag to add info to signal transmission action	
spin.channel_input	Message to be transmitted to a channel
spin.channel	Channel name (default value is ch0, ch1, ...)
spin.channel_buffer_size	Buffer size (default value is 0 - synchronous communication)
spin.channel_message_type	Message type (default value is generated in the form of spin.channel_input)
Tag to add info to event reception action	
spin.channel_output	Destination to output message received from a channel

V. EXPERIMENT OF APPLYING THE TOOL TO WEB APPLICATION

Not only conventional Internet-based applications which are represented by online shopping and online banking but also various systems including enterprise business applications have been built as Web applications. As a result, requirement for reliability and security of Web applications is getting higher [8].

We took up a screen transition design for a certain consumer electronics shopping site developed in a project I actually participated in and conducted an experiment to apply the prototyped conversion tool.

A. Shopping Site Specification

Since shopping site sales is directly affected by the ease of use for users of the shopping system, it is required to design screen transition taking flow of user's actions into consideration. With respect to the flow from product selection to completion of order placement, we should always provide more than one route to lead users to completion of order placement in a natural manner without interrupting behavior of the user. In addition, it is also necessary to incorporate requirements relating to the business model of the shopping site such as shipping charge calculation by delivery area and restriction on the payment method according to the amount of the payment.

Fig. 5 shows the screen transition diagram of a shopping site discussed in this study. You can go back to the previous screen or the "Product selection" screen from any entry screen other than the "Order completed" screen. Users may have

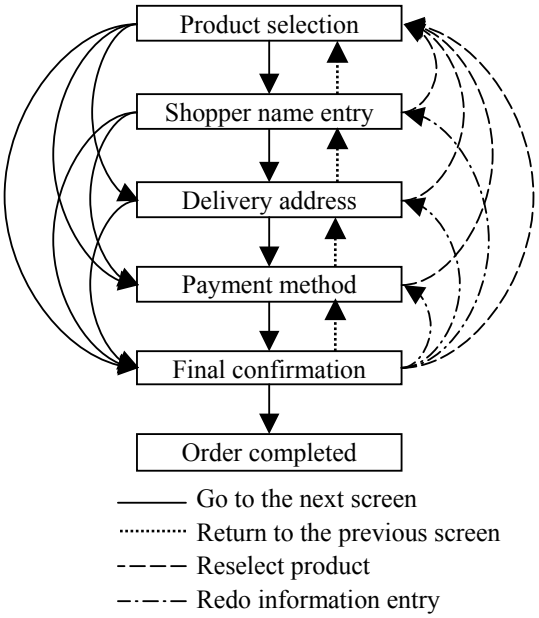

Fig. 5. Screen transition diagram of a shopping site

uncertainty even after he/she has put a product in the shopping cart until he/she confirms the order. Therefore, it is desirable to enable users to reselect products at any time. In addition, showing sales promotion information in each screen may encourage users to additionally purchase related products. In order not to miss such a commercial opportunity, it is necessary that users are always allowed to return to the "Product selection" screen.

And from the "Final confirmation" screen, users should be allowed to do over the entry process returning to an arbitrary entry screen. Then, requiring users to re-enter all items can discourage their willingness to buy. Therefore, the re-entry screen should be determined based on the already entered items so that no more than minimum screen transition is required.

Two categories of shipping charges are offered according to the delivery area. Shipping charge is determined as shown in Table II identifying the delivery area from the entered shipping address.

TABLE II
EXAMPLE OF SETTING SHIPPING CHARGE ACCORDING TO DELIVERY AREA

Delivery area	Shipping charge (yen)
Hokkaido/Okinawa/ Isolated island	1,000
Other areas	500

Available payment methods differ according to the amount of payment as shown in Table III. Therefore, only available selections should be shown to the user according to the payment method to avoid confusion.

TABLE III
EXAMPLE OF REQUIREMENTS FOR PAYMENT METHOD

Payment method	Requirement
Bank transfer	No limitation
Collect on delivery	Available only when the payment amount < 50,000
Credit card	Available only when the payment amount ≧ 3,000

The shopping site deals several thousands of products and there are various product prices. However, we are only required to consider price ranges which can cover combinations of conditions for the delivery area and the payment method. Since the combinations can be covered with five price ranges, we select products from those listed in Table IV for the verification model.

TABLE IV
PRODUCT PRICE

Product	Price (yen)
Product 1	1,000
Product 2	2,400
Product 3	10,000
Product 4	49,000
Product 5	50,000

B. Experiment Results

We first created a model of the specifications described in subsection V.A in an activity diagram. Fig. 6 shows the created shopping cart model shopping1. In addition, we defined in a class diagram the structure of users' order data (order) which are object treated by the model.

