Z语言与软件体系结构风格的形式化

郭广义,李代平,梅小虎
(广东工业大学 计算机学院,广东 广州 510075)

摘 要:软件体系结构风格是软件设计人员在长期开发某种类型软件经验的基础上总结出来的适合于构建某一类软件的模型,也称为构建模式。形式化则是一种基于数学的严谨的描述方式和方法。形式化不仅能够清晰地描述软件体系结构风格,并且为软件体系结构的设计提供了一种易于交流和理解的途径,因此形式化是现在软件体系结构研究的主要课题之一。文中通过Z语言描述管道-过滤器这一软件体系结构风格静态性质和动态行为来说明如何运用Z语言形式化的描述软件体系结构风格,从中可以看出Z语言的严谨、清晰、简洁。

关键词:软件体系结构;管道-过滤器;Z语言;形式化
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2009)05-0140-03

Z Language and Formalization of Software Architecture Style

GUO Guang-yi, LI Dai-ping, MEI Xiao-hu
(Computer Academy of Guangdong University of Technology, Guangzhou 510075, China)

Abstract: The style of software architecture is the designers of software who sum up from their experiences. It is fit for one kind of model of software, in the other word, also all construction mode. Formalization is one kind of precise description way which is based on math. The style of software architecture can be described clearly by formal methods. Formal method makes the design of software architectures easily to understand and commutation. Formalization is one main study topic of software architectures. Describes the static character and dynamic action of piles-filters style through Z language to illustrate how to describe the formalization of software architecture style use Z language. And this way is precise, clear and straightforward.

Key words: software architecture; pipe-filters; Z language; formalization

0 引 言

软件体系结构分为两大组成部分[1]:组件和连接件。在管道-过滤器风格中,过滤器为组件,用来转换和处理数据流。每个过滤器有一组输入端口用于读入数据,一组输出端口用于输出已处理的数据。而管道为连接件,用于控制系统的数据流,每个管道有一个输入端口和输出端口,用于实现数据传输。通过管道连接的过滤器序列称为处理流水线或管线,如图1中矩形表示过滤器,带箭头的直线为管道。

形式化[2~4]指的是用严格的数学语言将抽象的概念或思想描述成具体的数学公式或数学模型,软件风格的形式化,即用形式化方法将描述软件的构建模式的非形式文本或说明图转化为严格用数学言描述的数学模型[5]。文中将用形式化语言Z语言描述管道-过滤器这一软件体系结构风格[6~8]。Z语言是应用广泛的形式化规格语言。Z语言描述包括数据抽象[9]和过程抽象[10],即分别描述系统的静态性质和动态行为。

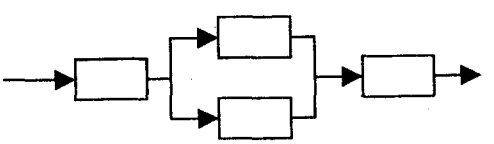

图1 管道-过滤器结构

1 数据抽象

数据抽象又称为表示抽象,是利用抽象的数据结构描述系统的功能,且不关心这些抽象数据类型的细节及其在计算机的表示和实现,在表示抽象中,数据从数据结构的表示细节中抽象出来,使用关系[11]、函数、集合、序列、包等抽象的数据类型,形式化Z语言则运用模式来描述表示数据的抽象。模式有唯一的名字,起标识作用,还包括一个声明部分和一个断言部分[12]。声明部分引入了某些类型的变量,这些变量为

收稿日期:2008-08-03
基金项目:广西自然科学基金项目(0229009)
作者简介:郭广义(1984-),男,福建龙海人,硕士研究生,研究方向为智能卡操作系统;李代平,博士,教授,研究方向为软件体系结构、并行计算、智能卡操作系统。

该模式内的局部变量,这些局部变量用来表示数据最基本的数据元素;断言部分描述了这些局部变量的关系[13],这些关系同时约束数据之间的交互操作。以下将给出管道－过滤器结构形式化过程中最基本的构成单元[14],即:过滤器模式、管道模式、管线模式。这几种模式是组成管道－过滤器结构这一软件体系结构风格的基本元素。

1.1　过滤器模式

过滤器是管道－过滤器结构最重要的组成部分,是数据处理的基本单元。过滤器的工作是接受数据、处理数据、发送数据。以下形式化描述过滤器,引入如下类型:FILTER 、PORT 、DATA ,分别表示过滤器集合、端口集合、数据集。过滤器包括过滤器名字标识、一组输入端口、一组输出端口、端口映射函数、处理函数, 建立 Filter 模式:

