DESIGN AND IMPLEMENTATION OF AUTOMATED VISUALIZATION FOR 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 types 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 2 and section 3 briefly introduce the background and related work. Section 4 discusses the design of visualized representation of various data types. Section 5 discusses the system logic design of the tool. Section 6 presents the tool researchers have built to support the proposed technique. Section 7 briefly explain how the visualized representation can be utilized in a single process animation and gives a small case study to demonstrate its usability. Finally, in Section 8, we conclude the paper and put forward some future research directions.

2. BACKGROUND AND RELATED RESEARCH

Formal specifications are mathematically based techniques whose purpose is to help with the implementation of systems and software. They are used to describe a system, to analyze its behavior, and to aid in its design by verifying key properties of interest through rigorous and effective reasoning tools [5][6]. These specifications are formal in the sense that they have a syntax, their semantics fall within one domain, and they are able to be used to infer useful information [7].

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 data 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].

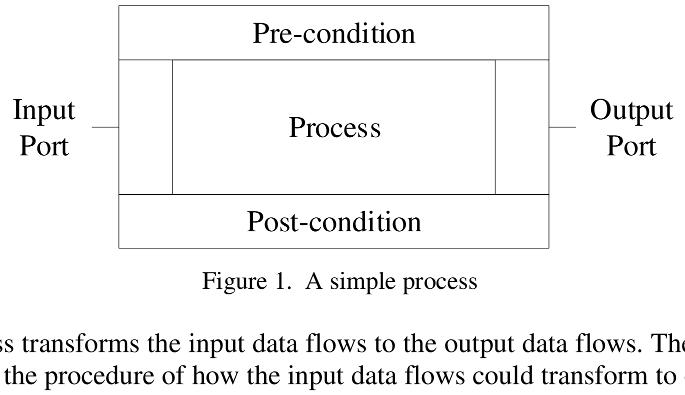


Figure 1. A simple process

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.

As the basic unit of process and data flow, data types affect the actual meaning behind data. A data type is a classification of data which represents type, range, usage and relation of data. It provides a set of values from which an expression may take its values. A data type also defines the operations that can be done on the data, the meaning of the data, and the way values of that type can be stored. The understanding the meaning of data types plays a crucial role in understanding the data.

In SOFL, data types are divided into two categories, built-in types and user-defined types. The built-in types can be further divided into basic types such as numeric, character, enumeration, boolean and compound types such as set, sequence, composite, map, product and union. The user-defined types is known as class that can be defined by the specification writers for constructing well-structured, maintainable, and reusable specifications.

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 [8] 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 [9]. 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 [10]. 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 [11]. 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 [12].

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.

In SOFL, data types can be divided into two categories, built-in types and user-defined types. The built-in types can be further divided into basic types such as numeric, character, enumeration, boolean and compound types such as set, sequence, composite, map, product and union. The user-defined types is known as class that can be defined by the specification writers for constructing well-structured, maintainable, and reusable specifications. 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 type contains *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 understand just by their values. Most of the time, people could understand the meaning of numbers, characters or true / false directly as same as they met in the real world without the knowledge of computer science or programming. So, in the design of visualization, we suggest to directly show the value of those basic 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 [13]. 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>, <Saturday>}.

We can see each value in an enumeration type is written with the form <*x*> while *x* is a string of SOFL characters. If we declare a variable with the type of an enumeration, the variable can take one of its enumerators. So we can image that an enumeration is a list of all possible values of a type and for a variable of this type could take one of those enumerators. It looks like a finite set.

For visualization, showing all the values of the enumerators in an enumeration by the form a list in the interface is quite convenient for user to understand. In view of this idea, the Week can be showed in a list like Figure 2.

