Formal Description of Pipes-filters Achitecture Style

Yunsai Zhai [1, a], Lichen Zhang [2, b]

[1]Faculty of Computer Science and Technology, Guangdong University of Technology, Guangzhou, 510090, China

[2] Faculty of Computer Science and Technology, Guangdong University of Technology, Guangzhou, 510090, China

[a]email: zys_yunsai@163.com, [b]email: zhanglichen1962@163.com

Keywords: Formal Method; Z Language; Formalization of Filters; Formalization of Pipelines; Formalization of Pipes-filters

Abstract. In order to solve some nonstandard and imprecise problems of non-formal methods, the formalization method based on strict mathematical is put forward software architecture of formalization not only can clearly describe the software architecture style, and makes design of architecture easy to understand and implement. In this paper, using the Z specification language describes the formalization of the pipes-filters architecture style that the pipes-filters model is oriented to data flow.

Introduction

From the recent exploration of software architecture, software architecture is currently largely still based on non-formal research.The non-formal graphs and texts are still the common architecture description in software development, which directly affects the reusability of the system and the description of the interaction between the components and their system. In order to solve this problem, the formal methods is introduced. The formal method[1] is a tool to solve the theoretical study of Computer Science and Software engineering practice related issues, and it is built on rigorous mathematical methods and arguments. Set theory, algebra theory, mathematical logic, structural type theory, programming language theory and other mathematical theories have formed the oretical basis of formal methods. Formal method, as the most rigorous method in current software development, is an important means to improve the security and reliability of software systems. The application of formal methods requires the support of formal specification language.

Z Language

Z language[2][3] is a kind of typical formal specification language. It is based on first order predicate logic and set theory as the main mathematical foundation and is using the state machine as a model. Z language is using the standard logical operators (∧ ,∨ ,¬, etc.) and set operators (∈ ,∪ ,∩, etc.) and their conventional definition.

Z language provides the structure of a framework (“schema”) and is used to describe a specification state space and operation. Z protocol language is using this schema structure type to represent the architecture style, and its expression form is as follows:

```
schema TypeName{// “TypeName” is a type name
   the statement part    // defined state variables
   the predicate part    // describe the relationships between variables
}
```

The Formalization of Filters

In the pipes-filters system[4] [5], the filters as a component are the basic unit of data processing,

and are used to convert and process data streams. Each filter has an unique identifier (“Filter_id”). Each filter has an input port (“in_ports”) and an output port, The system through the

© 2016. The authors - Published by Atlantis Press

input port read data stream, after the data flow is processed by the internal program, the system outputs new data at the output port. “Filter” is defined as the set of all filters; “DATAPORT” is defined as the collection of all ports; “DATA” is defined for the system to deal with the data set; “Alphabets” is defined as a port mapping function and is mapped to a data sequence by port; “transtitions” is defined as a processing function, in which the domain of definition is consisted of input data, and range of value is consisted of output data; “STATE” is defined as all the states of the filter set.

In the behavior of the filters, the result of a single step operation is that[6]: 1. Some of the data in the input port is removed; 2. These data are processed according to the current state of the filter; 3. Current status of filter is changed; 4. Processed data is generated on the output port. The one-step operation of filter will make some input data convert into output data. The frame of FilterCompute encapsulates such a one-step operation. So we can use “Filter” and “FilterCompute” to define filter in the way of formalization, and its expression form is as follows:

schema Filter{
f: Filter
Filter_id: Filter
in_ports,out_ports: DATAPORT
alphabets: DATAPORT +→ DATA transtitions:(STATE×(DATAPORT+→seq DATA))
↔ (STATE×(DATAPORT+→seq DATA))
states: STATE
start,curstate: STATE
instate,outstate:DATAPORT+→seq STATE
in_ports∩out_ports= ∅
dom(alphabets) =in_ports∪ out_ports
start ∈states
∀state1, state2 : STATE
ps1,ps2:DATAPORT+→ seq DATA
●((state1,ps1),(state2,ps2))∈ transitions=>state1∈
states∧ state2∈ states
// It uses the symbol “●” to separate quantifier constraint and predicate expression.
∧ dom (ps1)=in_ports∧ dom(ps2)=out_ports
∧ ∀i:in_ports ●ran (ps1(i))⊆ alphabets(i)
∧ ∀o:out_ports ● ran (ps2(o))⊆ alphabets(o)
curstate∈f.states
∀p:dom(instate) ● ran (instate(p))⊆ f.alphabets(p)
∀p:dom(outstate) ● ran(outstate(p))⊆ f.alphabets(p)
}
schema FilterCompute{
△ Filter //The symbol of "△ " is used to cite the framework of “Filter”
f′=f // The next state of the filter becomes the current state
∃ in,out:DATAPORT+→seq DATA
●((curstate,in), (curstate′,out))∈ f.transtitions
∧ ∀p:f.in_ports ●instate(p)=in(p)⁓instate′(p)
∧ ∀p:f.out_ports ●outstate′(p)=outstate(p)⁓out(p)
}

