# Design and Implementation of a Tool for Specifying Specification in SOFL*

Mo Li[1] and Shaoying Liu[2]

[1] Graduate School of Computer and Information Sciences,
Hosei University, Tokyo, Japan
mo.li.3e@stu.hosei.ac.jp
[2] Department of Computer and Information Sciences,
Hosei University, Tokyo, Japan
sliu@hosei.ac.jp

**Abstract.** Structure Object-oriented Formal Language (SOFL) is not just a formal language for writing formal specification. It is also an approach and a methodology. SOFL provides a three-step approach for modelling a software system using formal specification. Writing specification can be realized as the most important and fundamental task in this modelling approach. In practice, the activity of writing specification is error-prone, especially the activity of specifying formal specification. We think there are two reasons that cause the difficulty of specifying specification. One reason is that some specifiers may not be familiar with the formal notations used in SOFL, especially the mathematical notations. And the other reason is that there is no tool to guide the specifiers to write specification and make the specifying process easy. In this paper, we show a prototype of a tool that can provide the specifiers with a strong support in the process of specifying specification. This tool provides an integration environment for specifying all kinds of specifications used in SOFL approach, including informal specification, semiformal specification, formal specification, CDFD, and class. And the tool also provides the function to organize the specifications of a same software system.

**Keywords:** formal method, specification, modelling approach, tool.

## 1 Introduction

Formal methods have been recognized as an effective approach for software development. One of the products of formal method is formal specification. Formal specification can describe a software system precisely. The requirement of the software system is specified by formal notations, usually mathematical notations, in the formal specification. The content and structure of formal specifications is different over different formal specification languages. Several formal specification languages exist, like VDM-SL [4], Z [5], Object-Z [6], and so on. SOFL

* This research is supported in part by NII Collaborative Program, SCAT Research Foundation, and Hosei University. It is also partly supported by China 973 program under Grant No. 2010CB328102 and NSFC under Grant Nos. 61133001, 60910004..

S. Liu (Ed.): SOFL 2012, LNCS 7787, pp. 44–55, 2013.

(Structure Object-oriented Formal Language) [1] is one of the these formal languages.

SOFL is not only a formal language for writing formal specification, but also an approach and a methodology. SOFL provides a three-step approach for modelling a software system using formal specification. And a lot of techniques have been created for verifying and validating the formal specification and corresponding implementation program based on formal specification. The combination of SOFL three-step modelling approach and relative verification and validation techniques provide a framework of the entire software system development process. The soul of this framework is the formal specification. It is the final product of modelling process and the basis of following verification, validation and implementation. Writing formal specification can be realized as the most important and fundamental task in SOFL approach.

In practice, specifying formal specification is an error-prone activity. We think there are two reasons that cause the difficulty of specifying formal specification. One reason is that some specifiers may not be familiar with the formal notations used in SOFL, especially the mathematical notations. And the other reason is that there is no tool to guide the specifiers to write specification and make the specifying process easy. The tool support is actually very important. Since there is a lot of special concepts in SOFL, tool support can facilitate the specifiers to deal with these concept. Typically, CDFD (Conditional Data Flow Diagram) is an unique concept in SOFL. Drawing CDFD is required when specifying formal specification. A specific tool that can be used to draw CDFD directly will be very helpful.

In this paper, we show a prototype of a tool that can provide the specifiers with a strong support in the process of specifying specification. This tool provides an integration environment for specifying all kinds of specifications used in SOFL approach, including informal specification, semiformal specification, formal specification, and class. And the tool also provides the function to organize the specifications of a same software system. The prototype is implemented in C# programming language under the environment of Microsoft Visual Studio 2008.

The rest of this paper is organized as follows. We introduce some special concepts of SOFL in Section 2. Specifically, we also explain the three-step modelling approach in this section. In Section 3, we describe the major functions of the tool and explain how these functions support the modelling approach. We demonstrate the architecture of the tool in Section 4 and introduce the implementation in Section 5. Section 6 is related work. And finally, we conclude in Section 7.