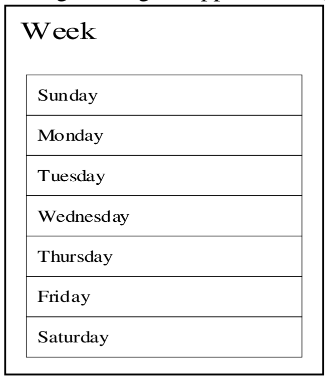


Figure 2. Visualization style for enumeration Week

In Figure 2, we listed all the possible values of enumerators as strings in a list and directly give the name of the enumeration. In this way, user could obtain the information the contained and understand the meaning in the enumeration.

4.3 Set Type

If we want to use a type to store a series of items without their orders, set must be the most suitable data 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 computer science, any set of a set type should be finite. In SOFL, a set value of a set type used for process animation is usually finite and all the elements in a set have the same type. There are two important features in a set, one is the elements are unordered, another is the elements should be distinct. For example, a set of programing language names can be:

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

This is a set consist of five strings. Each of the elements is distinct with others. Changing the order of each element won't change the value of the set. Typically, a set can be declared by applying the set type constructor to an element type like string or enumeration.

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.

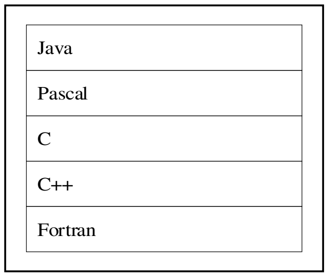


Figure 3. Visualization style for set type

In the visualization of a set type, a list with all elements was shown on the interface. User first must give the type of the elements in a set, then entered each of items while defining the set. The tool could remove the repeated values entered by users automatically since the restriction of no repeated elements.

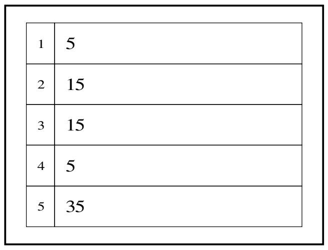
4.4 Sequence and String Type

Sometimes we want to put the items with accurate positions, of course set cannot met this requirement. In this case, sequence could be a better choice. A sequence is an ordered collection of objects that allows duplications of objects. Same as set, the elements in sequence are also have the same type and the size of a sequence is finite. The difference between sequence and set is that items in a sequence is ordered and allowing element duplication. That means if we change the order of the elements in a sequence, the value of the sequence would be different.

Here is an example showing a sequence of natural numbers:

[5, 15, 15, 5, 35].

In the visualization of a sequence, a list with all elements was shown with their orders. While creating the sequence, user should enter each value of the elements after their order numbers. The visualization of sequence is shown in Figure 4.



1	5
2	15
3	15
4	5
5	35

Figure 4. Visualization style for numeric sequence

From the visualization interface in Figure 4, user could easily understand that, the first item in sequence is 5, the second item in sequence is 15, and so on. Each order has only one value and values could be duplicated with different serial numbers.

There is also a special kind of sequence which all the elements in it were character type. Typically we call a sequence like this string. 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, considering the actual meaning in real world, it is better to display it directly as a whole. We can directly show their values like the basic types. It is better to display the string in a holistic manner than to split it and mark it with a serial number.

4.5 Composite and Product Type

In the real world, not all the items we put together are from the same type. Just like a meal, it should consist with vegetables, meat, fruits, seasoning and so on. Most of the time, the items in a collection would be diverse. Computer is the same, there should be a composite type to store items from multiple types. A composite type is a data type which can be constructed using the primitive data types and other composite types [14]. 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. Here we image a simple bank account. A simple account typically consists with account number (a nature number), password (a nature number) and account balance (a real number). In this case, a simple banking account named Account can be declared as a composite type of three fields:

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

We can see the components of the composite type are key-value pairs. The keys are names with practical meanings and the values are basic data types. In a real variable of composite type, the values should be specific values.

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.