Filter:
[filter_id:FILER
in,out:PORT
in_ports,out_ports:P PORT
alphabet:PORT ↦seq DATA
transition: alphabet(in)↦alphabet(out)
|in∈in_ports∧out∈out_ports∨
in_ports∩out_ports= ∅ ∨
in_ports∪out_ports=dom alphabet∨
∀in_port: PORT·in_port∈in_ports∧
dom transition(in_port)=alphabet(in_port)∨
∀out_port: PORT; ∃in_port:PORT
·out_port∈out_ports∧in_port∈in_ports∧
ran transition(in_port) =alphabet(out_port)]

Filter 模式中声明了过滤器由过滤器名字标识、一组输入端口、一组输出端口、处理函数、端口映射组成。断言部分指出输入数据与输出数据并不相等,即输入要经过处理才能转化成输出;transitions 处理函数的定义域和值域分别为输入数据和输出数据;输入端口和输出端口产生的数据都属于端口通过端口映射函数 alphabet 产生的数据的集合。

1.2　管道模式

管道在管道－过滤器结构除了起连接作用外,还起了缓冲数据并同步数据的作用,可以看做一个数据队列。管道包括一个输入端口和一个输出端口,还有用于实现缓冲数据、数据同步的数据序列,建立 Pipe 模式:

Pipe:
[souce_filter,sink_filter:FILTET
sorce_port,sink_port:PORT
queue:seq DATA |
sorce_port∈souce_filter∧
sink_por∈sink_filter∨
souce_filter.alphabet(source_port)
=last queue∧
sink_filter.alphabet(sink_port)=head queue]

Pipe 模式中声明了管道由一个输入端口、一个输出端口和一个队列组成,断言部分指出,管道输入端口与过滤器输出端口相连接;管道输出端口与过滤器输入端口相连接;输入的数据进入队尾,输出的数据取自队头。

1.3　管线模式

将过滤器和管道相互连接,就组成管线;管线可以算是最简单的管道－过滤器结构,因此再复杂的管道－过滤器结构都可以由管线组成。跟过滤器和管道一样,管线的形式化描述即管线模式,将 filter 模式和 pipe 模式连接成 pipeline 模式:

pipeline:
[filters:P Filter
pipes: P Pipe|
∀c_1,c_2:filters·c_1.filters_id=c_2.filters_id
⇔c_1=c_2∨
∀p: pipes·p.soruce_filter∈filters
∧p.sink_filter∈filters∨
∀f: Filters; pt: PORT·pt∈f.in_ports∧
#{p: pipes|f=p.sink_fitler∧pt=p.sinkport}
⩽1∨
∀f: filters; pt: PORT·pt∈f.out_port∧
#{p:pipes|f=p.source_fitler∧pt=
p.source_port}⩽1]

pipeline 模式中声明了管线由若干个过滤器和管道组成,而在断言部分指出,每个过滤器必须有唯一的标识[15];过滤器的输出端口只能与管道的输入端口相连接,且端口间的连接是一对一的,即不允许有多个过滤器的输出端口与一个管道的输出端口相连接或多个管道的输入端口与一个过滤器的输出端口相连接,同时管道的输出端口和过滤器输入端口同样有一对一的关系。

2　过程抽象

过程抽象又称为操作抽象,用于描述在数据抽象所引入的数据上的抽象算法和操作[16]。过程抽象将给出算法具体的操作过程和步骤及其各个操作间的关系。Z 语言同样运用模式来描述操作抽象,管道和过滤器系统的操作由过滤器操作[17]和管道操作组成,以下在表示抽象的基础上,描述基于过滤器模式的过滤器操作和基于管道模式的管道操作,并在这两种操作

基础上定义管道－过滤器系统[18]的操作即管线操作。

2.1 过滤器操作模式

过滤器操作过程包括从管道读入数据[19]，处理数据，将数据写入管道：

读入数据，即过滤器从管道读入数据，因为管道有缓冲和同步数据的作用，所以过滤器读入数据就是将数据从管道的数据队列读到过滤器的数据池内，定义读入数据模式如下：

filter_in:
[f:Filter,p:Pipe, pt:PORT|
pt∈f.in_ports∧
#{p: pipes|f=p.sink_fitler∧pt=p.sinkport}
≤1∨
f.alphaber(pt)=head p.queue]

