AUTOMATED VISUALIZATION OF INPUT/ OUTPUT FOR PROCESSES IN SOFL FORMAL SPECIFICATIONS

Yu Chen[1] and Shaoying Liu[2]

[1]Graduate School of Computer and Information Sciences, Hosei University, Japan
[2]Faculty of Computer and Information Sciences, Hosei University, Japan

ABSTRACT

While formal specification is regarded as an effective means to capture accurate requirements and design, validation of the specifications remains a challenge. Specification animation has been proposed to tackle the challenge, but lacking an effective representation of the input/output data in the animation can considerably limit the understanding of the animation by clients. In this paper, we put forward a tool supported technique for visualization of the input/output data of processes in SOFL formal specifications. After discussing the motives of our work, we describe how data of each kind of data type available in the SOFL language can be visualized to facilitate the representation and understanding of input/output data. We also present a supporting tool for the technique and a case study to demonstrate the usability and effectiveness of our proposed technique. Finally, we conclude the paper and point out the future research directions.

KEYWORDS

Visualization, SOFL, Formal specification, Data type, Formal methods

1. INTRODUCTION

Formal specification has proved to be effective to help capture accurate requirements and design in software development if used properly together with practical engineering approaches [1]. While this can considerably contribute to the communication between the developers in a software project, it may not effectively facilitate the communication between the developer and the client due to the fact that mathematical expressions in the formal specification can be difficult for the client to understand in general. Therefore, a potential risk that the formal specification may not correctly and completely define what the client really wants will eventually affect the reliability of the software.

To tackle this problem, formal specification animation has been proposed [2]. The common characteristic of the existing animation techniques is to use test data (also called animation data) to dynamically demonstrate the input-output relation for operations defined in the specification. Compared to the reading and understanding technique, animation is proved to be more effective in validating formal specifications against the client's requirements [3][4]. However, our experience and study suggest that the effect of specification animation is rather limited due to the fact that input and output data with complex structures can be difficult to comprehend during animation. Without resolving this limitation, specification animation would be difficult to be transferred to industry for realistic software developments.

In this paper, the researchers put forward a tool supported visualization of input and output data of operations in formal specifications. The proposed visualization technique can be widely applicable to model-based formal notations, such as VDM-SL, Z, and Event-B, SOFL has been chosen, standing for Structured Object-Oriented Formal Language, as the formal notation in our discussions, partly because SOFL has been used in various joint software projects with industry and partly because SOFL offers a comprehensible way to use formal specifications in practical software development.

Our major contributions in this paper are twofold. One is the design of a visualized representation of each type of data provided in the SOFL language. Such a visualization aims to facilitate the expression of the data in a graphical user interface (GUI). The other is the implementation of a software tool supporting the visualization and animation of a single operation called process in SOFL.

The remainder of this paper is organized as follows. Section II briefly introduce the background and related work. Section III discusses the design of visualized representation of various data types. Section IV briefly explain how the visualized representation can be utilized in a single process animation. Section V presents the tool researchers have built to support the proposed technique. Section VI gives a small case study to demonstrate its usability. Finally, in Section VII, we conclude the paper and put forward some future research directions.

2. BACKGROUND AND RELATED RESEARCH

In SOFL, a process performs an action, task, or operation that takes input and produces output. Figure 1 shows a simple form of a process. The process is composed of five parts: name, input port, output port, pre-condition and post-condition. The name of the process always puts in the center of the box. The input port in the left part of the box receives the input data flows and the output port in the right part of the box used to connect output date flows. The pre-condition in the upper part of the box is a condition which the inputs are required to meet, and the post-condition in the lower part of the box is a condition which the outputs are required to satisfy [1].

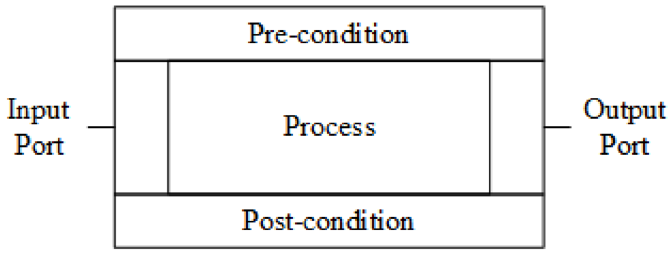


Figure 1. A simple process

Briefly, the process transforms the input data flows to the output data flows. The animation of the process will show the procedure of how the input data flows could transform to output data flows. But in the visualization process, input and output data flows are often composed of a number of complex data types which is difficult for user to understand. Therefore, the effect of animation is quite limited. Without solving this problem, the animation of a process in SOFL formal specifications will be difficult to put into use in industry.