We verified the model for the following three items:

1. Hasn't there occurred unintended screen transition?
2. Isn't there omission of any item of order data?
3. Do the amount of payment and the payment method satisfy the conditions?

For item 1, it is described in LTL that "users are to go through the Final confirmation screen before the order is executed". For items 2 and 3, we created an activity to receive order data through a channel besides this activity and verified the order data using assertion.

We generated PROMELA model from the shopping 1 model using the converter and checked the model with SPIN. As a result, the error in Fig. 7 was detected. The error shows that there are cases where the conditions for the amount of payment and the payment method of the verification item 3 are not satisfied. On "Payment method entry" screen, a correct payment method is selected in consideration of the condition by the amount of payment. However, the conditions could become not to be satisfied if the user subsequently returns to the "Shipping address entry" screen and change the address to cause the shipping charge to be changed.

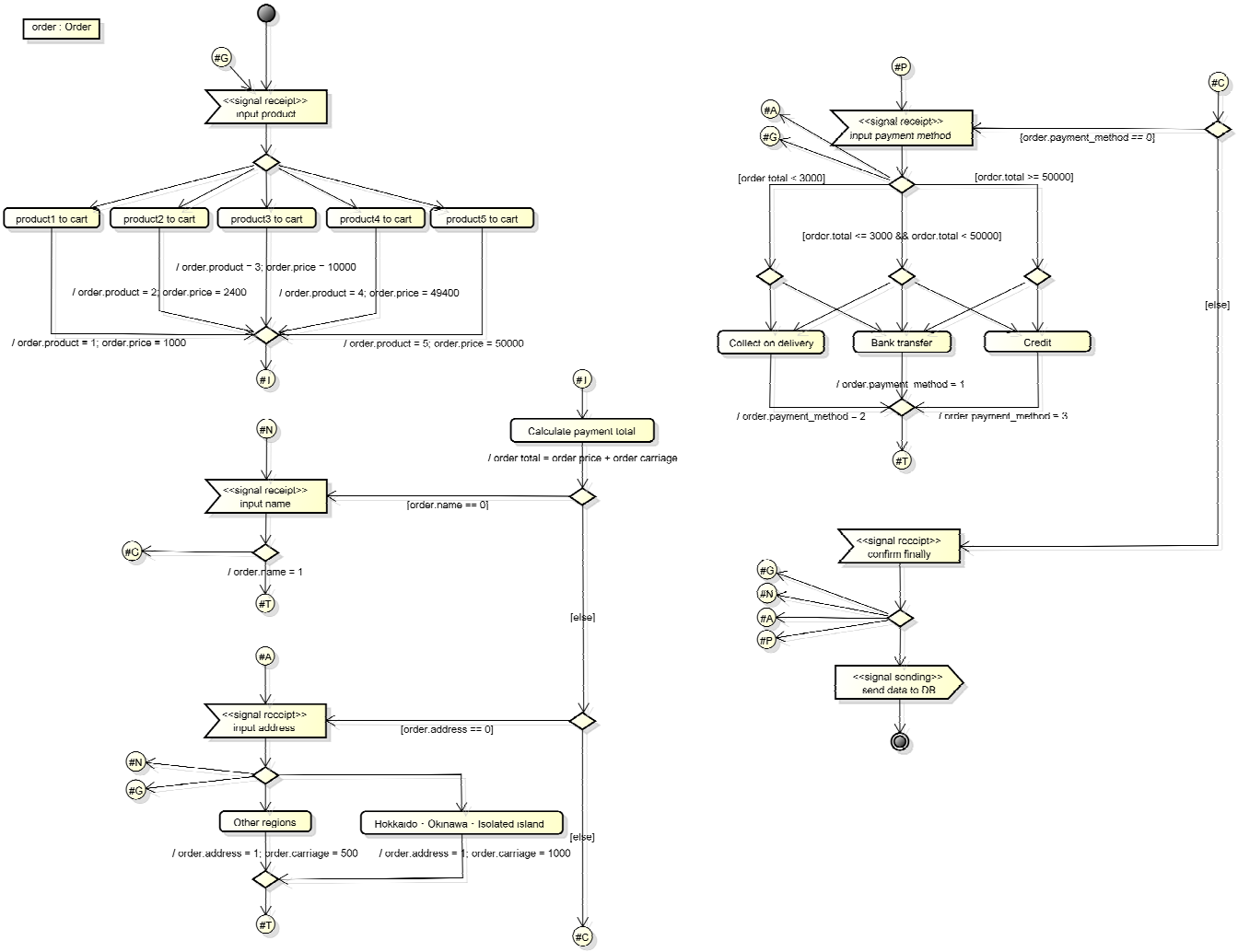

Fig. 6. Shopping cart model shopping1

```
warning: for p.o. reduction to be valid the never claim must be stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)
pan:1: assertion violated (((order.payment_method==1)||((order.payment_method==2)&&
(order.total<50000)))||((order.payment_method==3)&&(order.total>=3000))) (at depth 469)
pan: wrote shopping.pml.trail

(Spin Version 5.2.5 -- 17 April 2010)
Warning: Search not completed
        + Partial Order Reduction

Bit statespace search for:
        never claim             +
        assertion violations    + (if within scope of claim)
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      - (disabled by never claim)

State-vector 240 byte, depth reached 597, errors: 1
    1923 states, stored
     509 states, matched
    2432 transitions (= stored+matched)
       0 atomic steps
```

Fig. 7. Results of checking by SPIN

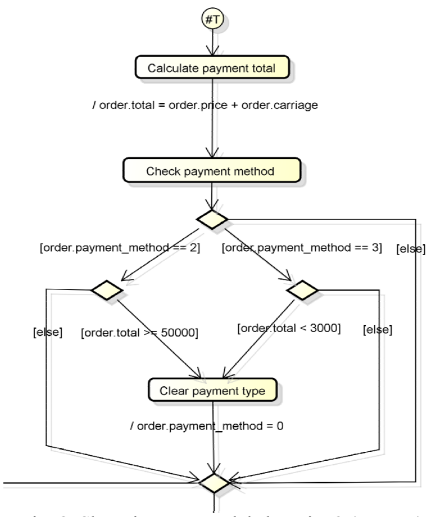

Fig. 8. Shopping cart model shopping2 (excerpt)

Therefore, we have improved the shopping1 model and added a process to clear the entered value for the payment method if the conditions for the amount of payment and the payment method are not satisfied when checked after the "Calculation of amount of payment" process which is always passed through in screen transition. Fig. 8 shows the part added to the activity diagram.

When we checked the model shopping 2 again with SPIN after the correction, no error was detected and validity of the model was testified.

VI. Summary And Future Tasks