## 2 SOFL Three-Step Modelling Approach

In the first step of three-step modelling approach, the informal specification should be specified. The informal specification is written in natural language and is the simplest specification in SOFL. It is used to communicate with the end users or domain experts. The basic unit of informal specification is "*module*".

A module is a group of descriptions of functions. It is like a component in software system.

The second step of the modelling approach is to build semiformal specification. The semiformal specification consists of two aspects. One is process specification, and the other is CDFD. The process specification is plain text specification.The basic unit in process specifications is "process". A process is an independent operation that processes data, and different processes contact with each other via data flows. A group of processes and their relationship are integrated to compose a "module". A module can be considered as a higher level process. Each process can also be decomposed into a lower level module. We use "module" and "specification" changeably in the following paper. Usually these two terms indicate the same thing, namely the process specification of a module.

The counterpart of process specification is CDFD, a graphic specification. For each module, there is a corresponding CDFD. The CDFD uses visual notation to express the relation between different processes that are included in formal specification. A process in a CDFD is treated as a transition and a data flow as a token. When all the input data flows of the process become available, the process will be enabled and executed. The CDFD is both a formal and intuitive notation that is suitable for describing the process specification. Note that, even the semiformal specification is composed by process specification and corresponding CDFD. Drawing a CDFD for a semiformal module is not required by SOFL.

The third, or the final step of the three-step approach is to specify formal specification. The formal specification has the same structure of semiformal specification. It is also composed by CDFD and corresponding process specification. The difference is that the notations used in formal process specification are formal notations, and drawing CDFD for a formal module is not an optional. SOFL requires the specifiers to draw CDFD for each formal module.

Except for the informal, semiformal and formal specification, there is another kind of specification include in the SOFL specifications. It is the "*class*" specification. Since SOFL is an object-oriented formal language, the most important concept of object-oriented design, *class*, is included in the SOFL specification.

The finished specifications should be verified and validated. The incorrect parts of the specifications will be corrected or modified. And then the modified specifications will be verified and validated again. This process will repeat. It is similar to the cycle of software development.

## 3 Design of the Tool

Based on the previous introduction, the entire process of modelling software system by using SOFL can be divided into two stages. The first stage is to specifying specifications used in SOFL, and the second stage is to verify and validate the formal specifications finished in the first stage. As shown in Figure 1, our tool is also separated into two parts and each part corresponds to one stage mentioned above. In this paper, we focus on describing the first part of the tool, namely the part that supports specifying specifications. The tool provides

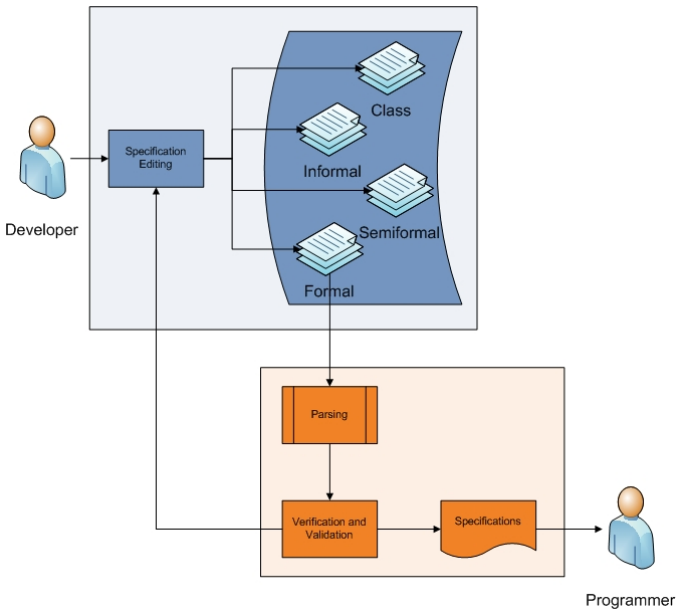


**Fig. 1.** The components of the tool

not only a plain text editor, but also a group of functions that facilitate the users in the specifying process expediently.