3. Related Work

Formal specification animation attracts a few developers since it provides an effective way to help people especially for simple users to understand the features defined by formal specifications. It helps people to verify whether the specification is consistent with their intended requirements. In this section, we introduce some related work on formal specification animation.

The most common idea of animation is, transforming the specification into one kind of program language. Several animation tools are built based on the specification transformation. PiZA [5] is an animator for Z formal specification. It translates Z specifications into Prolog to generate output variables.

Tim Miller and Paul Strooper introduced a framework for animating model-based specification by using testgraphs [6]. The framework provides a testgraph editor for users to edit testgraphs, and then derive sequences for animation by traversing the testgraph. Gargantini and Riccobene proposed an automatic driven approach to animating formal specifications in Parnas' SCR tabular notation [7]. An important feature of this work is the adoption of a model checker to help find counter-examples that contain a state not satisfying the property to be established by animation.

Liu and Wang introduced an animation tool called SOFL Animator for SOFL specification animation [8]. It provides syntactic level analysis and semantic level analysis of a specification. When performing animation, the tool will automatically translate the SOFL specification into Java program segments, and then use some test case to execute the program.

Li and Liu proposed a novel animation approach called Automatic Functional Scenarios-based Animation. This approach uses data as connection among independent operations involved in a specific behavior to "execute" specifications, and does not translate them to program. Researchers explain how to generate necessary data for animation by modifying an automatic operation function scenario-based test case generation method, and present a case study of applying this animation approach to SOFL specification [9].

4. Design in Data Types

Data types are essential for specifications because they provide a notation for defining data structures used in specifications [1]. It is crucially important to show a variety of data types with proper kinds of visual interface to make the user understand the input and output accurately. Different data types usually represent data with different structure, quantity, and meaning. For the user, the operators defined on the data types are not interesting, the research team therefore decide to ignore them and focus on the structures and values of the data of various types. Researchers design a visual expression for each of the data types in SOFL to facilitate the user in understanding the structure and meaning for each data type accurately and rapidly. Below the definition of each data type will be presented and then its best manifestations will be explored.

4.1 Numeric, Character and Boolean Types

Numeric, Character and Boolean types are basic data types in SOFL. Numeric types contain *nat0*, *nat*, *int* and *real* representing natural numbers including zero, natural numbers without zero, integers and real numbers, respectively. The character type is the atomic unit for constructing identifiers, operators and delimiters for separating different parts in a specification and contains all characters of the SOFL character set. The boolean type contains only two values: *true* and *false*.

Those basic data types are also very easy to read just by their values. So, in the design of visualization researchers suggest to directly show the value of each types.

4.2 Enumeration Type

An enumeration type is a data type consisting of a finite set of special values called elements, members, enumeral or enumerators of the type, usually with the feature of describing a systematic phenomenon [10]. A variable that has been declared as having an enumeration type can be assigned any of the enumerators as a value. For example, the seven days of a week can be seven enumerators named *Sunday*, *Monday*, *Tuesday*, *Wednesday*, *Thursday*, *Friday* and *Saturday*, belonging to an enumerated type named *Week*. Here is an example of an enumeration of *Week* in SOFL:

Week = {<*Sunday*>, <*Monday*>, <*Tuesday*>, <*Wednesday*>, <*Thursday*>, <*Friday*>, <*Thursday*>}

Actually, an enumeration looks like a finite set, so showing the value by a list in visualization is quite convenient. In view of this idea, the *Week* can be showed like Figure 2.

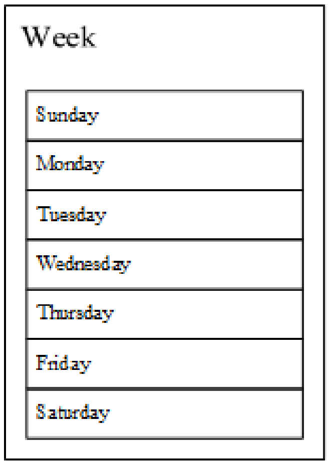


Figure 2. Visualization style for enumeration Week

4.3 Set Type

A set is an unordered collection of distinct objects where each object is known as an element of the set, without any particular order, and no repeated values. In SOFL, a set value of a set type used for process animation is usually finite. For example, a set of programing language names can be:

{"*Java*", "*Pascal*", "*C*", "*C++*", "*Fortran*"}.

Similar to the Enumeration Type, researchers also show the value by a list in visualization as shown in Figure 3. In the design of visualization, researchers focus on the two restrictions: no repeated values and finite quantity. User must give the exact quantity of the items while defining the set, and the tool could remove the repeated values entered by users automatically.