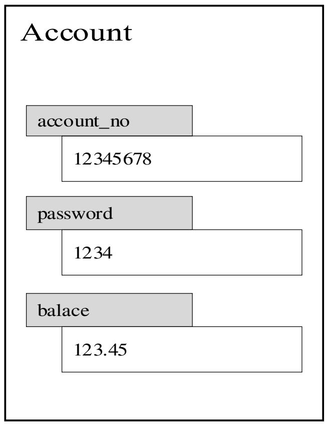


Figure 5. Visualization style for composite type: Account

In the visualization of composite type, we show all the key-value pair in the composite type. The keys are names indicate the actual meaning represented by values. In this way user could easily make the structure clear and understand the values. This is quite important in automated visualization of SOFL formal specification since user would comprehend how the process works by tracking the changes in values.

Composite types could be defined by users according to one's needs. It is powerful but lack of general purpose. Most of the time, there are lots of structures consisted with single or multiple types but had already accepted by most people. 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:

$$T = T\_1 * T\_2 * \ldots * T\_n$$

where T_1, T_2, …, T_n are n types. For example, type *Date* is declared as:

$$Date = nat0 * nat0 * nat0.$$

For the visualization style, 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.

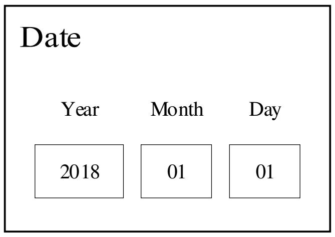


Figure 6. Visualization style for Product type: *Date*

In this case, it is no need to use tabs since all composed type in *Date* are non-composed types. Most of the time, users could not edit the structure of product types, they should just enter or edit the values in each form. Same as composite types, each value has a name to prompt its real meaning.

4.6 Map Type

Sometimes we need a kind of data type to search from one value to another value like a dictionary. An association from one set to another set can be defined as mapping. An abstract data type describing a mapping between two sets, which always composed of a collection of (key, value) pairs was called map, associative array, symbol table, or dictionary, such that each possible key appears at most once in the collection [15]. In SOFL, a map is represented with a notation similar to the set notation but use a symbol -> to connect two sets. Here's the format of a map:

$$\{a\_1 \text{ -> } b\_1, a\_2 \text{ -> } b\_2, \ldots, a\_n \text{ -> } b\_n\}.$$

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}.

The map above expresses an association from the string to numeric. The key of the set, whose type is string, cannot be duplicated, point to the values whose type is numeric. Since the map is a set of maplets (key-value pairs), the order of maplets are not significant, that is, changing the order of the maplets of a map does not affect the map itself.
And it can be designed as Figure 7.

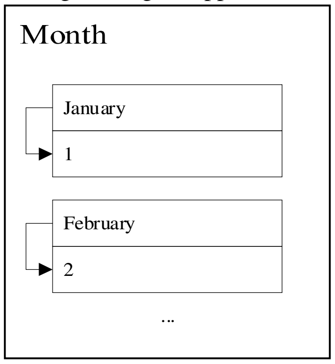


Figure 7. Visualization style for map type

In Figure 7, for each item from key set, there is a arrow pointing to the item from value set. The visualization style intended to emphasize the associations from key to value, regardless the orders of each association. In addition, it should be noted that the direction of the arrow is unidirectional, that is, the value can be searched from the key only, not vice versa.

4.7 Union Type

In actual situations, a composed object always consists of many features from different types. For example, a webpage may contains text, pictures, videos, links, each belonging to a different category. Using single types to represent an compound objects is hard and insufficient. A type composed of several other types called union could solve this problem.

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 [16]. 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. As we can see, a value T can come from one of the types $T1$, $T2$, …, Tn, and types $T1$, $T2$, …, Tn can be different. For example, we can declare the union type *Mixture* that is composed of three types, *string*, *char* and *set of nat* as follow:

$$Mixture = string \mid char \mid set\ of\ nat.$$

Since union types are constituted from other different types, they can be presented by using visualizations in those types while showing all the components as a list.

4.8 Class Type

The types we have already introduced are all build-in types in SOFL such as basic types and compound types. Sometimes, we need to build our own types on the basis of the built-in types to provide more flexible and powerful features over the values contained in the types. Such a user-defined type is called class.

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) [17]. 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.

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.

The visualization style of class is like a list with the name and values of every member, like composite type.