In this paper, we have suggested a method to automatically convert a model drawn in UML activity diagram into PROMELA, a specifications description language of SPIN, and described implementation method of the converter and presented an example of applying the converter to an actual case. We obtained satisfactory results applying a prototyped converter to a practical level Web application, though it was under certain conditions and only applied to a part of the model. Software analysis/design is a succession of trial and error and UML is ordinarily rewritten many times. We could experience this time the convenience of the environment where we can check UML model while we are designing it which we have constructed introducing automatic conversion to the flow from UML description to model checking.

Since the converter created this time mainly targeted screen transition design for Web applications, it does not support fork nodes and merge nodes which are parallel processing description in activity diagram. We intend to make the converter support description of parallel processes so that it can deal with business logic part where multiple tasks run in parallel.

In the application experiment, we prepared another activity and checked received order data by assertion in order to verify validity of the shopping site model. While description of behavior and that of characteristic to be verified could be separated this time, there can be cases where those descriptions are mixed to cause it to be difficult to determine if there is a problem in the description of behavior or in the description of the assertion when an error is detected depending on construction of the model. We intend to investigate a scheme to express clearly separating description of characteristics in the future.

Acknowledgment

This research was partially supported by the Ministry of Education, Science, Sports and Culture of Japan, Grant-in-Aid for Scientific Research (C), 23500174, 2011.

References

[1] J. Rumbaugh, I. Jacobson, and G. Booch, "The Unified Modeling Language Reference Manual (2nd Edition)," Pearson Higher Education, 2004.

[2] T. Inoue and Y. Shinkawa, "Validating UML Activity Diagrams through Model Checking," IEICE Technical Report, SWIM2008-19, pp.19-24, 2008.

[3] K. Homma, K. Takahashi, and A. Togashi, "Modeling and Verification of Web Applications Using Formal Approach," IEICE Technical Report, SS2009-8, pp.43-48, 2009.

[4] G.J. Holzmann, "THE SPIN MODEL CHECKER," Addison Wesley, 2003.

[5] S. Nakajima, "Model Checking with SPIN," Kindai Kaguda Sha Co., Ltd., 2008. (in Japanese)

[6] Change Vision, Inc. astah* professional, http://astah.change-vision.com/ja/product/astah-professional.html

[7] OMG, http://www.omg.org/spec/XMI/2.4/Beta2

[8] E.H. Choi, T. Kawamoto, and H. Watanabe, "Model Checking of Page Flow Specification," Japan Society for Software Science and Technology (JSSST), vol.22, no.3, pp.146-153, 2005.