The Formalization of Pipelines

In the pipes-filters system, the pipeline as the connection is a highroad to realize data transmission and it represents the data output from one filter to another. Each pipeline has a unique identity(“Pipe_id”), and there is an input port (i.e. the source port) and an output port (i.e., the pool port). Each channel has a different source and pool for sending and receiving data messages. At any

moment, the pipeline of the two ports have data resides. Any of the two pipelines are different that identification is different and transmission of the data is also different.

In the behavior of the pipelines, the result of a single step operation is that: it will send certain data sequences from the source port of data ("source_ port") to the pool port of data("sink_port"). In the process of removing and sending the data, the order and the content of the data still remain the same, that is, the data in the source port and the pool port is the same. The two filters connected by a pipeline ("p") are the source point filter ("source_filter") and the end filter ("sink_filter").The source port of the pipeline is connected to the output port of the source filter, and the pool port of the pipeline is connected to the input port of the end filter. So we can use "Pipe" and "PipeCompute" to define pipeline in the way of formalization, and its expression form is as follows:

schema Pipe{
p: Pipe; Pipe_id: Pipe
source_filter, sink_filter: Filter ; source_port, sink_port: PORT
source_data,sink_data: seq DATA
alphabets: P DATA
source_port∈source_filter.out_ports ∧ sink_port ∈ sink_filter.in_ports
source_filter.alphabets(source_port)=alphabets
sink_filter.alphabets(sink_port)=alphabets
ran(source_data)□p.alphabets∧ ran(sink_data)⊆ p.alphabets
}
schema PipeCompute{
△ Pipe
p′=p
□deliver:seq DATA | #deliver>0
●source_data=deliver~source_data′∧ sink_data′=sink_data~deliver
}

The Formalization of Pipes-filters

The pipes-filters system is composed of a filter set ("Filter") and a pipeline set ("Pipe") where the filter is used as a component, the pipeline is used as a connection, usually one pipeline connects two filters. Each filter and each pipeline has a unique identifier, each are not identical. The filter is an independent entity and not affected by each other. The input port of the pipeline is the output port of the first filter, the output port of the pipeline is the input port of the second filters. The data after the first filter processing enters into the pipeline through the source port of pipeline, flowing out from the pool port of pipeline and finally going into the second filter. The order and contents of data in this process still remains the same. So we can use "System" to define pipeline in the way of formalization, and its expression form is as follows:

schema System{
filters: P Filter; pipes: P Pipe
∀f1,f2:filters | f1≠f2 ●(f1.in_ports∪f1.out_ports)∩(f2.in_ports ∪f2.out_ports)=∉
∀p1,p2:pipes | p1≠p2
●(p1.source_port∪p1.sink_port)∩(p2.source_port ∪p2.sink_port)=∉
∀p:pipes ●∃ f1,f2:filters
●p.source_port∈f1.out_ports ∧p.sink_port∈f2.in_ports
∧f1.alphabets(p.source_port)=p.alphabets ∧f2.alphabets(p.sink_port)=p.alphabets
}

Conclusions

The formal methods based on the rigorous mathematical foundation provides a strict and effective way for the design and analysis of model. It can find inconsistencies, ambiguities and

errors in the earlier, and effectively reduce design errors and improve the feasibility of the system. The paper uses the formal methods to describe the pipe-filter software architecture and further explains how to use formal language to describe software architecture. It can be seen that formal description can give a precise mathematical model, and the implementation of the algorithm can be described in detail, and the design of software architecture is no longer a non-formal text and graphics.

Acknowledgment

In this paper, the research was sponsored by the Guangdong Natural Science Foundation under grant (Project No. 2015A030313490), all support is gratefully acknowledged.

References

[1] Decheng Miao, Libo Feng. Study on the application of formal methods in software engineering [J]. Journal of Hebei University Of Science and Technology, 2011,32 (6): 575~579.

[2] Tianlong Gu. The formal method of software development [M]. Higher Education Press, 2005

[3] Guangyi Guo. The formalization of Z language and software architecture style [J]. Computer Technology and Development, 2009,19 (5): 140~142.

[4] Zheng Qin. Software architecture (Third Edition) [M]. Tsinghua University Press, 2015

[5] Yuanyuan Xiao. Application Research on software architecture of pipe-filter [J]. Public Science and Technology, 2010 (11): 21~22.

[6] Chong Feng. Software architecture theory and practice [M]. People's Posts and Telecommunications Publishing House, 2004.