5. DESIGN IN SYSTEM LOGIC

In SOFL, the most important component in specifications is called module. A specification is composed of a set of related modules in a hierarchical fashion. Each module has a behavior represented by a graphical notation, known as condition data flow diagram, and a structure to encapsulate data and processes used in the condition data flow diagram.

A condition data flow diagram (CDFD), is a directed graph that specifies how processes work together to provide functional behaviors. In SOFL formal specifications, people often use CDFD to model how processes work and data flows. Thus, CDFD shows the status of a series of processes.

While we focus on single process, the four parts of the process, input port, output port, pre-condition, post-condition be the focus of our attention. In order to automatic visualize the input / output for processes in SOFL formal specifications, after designing the styles of each data types, we will design the system logic by the following point of views.

5.1 Overall target design

This system is designed to develop a tool that visualizes the processes in SOFL. The processes in SOFL mainly include input modules, output modules, preconditions and postconditions. For the four modules, the system can solve the following problems in a targeted manner: First, the simple and complex data types in the input module and the output module are visualized, so that the user can intuitively feel the actual meaning of the data. The second is to animate the process and establish an animation process from data input, pre-condition verification, process execution, post-condition verification, and data output.

5.2 Logical features design

This topic is mainly aimed at the formal method of SOFL language formal specification is difficult to be understood by the general user, so design the automated visualization tool to display it with graphics and animation, so that customers fully understand the formal specifications and software developers correctly understand the needs of customers To improve the reliability of software development. It mainly includes the following three points:

- Visualize the data and data types of the input module and output module;
- Verify the user's input and output data and pre-conditions and post-conditions to determine the validity of the data;
- Animate the execution of the process from input to output.

For input and output data, ports are used to store the data flows. Each of the ports has a number of variables, and each variable takes a data type. The number of ports and variables in each port is not fixed and could be defined by user. User can also add, remove or edit the number or value of those ports and variables in each port. After user defined or edit the input / output port, the result could be saved.

5.3 User interface design

Focusing the import parts in single process, we consider the similar structure of a process as in CDFD 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.

The main interface is a big box that is similar as the structure of a simple process and it has modules such as input port, output port, pre-condition, post-condition and process name. Each of the modules could be edit by users. The key operations are concentrated on the left and right sides. User could add a new port by click the button and then view the detail of the port by clicking the port.

Since the size of the main surface are limited, we decided to show only the concept of input / output data. The data was stored in ports and each of those ports could be clicked. When the user clicks on the items, the details will be shown in a pop-up box as shown in Figure 9. For input port, the user could view, add, edit or remove the quantity, types of data and the values of them. Output port is similar, but the user could only view the values after the execution of the formal specifications. 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 as shown in Figure 10.

The folded items mostly be the higher levels in the data. When the level of data is too huge, the visualization will become ugly. Most of the time we will only show the first three levels. For further levels, another pop-up box will be much convenient than folded items.

After setting up the types of variables of data, user could enter the values in input port. Then user could execute the custom script to simulate the execution of SOFL formal specifications. After that, the result will be shown in the output port.

All the data including ports, variables with definition and values could be stored and reopened in this tool. Database was used to store those records while user could save and view their status as they like.