According to the three-step modelling approach, for each target system, specifiers should construct the informal specification first, then the semi-formal specification, and finally the formal specification. In order to support this specifying process, our tool provides following 10 major functions:

1. creating a software project
2. adding an informal module
3. specifying informal specification
4. adding a semiformal module
5. specifying semiformal process specification
6. adding a formal module
7. specifying formal process specification
8. drawing CDFD
9. specifying class
10. export specifications, including CDFDs

Most of functions listed above correspond to specifying specifications in SOFL: informal specification, semiformal specification, formal specification, and class. Specially, semiformal and formal specification include process specification and CDFD. The difference is that the CDFD in semiformal specification is optional, but the CDFD in formal specification is required. Since the CDFD is a different presentation of corresponding formal process specification, keeping consistency between CDFD and process specification is one of our major concern.

## 4 Architecture of the Tool

The specifications are the final products of using our tool. To help users organizing the working space, all of the specifications that describe the same software system is grouped. The combination of the specifications is called a "*project*". Figure 2 shows the hierarchy of the specifications. The CDFD with shadow box indicates that the CDFD in semiformal specification is optional.

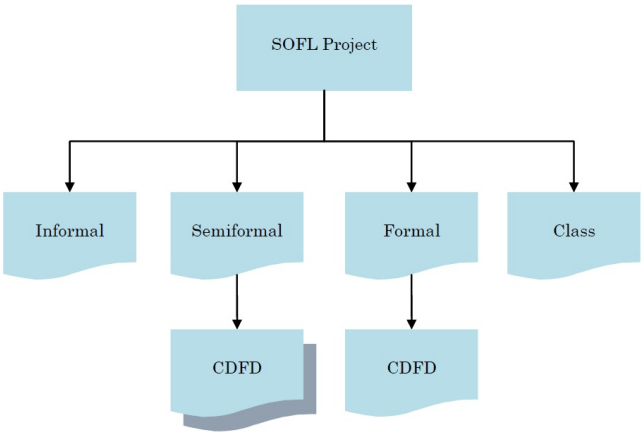


**Fig. 2.** The hierarchy of SOFL specifications

All of the specifications including CDFD are saved as XML files in our tool. We use the module as the basic unit to create a XML file. For example, if users add a new formal module to the SOFL project, two new independent XML files will be created to save the CDFD and process specification, respectively. All the specifications saved in XML files can be exported as other files format for later reference. The XML files will also be used as bases for verification and validation.

**Table 1.** The files used to save specifications

| No. | Suffix | Description |
|---|---|---|
| 1 | .soflproject | save the hierarchy of the SOFL project |
| 2 | .ifModule | save the specification of a informal module |
| 3 | .sfModule | save the process specification of a semiformal module |
| 4 | .sfCDFD | save the CDFD of a semiformal module, the content can be empty |
| 5 | .fModule | save the process specification of formal module |
| 6 | .cdfd | save the CDFD of a formal module |
| 7 | .classSpec | save the class definition |