处理数据，即过滤器读入数据后在过滤器内部执行一系列的算法对其数据的操作处理过程，处理数据的形式化描述处理数据模式描述如下：

filter_transit
[f:Filter,in,out:PORT|
in∈f.in_ports∧out∈f.out_ports
f.alphabet(out)=
f.tansition(f.alphabet(in), f.alphabet(out))]

数据输出，当数据处理完后，可以通过管道输出到其它的过滤器继续后续的处理，或输出到数据池，所以与输入数据一样，数据输出同样包含了将数据输出到管道对列的操作，数据输出的形式化描述数据输出模式如下：

filter_out:
[f:Filter,p:Pipe,pt:PORT|
pt∈f.out_ports∧
#{p: pipes|f=p.source_fitler
∧pt=p.source_port}≤1∨
last p.queue= f.alphaber(pt)]

以上分别形式化描述了读入数据模式、处理数据模式、输出数据模式。过滤器的操作也就是这三个操作的联合，因此过滤器操作模式 filter_operate 可以定义如下：

filter_operate≙{filter_in∧filter_transit
∧filter_out}

2.2 管道操作模式

管道操作，包括数据流入和流出管道，其实管道操作中的数据流入管道，和数据流出管道跟过滤器的数据输出和数据输入是一样的，都包含了对管道数据队列的读出和写入，即数据流入管道模式 pipe_in 和数据流出管道模式 pipe_out，因此可以定义如下：

pipe_in≙filter_out
pipe_out≙filter_in

由数据流入管道模式 pipe_in 和数据流出管道模式 pipe_out 可以定义管道的操作模式 pipe_operater 如下：

pipe_operate≙{pipe_in∧pipe_out}

2.3 管线操作模式

管线的操作，可以看作管道最基本的操作，实际上管线操作就是过滤器从管道读入数据，处理数据之后将数据输出到管道的整个过程，因此有过滤器操作模式和管道操作模式，可以定义管线的操作模式如下：

pipe_line_operate≙{filter_operate∧
pipe_operate}

既然管线操作是管道的最基本操作，那么管道操作也就是管线的一次或多次重复而已[20,21]，即管道操作 filter_pipe_operate 可以定义如下：

filter_pipe_operate≙seq{pipe_line_operate}

3 结束语

文中运用 Z 语言，给出了从管道－过滤器结构详尽的形式化描述的实例。包括过程抽象到操作抽象，过程抽象包括管道模式、过滤器模式、管线模式，用于描述管道－过滤器这一软件体系结构风格的静态性质，而操作抽象包括管道操作模式、过滤器操作模式、管线操作模式则详细地描述了管道－过滤器动态行为。由静态到动态可以很清晰地说明管道－过滤器结构的具体性质和操作过程。可以看出形式化描述可以给出严谨精确的数学模型，而且能够详细地描述算法的执行过程，通过形式化，软件体系结构的设计不再是非形式的文本和图形。

参考文献：

[1] Shaw M, Garland D. Software and Architecture[M]. [s.l.]: Peason Education,2006.
[2] 王一宾，刘 奎，汪 洋. 软件体系结构研究与实践[J]. 计算机技术与发展，2007,17(9):142－145.
[3] 古天龙. 软件结构的形式化方法[M]. 北京：高等教育出版社，2005.
[4] 刘 真. 软件体系结构[M]. 北京：中国电力出版社，2004.
[5] 牛振东，江 鹏，金福生. 软件体系结构[M]. 北京：清华大学出版社，2007.
[6] 张友生. 软件体系结构[M]. 第 2 版. 北京：清华大学出版社，2006.
[7] 孙 昌，金茂忠. 软件体系结构描述研究进展[J]. 计算机科学，2003,30(2):136－139.
[8] 孙志勇，刘宗田，袁兆山. 软件体系结构描述语言 ADL 及

（下转第 146 页）

3 性能测试

仿照参考文献[5]中的测试，使用了两个进程来运行：一个是实时进程，还有一个是重负荷的后台进程。实时进程主要做了以下的任务：读取当前时间 t_1；睡眠 T 时间；再次读当前进程 t_2。并且重复此动作若干次。理想的情况下，t_2 差不多等于 $t_1 + T$，此时，延迟为 0。选取了在前面所更改过的代码作为后台重负荷进程。表 1 是测试的最终数据表。

表 1 调度延迟时间比较

	原始内核	加了延迟加锁补丁的内核	加了改进的延迟加锁补丁的内核
最小	22μs	20μs	19μs
最大	820μs	191μs	150μs
平均	191.23μs	59.67μs	40.3μs