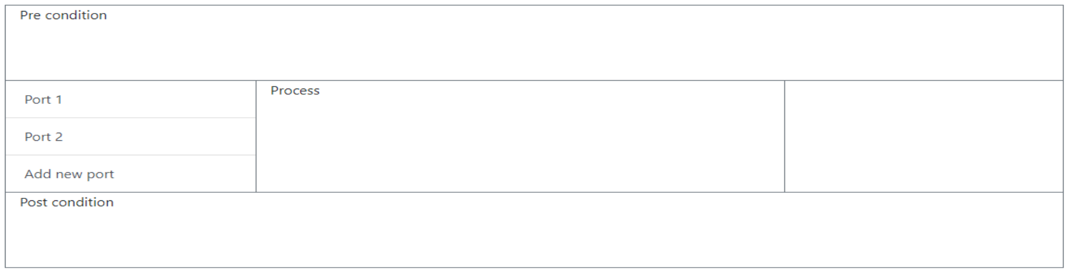


Figure 8. Main interface

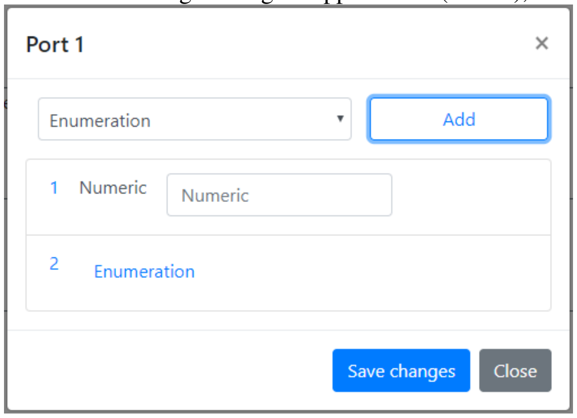


Figure 9. Details of a port

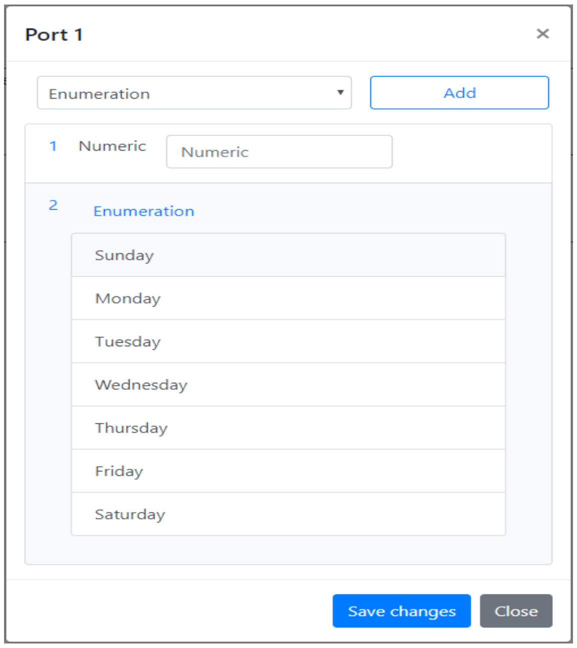


Figure 10. Unfold the enumeration after clicking

5.4 System pattern design

The system development adopts the B/S architecture, and the front end and the back end are independently developed, and finally the communication mechanism is used to complete the system. The backend processing part uses the Django development component, using the MVT pattern (Model-View-Template); the front-end development part uses the Webpack and Vue.js components, using the MVVM pattern (Model-View-View-Model).

In order to implement this system, different platforms and architecture could be chosen. Considering the convenience and easy of use, web platform is the most suitable choice. Comparing with other platforms, user can visit the tool without installation, share data easily and use the tool in multiple devices.

As a web-based system, it uses a four-layer design and ach of the layers is shown as follows:

- Presentation layer: HTML-based webpage design module. For diversified, differentiated data types, structures, and quantities, CSS, Bootstrap, and other development tools are used to visual webpages. JavaScript elements, Vue.js, and jQuery are used to organize page elements and data fields and forms, and rendering dynamically. Webpack is used as an integration tool during development to guide, manage, organize, and package all front-end modules and code.
- Business Interface Layer: Django was used as a web application framework by using the MVT design pattern, building a lightweight, independent web server, while exposing ports to front-end user HTML forms and other data exchanges.
- Business Logic Layer: Written in JavaScript language and Python language to realize three important functions in the business logic of the system. The data access module communicates with the data area, data reading is transmitted upward to the display layer through the service unified interface, and the data of the display layer is received into the data area at the same time. The data rendering module deconstructs the file stored in JSON format in the data area, extracts its parameters associated with the process, and returns it to the data access module for processing by the upper layer; meanwhile, the upper layer data is encapsulated into a JSON format and stored in the data area. The lexical analysis module interprets the formal SOFL formal specification entered by the user as text into executable rules, thereby performing validation with user input data and returning results.
- Data layer: The data of upper layer is accessed using SQL structures or a JSON file.