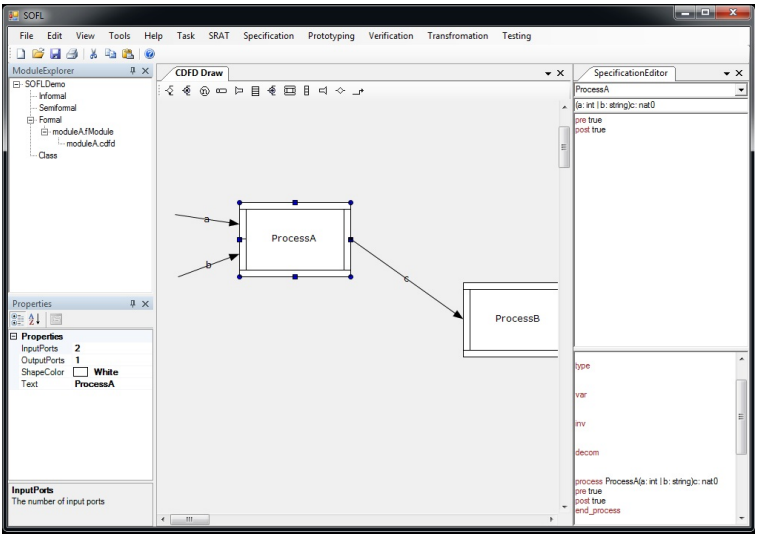


**Fig. 3.** The *Viewer* for specifying formal specification

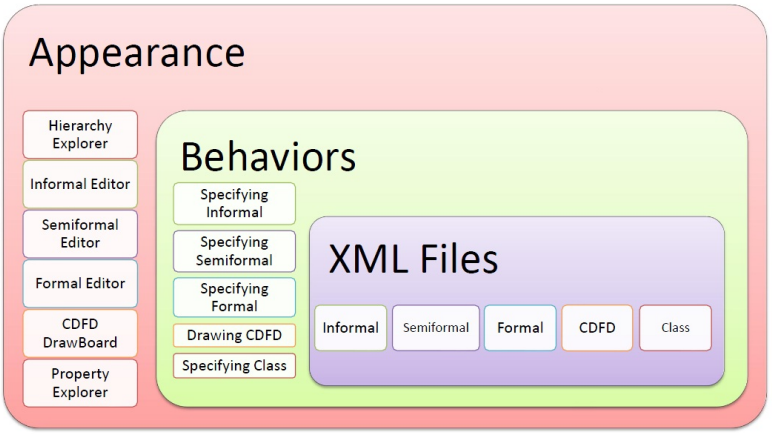


**Fig. 4.** The architecture of the tool

Expect for the XML files, which are used to save the specifications, one specific XML file is used to store the hierarchy of all the specifications, namely, the hierarchy of SOFL project. The internal structure of this XML file is consistent with the structure shown in Figure 2. Table 1 lists all the XML files that will be created by the tool. "Suffix" column shows the suffix of the XML file, and "Description" column explains the content of the XML file.

In our tool, we use different combinations of small windows to provide users with the interfaces for different tasks. The small windows in the tool are called "*Explorers*". Each explorer focuses on presenting one aspect of a specific task. An combination of explorers is called an "*Viewer*" in the tool. For example, Figure 3 shows the default viewer for specifying formal specification. Except for the default viewers, users can rearrange the layout of the explorers to build customized viewers for different tasks. The users who used to use Eclipse or Visual Studio will be familiar with such kind of interface.

Figure 4 demonstrates the architecture of the tool. Users use the functions provided by the tool through different viewers. All the specifications and CDFDs specified by the users are stored in a group of XML files.

## 5 Formal Specification Editor

There is not doubt that the formal specification is the most important part in SOFL. Almost all of the verification and validation techniques are designed based on formal specification. But specifying formal specification is the most challenge task when using SOFL. Even the skilful developer would make mistake when specifying formal specification. And the formal notations in formal specification will sometime confuse the readers, too. In order to help the users to specify and understand the formal specification, SOFL uses a graphic specification called CDFD to simplify the process specification. The CDFD can be realized as a overview of the corresponding process specification. When specifying a formal specification, the user can draw CDFD first, and use the CDFD as a guideline to write process specification.

In our tool, two XML files will be created when the user adds a new formal module to a software project. Figure 3 shows the default viewer for specifying formal specification. This viewer includes four explorers. The two explorers at the left hand side are "*Hierarchy Explorer*" and "*Property Explorer*". The center explorer is "*CDFD Drawboard*", and the explorer at the right hand side is "*Formal Editor*".

The "*Hierarchy Explorer*" displays the hierarchy of the project. The root node of the tree structure in the "*Hierarchy Explorer*" presents the project's name. Under the project's name there are four second level nodes, and each node corresponds to a specific specification type. Different kinds of specifications, namely modules are listed under corresponding node. The viewer for specifying formal specification will open when node presenting a formal module being double clicked.