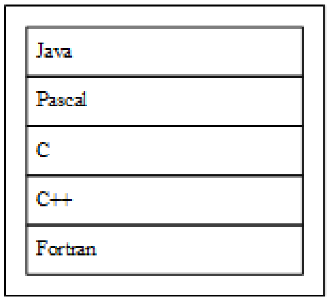


Figure 3. Visualization style for set type

4.4 Sequence and String Type

A sequence is an ordered collection of objects that allows duplications of objects. The difference between sequence and set is that items in a sequence is ordered and allowing element duplication.

Here is an example showing a sequence of natural numbers:

[5, 15, 15, 5, 35]

The visualization of sequence is shown in Figure 4. Researchers add a number at the beginning of each line to show its order.

String is a type which classify all sequences composed of characters. For example, the following is a string value:

"university"

Although a string is a sequence, it is better to display it directly as a whole, from the user-friendly point of view.

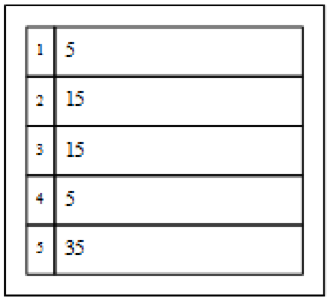


Figure 4. Visualization style for numeric sequence

4.5 Composite and Product Type

A composite type is a data type which can be constructed using the primitive data types and other composite types [11]. In SOFL, the general format of a composite type is:

composed of
  *f*_1: *T*_1
  *f*_2: *T*_2

…
f_n: *T_n*
end

where *f_i* (*i*=1…*n*) are variables called fields and *T_i* are their types.

Each field is intended to represent an attribute of a composite object of the type. For example, Account can be declared as a composite type of three fields:

Account = **composed of**
account_no: *nat*
password: *nat*
balance: *real*
end

Unlike those previous data types, using only values or lists is not sufficient to present an intuitive visualization that is easily accepted by the user. In this respect researchers learn from the concept of many UI design called tab, with different tabs to show different components. So no matter it is simple type or composed type, all can be composed into a compound data of a composed type with perfect visualization as shown in Figure 5.

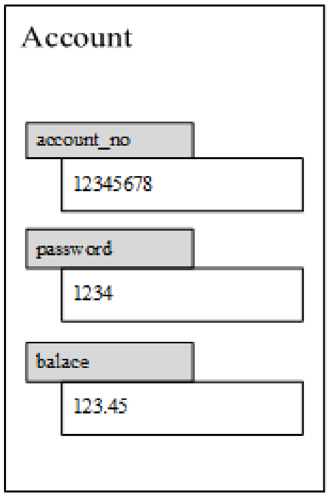


Figure 5. Visualization style for Composite type: Account

A product type defines a set of tuples with a fixed length. A tuple is composed of a list of values of possibly different types. In SOFL, a product type could be defined as:

T = *T_*1 * *T_*2 * … * *T_n*

where *T_*1, *T_*2, …, *T_n* are *n* types.

For example, type Date is declared as

Date = *nat*0 * *nat*0 * *nat*0

The same visualization can be used as in composed type to present, and for simple types like Date, it can be designed as shown in Figure 6.

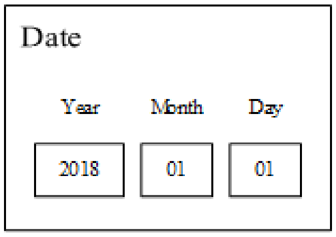


Figure 6. Visualization style for Product type: Data

In this case, it is no need to use tabs since all composed type in Date are non-composed types.

4.6 Map Type

A map, associative array, symbol table, or dictionary is an abstract data type composed of a collection of (key, value) pairs, such that each possible key appears at most once in the collection [12]. A map describes a mapping between two sets. In SOFL, a map is represented with a notation similar to the set notation but use a symbol -> to connect two sets:

{*a*_1 -> *b*_1, *a*_2 -> *b*_2, …, *a*_*n* -> *b*_*n*}

In SOFL, there are two restrictions in map types. One is that all the keys cannot be identical; another is that sets of keys and values are finite. The map type emphasizes the association or correspondence from key to value, and can be considered as a set for each association. So researchers use arrows to represent each set of associations in visualization, while associations use a list similar to set. For example, a map could be defined from twelve months to numbers like:

{*January* -> 1, *February* -> 2, *March* -> 3, *April* -> 4, *May* -> 5, *June* -> 6, *July* -> 7, *August* -> 8, *September* -> 9, *October* -> 10, *November* -> 11, *December* -> 12}

And it can designed as Figure 7.