6. TOOL IMPLEMENTATION

The following formatting rules must be followed strictly. This (.doc) document may be used as a template for papers prepared using Microsoft Word. Papers not conforming to these requirements may not be published in the conference proceedings.

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 [18].

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 front-end development adopts the webpack-based package manager method and is developed in a Node.js-based environment. Developing processes could divided into static page design, template design, style design, logic design, data processing and filling. The design of static pages is primarily based on HTML, based on the latest HTML5 specification. The design of template uses the templating approach of Vue.js. The style design uses CSS, and tools such as Bootstrap are used to optimize the visual effects. The logic design uses the JavaScript language to implement functions, uses jQuery to manipulate page elements, and finally uses Webpack to organize the above items for packaging, debugging, and compilation.

The back-end development uses Python-based Django's open source web application framework, which uses MVT's software design pattern to implement data communication, data processing, and function implementation. The data access module uses Django to read and write operations with the database or data files. The data rendering module converts the parameters and data in the JSON file and the system module.

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 [19]. It is an open-source JavaScript module bunder and takes modules with dependencies and generates static assets representing those modules [20].

Webpack requires Node.js and support lots of extensions. It is highly extensible by the use of loaders so it can achieve very rich features. No matter in development stages or production phase, it can provided efficient, stable and lightweight environment.

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 [21].

Bootstrap provides lots of elements that are very popular in webpages and has very beautiful vision styles. Some of the elements are very suitable for the visualization of data and structures. It can also make a uniform style for each interface in the system.

6.3 Vue.js

Vue.js is a progressive framework for building user interfaces. Unlike other monolithic frameworks, Vue is designed from the ground up to be incrementally adoptable. The core library is focused on the view layer only, and is easy to pick up and integrate with other libraries or existing projects. On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with modern tooling and supporting libraries. Vue.js can be used to bind the data between the database and webpage elements dynamically. It is very suitable for dealing with varied and complex data in web system.

6.4 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 [22].

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. Suppose there are at least two accounts in a banking system, each of those accounts has a balance that means how much money was stored. There is a process named transfer, whose purpose is transfer money from one account to another legally. In order to complete the transfer transaction, several conditions need to be met: the password must be correct and the balance must be enough for this transaction.

Combined with the various data types we introduced before, we can define them as SOFL formal specifications. The definition of an account is written as:

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

where *Account* is a composite type, consists with three items: account number, password and balance.

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 was created, including 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 and transfer_result = true)
  end_process

where the pre-condition requires the password should be correct and the balance should be enough for transaction and the post-condition indicates that the amount of transfer transaction will be deducted from the transferor's account and will be added to transferee's account. Finally, the result of the transfer transaction should return true.

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.

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

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 Figure 13 – 15.