### 5.1 Drawing CDFD

Drawing CDFD is the first step to specify a formal module. Of course drawing CDFD first is not required, but we strongly recommend it. In the tool, the "*Property Explorer*" and "*CDFD Drawboard*" work together to help users draw CDFD. On the top of the explorer "*CDFD Drawboard*", there is a tool bar.

Each button in this tool bar corresponds to a component in CDFD. Users can add a figure of a component onto the board by clicking the corresponding button. The figure of a component is called an "object" on the board. User can move the objects, resize the objects like using other popular drawing tool. Two objects can be connected via data flow. The point at which object and data flow are connected are called "connector". In different figures, the numbers of connectors are different. For example, considering the two processes "ProcessA" and "ProcessB" shown in Figure 3, there are two connectors at the left side of "ProcessA" but only one connector at the left side of "ProcessB". This is because "ProcessA" has two input ports, while "ProcessB" has only one input port.

Some components of CDFD have their own name or specific properties. In order to edit the name or properties of an object in the board, users can just simply select the object by clicking it, and the corresponding properties will be list in the "*Property Explorer*" automatically. Users can change the name or values of properties, and the modification will change the figure of the object directly. For the sake of space, we just demonstrate one example here. For instance, the selected object in Figure 3 is "ProcessA". It has a name and properties such as input port number and so on. All of the properties are listed in the "*Property Explorer*" at the left-bottom corner. We can see that the input port number is 2, output port number is 1, and name is "ProcessA". In addition, we add an additional property for this component, "ShapeColor". Users can select a predefined color to highlight the object.

### 5.2 Specifying Formal Process Specification

The "*Formal Editor*" explorer in Figure 3 is used to specify the formal process specification, the counterpart of CDFD. The entire explorer is divided into two sections. One is for editing and the other is for displaying. The editing section is separated into three parts. For top to bottom, the three parts are "Component List", "Head Displayer", and "Content Editor". The "Component List" is a drop-down list and all the components of a process specification are listed in it. The component includes *Constant Declaration*, *Type Declaration*, etc. Users can select one specific component to edit each time. Note that the processes defined in the process specification are also listed in this drop-down list. Users can also select a process to edit. Once users select a specific component in the "Component List", the "Head Displayer" will display the head declaration of this component. And the users can edit the content of the selected component. Everything that users type in the "Content Editor" will be displayed in the displaying section automatically. For example, the component selected to edit in Figure 3 is process "ProcessA". We can see the head declaration of "ProcessA" is displayed in the "Head Displayer", and the content in "Content Editor" is also presented in displaying section.

**Table 2.** The events that may effect the consistency

| No. of Event | Event | Effected Process |
|---|---|---|
| 1 | change process's name | 1 |
| 2 | change process's input port number | 1 |
| 3 | change process's output port number | 1 |
| 4 | change data flow's name | 2 |
| 5 | change data flow's type | 2 |
| 6 | add a new process to CDFD | 1 |
| 7 | delete a process from CDFD | 1 |
| 8 | add a new data flow to CDFD | usually 0 |
| 9 | delete a data flow from CDFD | 2 |
| 10 | connect a process and a data flow | 1 |
| 11 | disconnect a process and a data flow | 1 |

### 5.3 Keeping Consistency Mechanism

One of the mistakes made in specifying formal specification is inconsistency between CDFD and corresponding process specification. In our tool, we provide a well designed mechanism to keep the consistency between CDFD and process specification. We defined total 11 events that can effect the consistency. These 11 events are listed in Table 2. The column "Event" is the description of the event, and the column "Effected Process" is the number of processes whose definition will be changed by the event. For instance, the forth raw of the table indicates that the event of changing the name of a data flow will effect two processes in the process specification. The two processes are connected by the data flow. When this event happen, the corresponding definition of these two processes in the process specification should be changed.