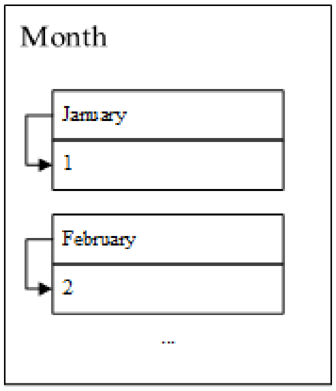


Figure 7. Visualization style for map type

4.7 Union Type

A union is a value that may have any of several representations or formats within the same position in memory; or it is a data structure that consists of a variable that may hold such a value [13]. In other words, a union type definition will specify which of a number of permitted primitive types may be stored in its instances, e.g., “float or long integer”. Contrast with a record (or structure), which could be defined to contain a float and an integer; in a union, there is only one value at any given time. In SOFL, a union type constituted of types could be declared as:

$T = T1 \mid T2 \mid \ldots \mid Tn$

where $T1$, $T2$, …, Tn denote n types.

Since union types are constituted from other different types, they can presented by using visualizations in those types.

4.8 Classes

Class types are common in most object-orient programming languages as extensible types for creating objects, providing initial values for state (member variables) and implementations of behavior (member functions or methods) [14]. In SOFL, a class is a user-defined type, which defines a collection of objects with the same features. The features of objects include attributes, describing their data resources, and operations offering the means for manipulating their data resources and providing functional services for other objects.

In computer science, Unified Modeling Language (UML) is often used to express classes, but users usually merely treat a class as a composite type, so researchers design its visual expression the same as that of composite types.

5. DESIGN IN SYSTEM LOGIC

In SOFL formal specifications, people often use Condition Data Flow Diagram (CDFD) to model how processes work. In the design of main interface, researchers consider the similar style of a process and put data in two ports at each side of the process which could be shown or edited by users, as illustrated in Figure 8.

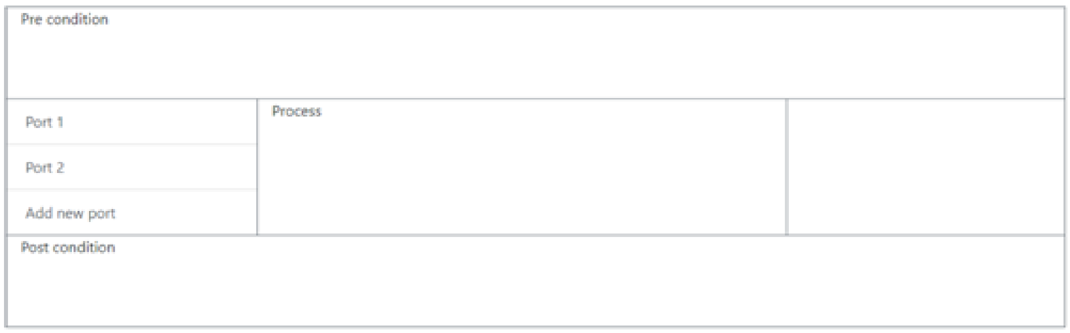


Figure 8. Main interface

Since the space of the main surface are limited, it only shows the concept of input / output data. When the user clicks on the items, the details will be shown in a pop-up box (Figure 9). For input port, the user could add or remove the quantity, types of data and view or edit the values of them. Output port is similar, but the user could only view the values. In the pop-up box, multiple variables are listed, and those composed types will be folded. User could see the detail when clicking to unfold (Figure 10).