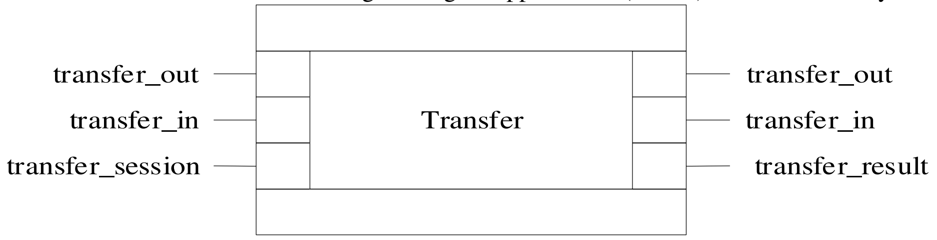


Figure 11. The ports in process Transfer

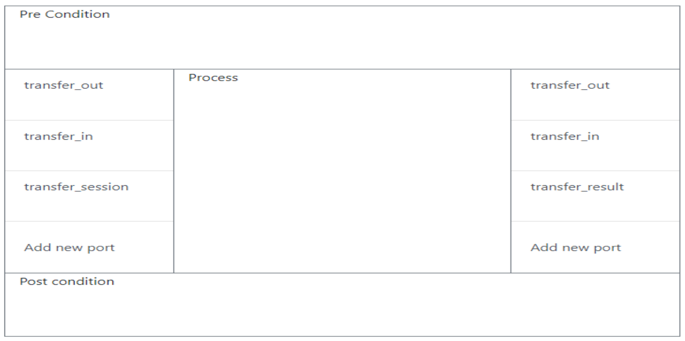
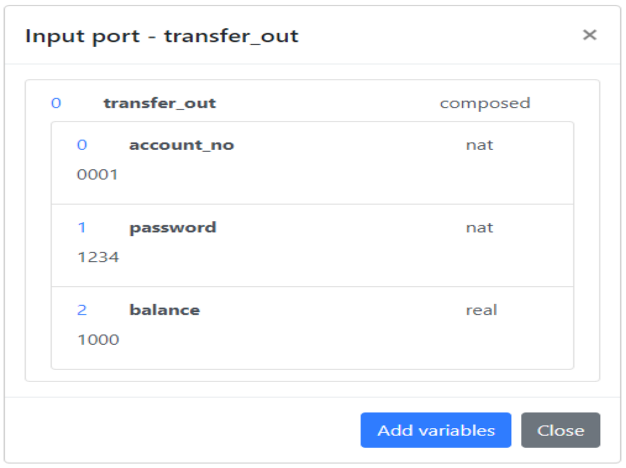


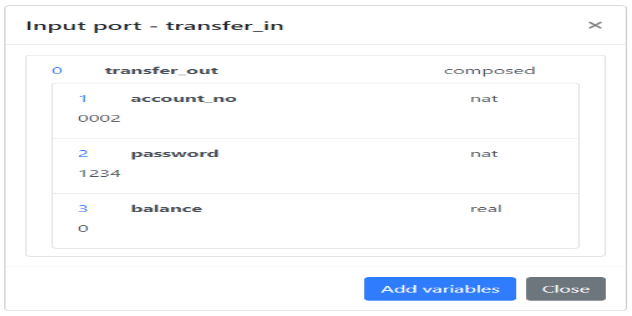
Figure 12. The user interface of process Transfer



Input port - transfer_out

0	transfer_out	composed
0	account_no	nat
0001		
1	password	nat
1234		
2	balance	real
1000		

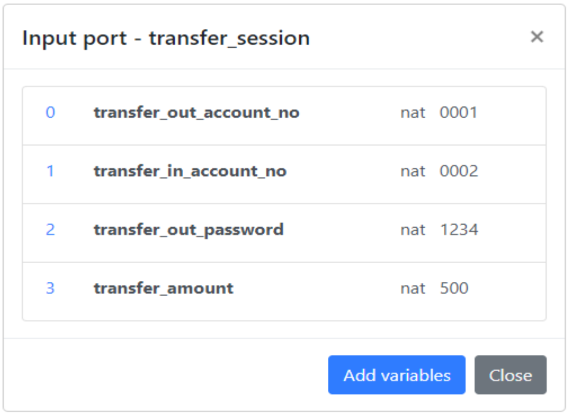
Figure 13. The user interface of the input port: *transfer_out*



Input port - transfer_in

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

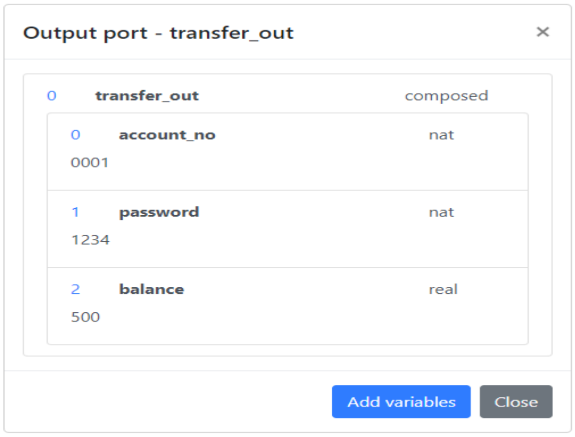
Figure 14. The user interface of the input port: *transfer_in*