In order to make this mechanism work well, we require all of these events must happen in the process of drawing CDFD. It means all the change and modification described in Table 2 must be done through "CDFD Drawingboard" and "Property Explorer". And the content of process specification will be modified automatically. Users cannot do these change or modification in the "Formal Editor". Of course we can provide users with a check list based on Table 2 for inspecting the consistency, but we think build this mechanism into the tool will give the users a very good guide to specify the specification and it can avoid mistakes in inspecting the consistency.

### 5.4 Specifying Semiformal Specification

Specifying semiformal specification is similar to specifying formal specification. Two specifications have almost same structure. In our tool, the default viewer for

specifying semiformal specification is similar to the default viewer for specifying formal specification. Figure 5 is the snapshot of default viewer for specifying semiformal specification. Compare to Figure 3, the "CDFD Drawboard" and "Property Explorer" disappear. This is because drawing CDFD is optional in the process of specifying semiformal specification. And the "Formal Editor" is replaced by "Semiformal Editor". The "Semiformal Editor" looks like "Formal Editor". The only difference is the tool bar in "Semiformal Editor". We can see from Figure 5 that there are two buttons in the tool bar. The two buttons correspond to adding a *process* and a *function* to the "Component List" respectively. When specifying formal specification, adding a process to specification is an event that will effect the consistency between CDFD and process specification. Therefore, a process can be added to the process specification by adding a process object to the CDFD. But in the process of specifying semiformal specification, we do not force the users to draw CDFD, drawing CDFD is just an option. If the users do not want to draw CDFD, they can use the two buttons in the "Semiformal Editor" to add process or function definition to the semiformal process specification. In this case, there is no need to check the consistency between semiformal process specification and its corresponding CDFD, and we do not provide the relative functions in the tool.

If users want to draw CDFD for a semiformal process specification, he or she can just simply double click the node with the suffix ".sfCDFD", and the same "CDFD Drawboard" and "Property Explorer" shown in Figure 3 will be opened. Note that the occurrences of the events listed in Table 2 will not be presented in "Semiformal Editor".

## 6 Related Work

The tool described in this paper provides several functions that can make the process of specifying SOFL specification easier. There is another tool that can support specifying SOFL formal specification is introduced in [7]. Compare to our tool, this tool does not provide the function to check and keep the consistency between the CDFD and formal process specification. Furthermore, the flexibility of this tool is limited, while our tool is designed not only for providing support for specifying specification, but also for utilizing the latest published verification and validation techniques. The prototype introduced in [8] is designed to exact functional scenarios from SOFL formal specification. It reads formal specification from XML files and generate all possible functional scenarios. The functional scenario is the basic in several verification and validation techniques.

Several tools have been to support different formal languages or formal methods. Overture [9] is a community-based project of open-source tools to support modelling and analysis in the design of software systems using VDM. It provides several functions such as editing, checking, debugging, etc. B4Free [10] is a set of tools for the development of B formal models. B4Free offers a graphic representation and numerous functionalities to present and manage projects in B language. And it also provides automatic management of dependencies between