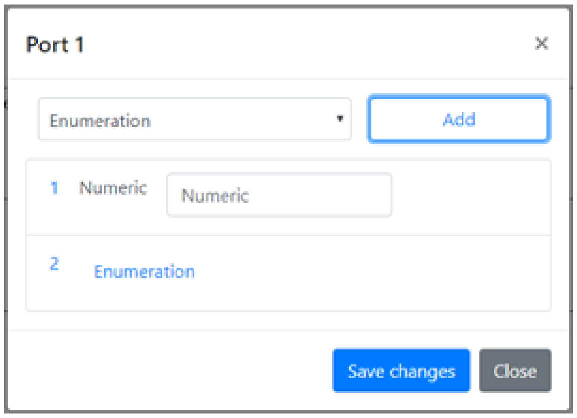


Figure 9. Details of a port

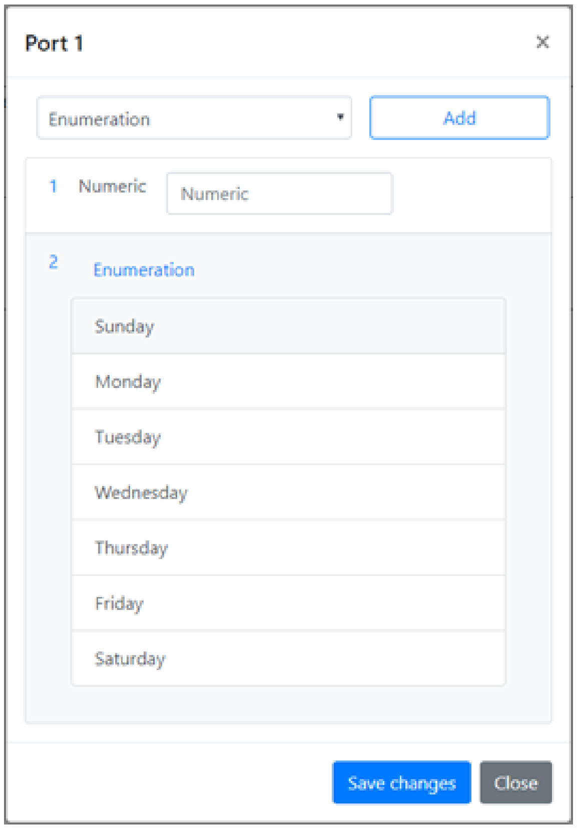


Figure 10. Unfold the enumeration after clicking

6. TOOL IMPLEMENTATION

In the design of network applications, there are mainly two kinds of system architecture. One is called Client-Server (C/S) model which is a distributed application structure that partitions tasks or workloads between servers and clients. The other one is the called Browser-Server (B/S) model which relies on the web technique and is widely used in many systems. By using the browser, users can send requests to server and get response from server by web pages [15].

For a software system, whether choosing B/S or C/S mainly depends on the scene of the user. C/S model is more suitable for those frequently used, complex software programs. B/S model is more suitable for those who are lightweight, for ordinary users of the application. For this tool, researchers choose B/S model to implement since the features for users are simple and publishing for developers are convenient. Software using C/S model always takes long steps for installation or updating which brings terrible user experience. In addition, software using B/S model could be executed in more platforms than C/S model.

The tools utilized for the implementation of our tool could be divided into *front-end tools* and *back-end tools*. In front-end, webpack was used for package management and bootstrap for style designing. In back-end, Django was used for data communication and feature realization.

6.1 Webpack

Webpack is a static module bundler for modern JavaScript applications. When webpack processes your application, it recursively builds a dependency graph that includes every module your application needs, then packages all of those modules into one or more bundles [16]. It is an open-source JavaScript module bunder and takes modules with dependencies and generates static assets representing those modules [17].

6.2 Bootstrap

Bootstrap is a free and open-source front-end library for designing websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. Unlike many web frameworks, it concerns itself with front-end development only [18].

6.3 Django

Django is a free and open-source web framework, written in Python, which follows the model-view-template (MVT) architectural pattern. It is maintained by the Django Software Foundation (DSF), an independent organization established as a 501(c)(3) non-profit [19].

Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusability and "pluggability" of components, rapid development, and the principle of don't repeat yourself. Python is used throughout, even for settings files and data models. Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models.

7. CASE STUDY

In order to show the effect of our solution, a simple process of bank transfer system was designed to show its animation. The main feature is transferring money from one account to another. The structure of an account is defined as:

```
Account = composed of
  account_no: nat
  password: nat
  balance: real
  end
```

In this process, two accounts have been created, one is for transferor named transfer_out, and another is for transferee named transfer_in. Then a transfer process has been created, include the account number of transferor, the account no of transferee, the password and the amount. The process is specified as:

```
process Transfer (transfer_out: Account | transfer_in: Account |
    transfer_out_account_no: nat, transfer_in_account_no: nat,
    transfer_out_password: nat0, transfer_amount: real)
    transfer_out: Account | transfer_in: Account | transfer_result: bool

    pre (transfer_out_password = transfer_out.password and
    transfer_amount <= transfer_out.balance)

    post (transfer_out.balace = ~transfer_out.balance - transfer_amount and
    transfer_in.balance = ~transfer_in.balance + transfer_amount)

    end_process
```

The structures can be saw from the process that there are 3 input ports and 3 output ports as shown in Figure 11. In the input port of *transfer_out* and *transfer_in*, there is only one variable whose type is Account in each port. In the input port of *tranfer_session*, there are four variables including the account numbers of transferor, the account numbers of transferee, the password and the transfer amount. Output ports are similar.