从表中看以看出，加了改进代码的延迟加锁机制较原来的机制，在实时性能上，是有所提高的。特别是最大延迟时间提高不少平均调度延迟时间较为改进的延迟加锁技术也减少了 30% 左右。这主要归功于前面所述的锁分解机制。锁分解机制，不管是单独使用，还是用以改进延迟加锁技术，对于实时性能的提高，都是一种不错的方法。

4 结束语

文中讨论了 Linux 提高实时性能的一种简单而有效的方法：延迟加锁技术，并对其不足进行了研究，提出了一些改进的方法。通过引入锁分解技术，使得原先的实时进程最小周期的限制得以一定程度上的缓解；通过扩大内核中所授权管理的锁的范围，使得实时性能在某些程度上得以进一步的提高；通过修改原先对于软中断及下半部机制的处理方式，使得在不影响实时性提高的情况下，减少内核的更改量。实验表明，这些改进是切实而有效的。

参考文献：

[1] 余 兵，黎忠文. Linux 操作系统实时性分析[J]. 计算机技术与发展，2007，17(9)：41－44.

[2] 周 鹏，周明天. Linux 内核中一种高精度定时器的设计与实现[J]. 计算机技术与发展，2006，16(4)：73－75.

[3] 赵慧斌，李小群，孙玉芳. 非独占锁的优先级继承协议及其在 Linux 下的实现[J]. 电子学报，2003，31(8)：1145－1149.

[4] Lee Jupyung，Park Kyu－Ho. Delayed Locking Technique for Improving Real－Time Performance of Embedded Linux by Prediction of Timer Interrupt[C]//Proceeding ofthe 11th IEEE Real Time and Embedded Technology and Applications Symposium. USA：[s. n.]，2005：1080－1012.

[5] Yang Jian，Chen Yu，Wang Huayong，et al. A Linux Kernel with Fixed Interrrupt Latency for Embedded Real－Time System[C]//Proceedings of the IEEE Second International Conference on Embedded Software and System(ICESS'05). USA：[s. n.]，2005.

[6] 赵慧斌，李小群，孙玉芳. 改善 Linux 核心可抢占性方法的研究与实现[J]. 计算机学报，2004，27(2)：244－251.

[7] Love R. Interactive Kernel Performance[C]//Linux Symposium. Ottawa，Canada：[s. n.]，2003.

[8] 洪雪玉，张 凌. Linux 实时性研究及其中断进程化的实现[J]. 计算机工程，2007，33(10)：64－69.

(上接第 142 页)

其研究进展[J]. 计算机科学，2000，27(1)：36－39.

[9] 赵会群，孙 晶，王国远. 软件体系结构性能评价研究[J]. 计算机科学，2003，30 (2)：144－146.

[10] 戎 玫，张广泉. 软件体系结构求精方法研究[J]. 计算机科学，2003，30(4)：108－110.

[11] 韦 群，熊 璋，赵 芳，软件体系结构开发方法及其应用[J]. 计算机工程与设计，2003，24(3)：77－80.

[12] 张凤荔，杜小丹. 软件体系结构在软件生存期的应用研[J]. 微型机与应用，1999(10)：4－5.

[13] 赵会群，孙 晶，王国仁，等. 软件体系结构：一个新的研究领域[J]. 计算机科学，2002，29(11)：146－149.

[14] 童云卫，郝克刚. 一种面向方面的软件体系结构[J]. 微机发展(现名为：计算机技术与发展)，2004，14(6)：61－63.

[15] 张友生，陈松乔. 正交软件体系结构的设计与进化[J]. 小型微型计算机系统，2004，25(2)：295－299.

[16] 孙昌爱，金茂忠，刘 超. 软件体系结构研究综述[J]. 软件学报，2002，13(7)：1228－1237.

[17] 梅 宏，申峻嵘. 软件体系结构研究进展[J]. 软件学报，2006，17(6)：1257－1275.

[18] 王学奇，陈华勇，肖明清. 开放性测试软件体系结构研究[J]. 微计算机信息，2005，10(3)：145－147.

[19] 周 欣，黄 璜，孙家啸，等. 软件体系结构质量评价概述[J]. 计算机科学，2003，30(1)：49－52.

[20] 高 晖，张 莉. 软件体系结构层次的结构度量研究[J]. 计算机工程与应用，2007(24)：19－23.

[21] 李龙澍. 软件体系结构风格综述[J]. 安庆师范学院学报：自然科学版，2006，12(4)：1－4.