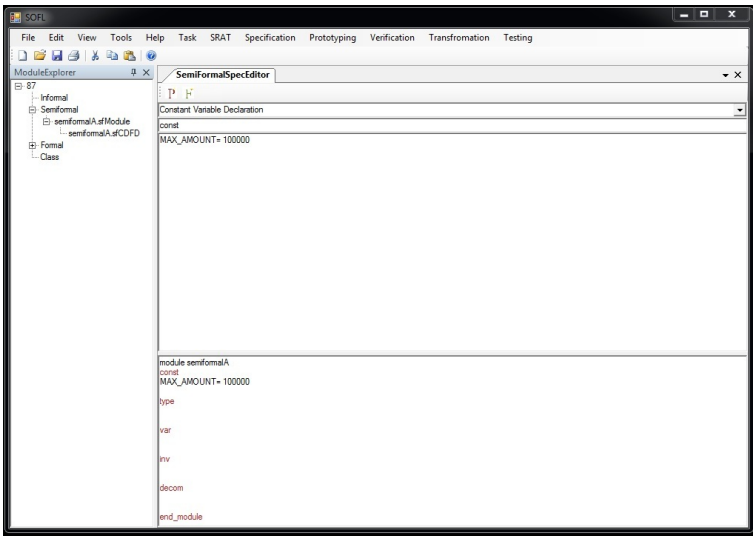


**Fig. 5.** The default *Viewer* for specifying semiformal specification

B components. The Rodin Platform [11] is an Eclipse-based IDE for Event-B that provides effective support for refinement and mathematical proof. PiZA [12] is an animator for Z. It translates the Z specifications into Prolog to generate output variables.

## 7 Conclusions and Future Work

In this paper, we present the prototype of a tool that supports the entire specification specifying process when using SOFL three-step modelling approach. The tool offers customized editor for each kind of specification and numerous functions facilitating specifiers. These functions include manage SOFL project, checking consistency between CDFD and formal process specification, etc. All of the specifications specified by users and the hierarchy of project are saved in XML files. The XML files of formal specifications will be used as basis for later verification and validation. If the users want to distribute the specifications, the specifications can be exported as different formats. The process specification can be experted as MS Word file or PDF file, and the CDFD can be exported as JPEG file or BMP file.

As show in Figure 1, the goal of our tool is not to provide an editing environment only. We want to provide an entire environment for the SOFL three-step modelling approach and the following verifying and validating process. Obviously, building a powerful specifying tool is a good start. On the one hand, verification and validation cannot be performed without formal specification. On the other hand, a well defined and uniform formal specification file can simplify the process of tool support verification and validation. In the future, our

work can be separated into two parts. The first part is to refine the exiting tool. Enhance the usability of the tool. The second part of our work is to build a parser for formal specification. Almost all of the verification and validation techniques are based on analysing the formal specification. Therefore, build a parser is necessary.

## References


1. Liu, S.: Formal Engineering for Industrial Software Development Using the SOFL Method. Springer (2004) ISBN 3-540-20602-7
2. Liu, S., Sun, Y.: Structured Methodology + Object-Oriented Methodology + Formal Methods: Methodology of SOFL. In: 1st IEEE International Conference on Engineering of Complex Computer Systems, pp. 137–144. IEEE Press, Ft. Landerdale (1995)
3. Liu, S., Shibata, M., Sato, R.: Applying SOFL to Develop a University Information System. In: 6th Asia-Pacific Software Engineering Conference, pp. 404–411. IEEE Press, Takamatsu (1999)
4. Dawes, J.: The VDM-SL Reference Guide. Pitman (1991)
5. Diller, A.: Z: An Introduction to Formal Methods. John Wiley & Sons (1994)
6. Meira, S.R.L., Cavalcanti, A.L.C.: Modular Object-Oriented Z Specifications. In: 5th Annual Z User Meeting on Z User Workshop, pp. 173–192. Springer, London (1991)
7. Liu, S.: Integrating top-down and scenario-based methods for constructing software specifications. In: 8th International Conference on Quality Software, pp. 105–113. IEEE Press, Oxford (2008)
8. Li, M., Liu, S.: Automatically Generating Functional Scenarios from SOFL CDFD for Specification Inspection. In: 10th IASTED International Conference on Software Engineering, Innsbruck, Austria, pp. 18–25 (2011)
9. Overture: Formal modelling in VDM, `http://www.overturetool.org/`
10. B4Free, `http://www.b4free.com/index-en.php`
11. Event-B.org, `http://www.event-b.org/`
12. Hewitt, M.A., O'Halloran, C.M., Sennett, C.T.: Experiences with PiZA, an animator for Z. In: Till, D., Bowen, J.P., Hinchey, M.G. (eds.) ZUM 1997. LNCS, vol. 1212, pp. 37–51. Springer, Heidelberg (1997)