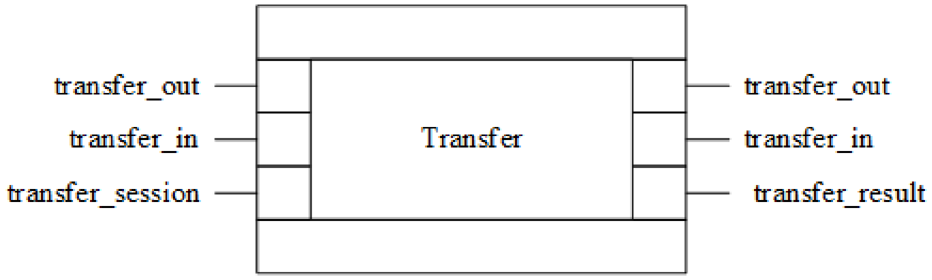


Figure 11. The ports in process Transfer

We entered all the ports in this process to our tool and could see the user interface as illustrated in Figure 12.

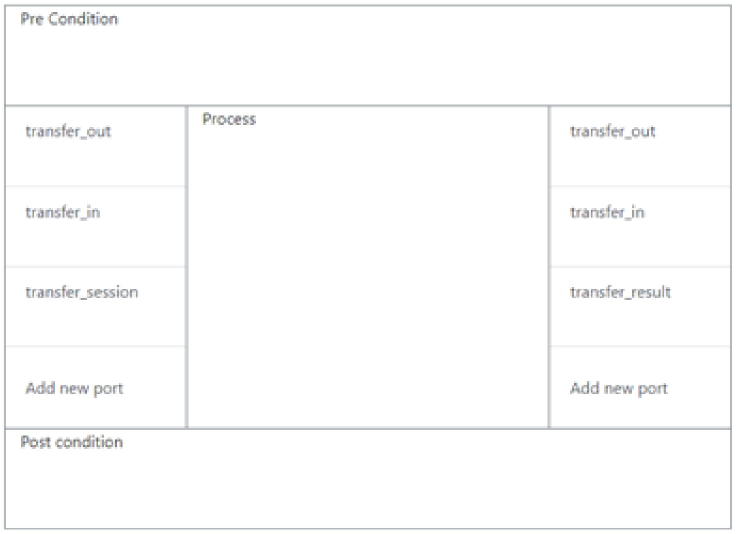
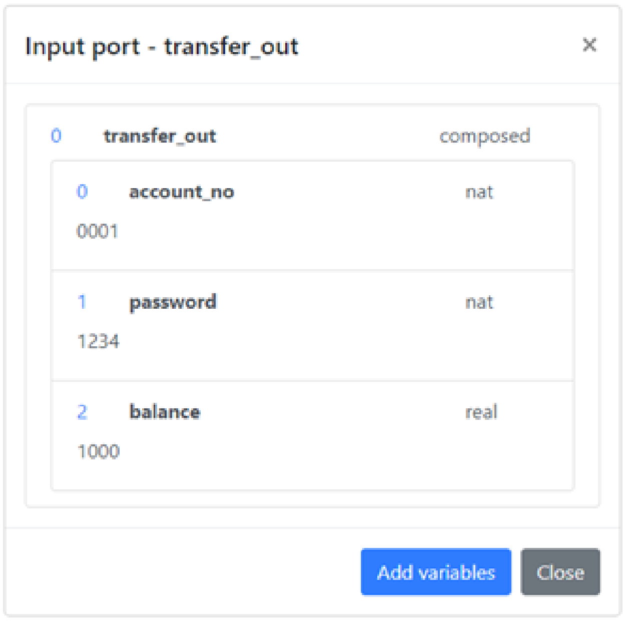


Figure 12. The user interface of process *Transfer*

Then we entered each of the ports. After clicking each port, we can add variables and values as we defined in the specification. We get a user interface shown in Figures 13 – 15.