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

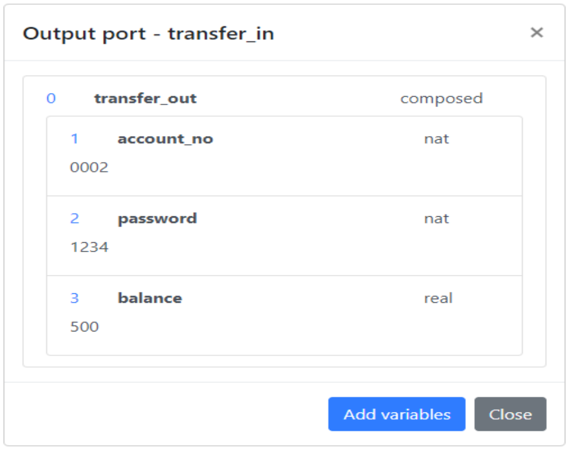
Figure 15. The user interface of the input port: *transfer_session*

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 Figure 16 – 18.



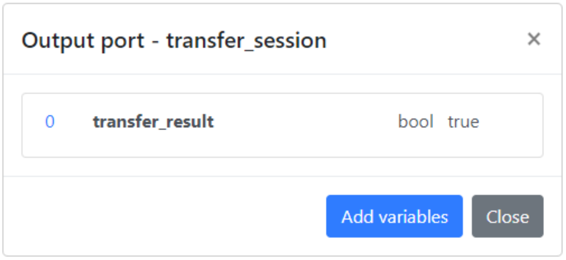
```
0  transfer_out        composed
   0  account_no       nat
   0001
   1  password         nat
   1234
   2  balance          real
   500
```

Figure 16. The user interface of the output port: *transfer_out*



```
0  transfer_out        composed
   1  account_no       nat
   0002
   2  password         nat
   1234
   3  balance          real
   500
```

Figure 17. The user interface of the output port: *transfer_in*


Output port - transfer_session

0 transfer_result bool true

Figure 18. The user interface of the output 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.

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.

As a visualization tool, it is difficult to evaluate its performance because it does not output a fixed result as a basis for evaluation. In the meanwhile, the level of understanding for the input / output data will vary depending on the user's own situation. The challenge is, the types and quantities of data in the input and output modules are dynamic, the layout of how to organize the page display is very demanding. Due to the existence of complex data types, some data types are internally nested with other data types, making the design of the visualization slightly more complicated. The input methods of different data types are also different, and it is technically difficult to match each data on the page with the back-end storage one-to-one.

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. Simultaneously, we can also add the feature to collect the users' feedback and score in the tool for the evaluation of the system performance.

ACKNOWLEDGEMENTS

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] Hierons, R. M.; Krause, P.; Lüttgen, G.; Simons, A. J. H.; Vilkomir, S.; Woodward, M. R.; Zedan, H.; Bogdanov, K.; Bowen, J. P.; Cleaveland, R.; Derrick, J.; Dick, J.; Gheorghe, M.; Harman, M.; Kapoor, K. (2009), “Using formal specifications to support testing”, ACM Computing Surveys, 41 (2): 1. doi:10.1145/1459352.1459354.

[6] Gaudel, M. -C. (1994), “Formal specification techniques”, Proceedings of 16th International Conference on Software Engineering, pp. 223–223. doi:10.1109/ICSE.1994.296781. ISBN 0-8186-5855-X.

[7] Lamsweerde, A. V. (2000), “Formal specification”, Proceedings of the conference on the future of Software engineering - ICSE '00. p. 147. doi:10.1145/336512.336546. ISBN 1581132530.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

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

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

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

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

[17] 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.

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

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

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

[21] 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 a professor researching Software Engineering at Computer and Information Sciences in Hosei University, Tokyo, Japan.