Input port - transfer_out

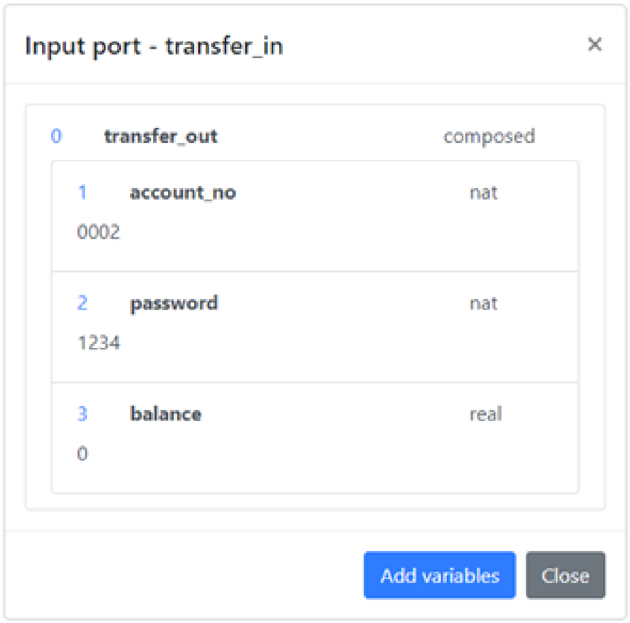
0 transfer_out composed

0 account_no nat
0001

1 password nat
1234

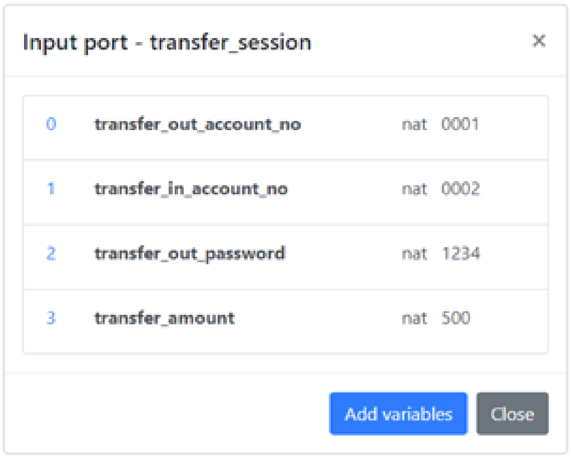
2 balance real
1000

Figure 13. The user interface of the input port: *transfer_out*


Input port - transfer_in

0	transfer_out	composed
1	account_no	nat
0002		
2	password	nat
1234		
3	balance	real
0		

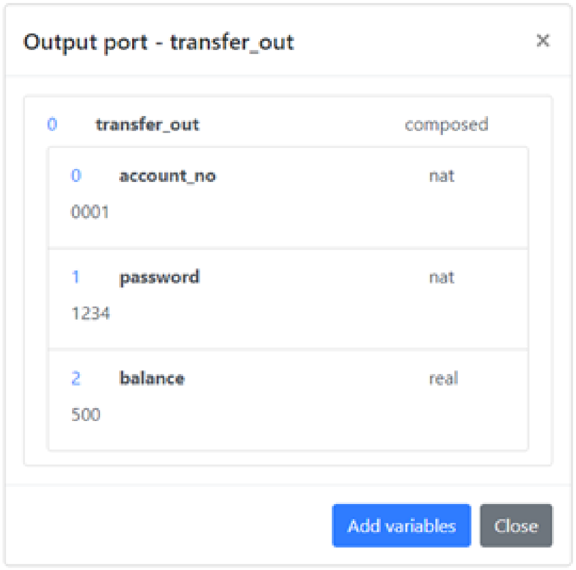
Figure 14. The user interface of the input port: *transfer_in*


Input port - transfer_session

0	transfer_out_account_no	nat	0001
1	transfer_in_account_no	nat	0002
2	transfer_out_password	nat	1234
3	transfer_amount	nat	500

Figure 15. The user interface of the input port: *transfer_session*

After finishing all the input data, we could run the script to animate the process. Then we can check the result from the output port. In the case study, we could see the ports like Figures 16 – 18.


Output port - transfer_out

0 transfer_out composed

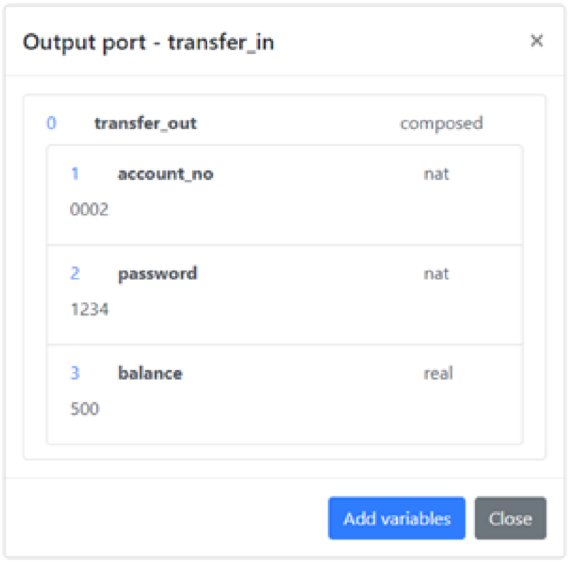
0 account_no nat
0001

1 password nat
1234

2 balance real
500

Figure 16. The user interface of the input port: *transfer_session*

Let's briefly analyze the implementation process. First, the password in tranfer_session is the same as the password in transfer_out, and the transfer_amount is lower than the balance of transfer_out, so the process begins to execute the transferring steps. We can see the differences between Figure 13 and Figure 16 that the balance of account transfer_out has been reduced and between Figure 14 and Figure 17 that the balance of account transfer_in has been increased. Then from Figure 18 we get the transfer_result, the transferring has been successfully executed.


Output port - transfer_in

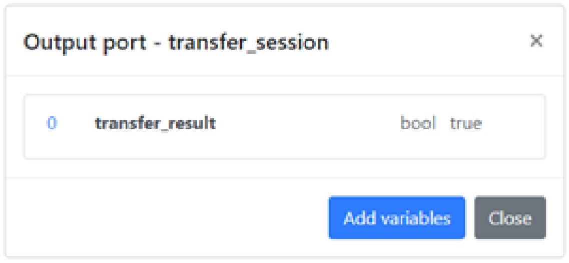
0 transfer_out composed

1 account_no nat
0002

2 password nat
1234

3 balance real
500

Figure 17. The user interface of the input port: *transfer_session*


Output port - transfer_session

```
0    transfer_result    bool  true
```

Figure 18. The user interface of the input port: *transfer_session*

8. CONCLUSION AND FUTURE WORK

In this paper, researchers provided an approach to animating the data types in formal specifications. Researchers focus on single process and try to show all its ports and variables in an intuitive user interface. Researchers analyzed and design the suitable styles to display the data contained in variables. Users could understand what the process is doing by tracking the changes of the values. In the meanwhile, users could enter their own test cases to observe the executing results. This could help user to validate the specification against the user's requirements accurately.

In the future, researchers will continue to finish the module of verification of pre-condition and post-condition. This could help people to verify whether their inputs meet the pre-condition and the outputs satisfy the post-condition. After completing this functionality, the automatic animation of processes will be more integral.

ACKNOWLEDGEMENT

This work was supported by JSPS KAKENHI Grant Number 26240008.

REFERENCES

[1] Shaoying Liu, "Formal Engineering for Industrial Software Development Using the SOFL Method, Springer-Verlag", ISBN 3-540-20602-7, 2004.

[2] Tim Miller, and Paul Strooper. "Animation Can Show Only the Presence of Errors, Never Their Absence", Proceedings of 2011 Australian Software Engineering Conference. IEEEE CS Press, pages 76-88. 2001.

[3] M. Hewitt, C. O'Halloran, and C. Sennett, "Experiences with PiZA: an Animator for Z", in ZUM'97. 1997, vol. 1212 of LNCS, pp. 37-51, Springer.

[4] Tim Miller and Paul Strooper, "A Framework and Tool Support for the Systematic Testing of Model-Based Specications", ACM TOSEM, vol. 12, no. 4, pp. 409-439, 2003.

[5] M. Hewitt, C. O'Halloran, and C. Sennett, "Experiences with PiZA, an animator for Z. ZUM '97: The Z Formal Specification Notation", pages 37-51. 1996.

[6] Tim Miller, and Paul Strooper. Model-Based Specification Animation Using Testgraphs. The 4th International Conference on Formal Engineering Methods (ICFEM 2002). pages 192-203. 2002.

[7] Gargantini, A., and Riccobene, E. Automatic model driven animation of SCR specifications. Fundamental Approaches to Software Engineering, 6th International Conference (FASE 2003). pages 294-3. 2003.

[8] Shaoying Liu, and Hao Wang. An automated approach to specification animation for validation. The Journal of Systems and Software, No.80. pages 1271-1285. 2007.

[9] Mo Li, Shaoying Liu, "Automated Functional Scenarios-based Formal Specification Animation", 2012 19th Asia-Pacific Software Engineering Conference, 2012.

[10] Enumerated type. (2018, May 09). Retrieved from https://en.wikipedia.org/wiki/Enumerated_type

[11] Composite data type. (2018, April 10). Retrieved from https://en.wikipedia.org/wiki/Composite_data_type

[12] Associative array. (2018, May 11). Retrieved from https://en.wikipedia.org/wiki/Associative_array

[13] Associative array. (2018, May 11). Retrieved from https://en.wikipedia.org/wiki/Associative_array

[14] Kim B. Bruce, Foundations of Object-oriented Languages: Types and Semantics, ISBN 0-262-02523-X, 2002.

[15] Xin Wang, Qijun Chen, "Design and Implementation of Real-time Building Energy Visualization Platform Based on Mixed Model of B/S and C/S", 2014.

[16] Concepts, webpack documentation, (n.d.). Retrieved from https://webpack.js.org/concepts/

[17] Webpack. (2018, May 19). Retrieved from https://en.wikipedia.org/wiki/Webpack

[18] Bootstrap (front-end framework). (2018, May 17). Retrieved from https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)

[19] Django (web framework). (2018, May 18). Retrieved from https://en.wikipedia.org/wiki/Django_(web_framework)

AUTHORS

Yu Chen, now is a graduate student at Hosei University (Japan) and Science of Technology of China, researching SOFL Formal Methods.

Shaoying Liu, is Professor of Software Engineering at Hosei University, Japan.