软件体系结构求精方法研究*)

戎 玫[1] 张广泉[2,3]

(暨南大学中旅学院计算机中心 深圳518053)[1] (苏州大学计算机科学与技术学院 苏州215006)[2]

(中国科学院软件所计算机科学重点实验室 北京100080)[3]

Refinement Methods for Software Architecture

RONG Mei[1] ZHANG Guang-Quan[2,3]

(Computing Center of Tourism College, Jinan University, Shenzhen 518053 China)[1]

(College of Computer Science and Technology, Suzhou University, Suzhou 215006)[2]

(State Key Lab. of Computer Science, The Chinese Academy of Sciences, Beijing 100080, China)[3]

Abstract Software architecture is the highest-level abstraction of a system. It provides a model of the large-scale structural properties of systems. Recently, software architecture has been an important research sub-field of software engineering. In this paper, our research work focus on refinement methods of software architecture. By studying the several refinement methods for software architectural design, we present a component-based refinement method that refines an architectural design by replacing a component's static semantics to its dynamic semantics.

Keywords Software architecture, Component-based, Refinement, XYZ/E, Temporal logic language

1. 引言

软件设计在相当大的程度上可与建筑设计相类比，在古今中外建筑设计中，有诸多如欧洲的"歌特式"、"巴洛克式"、"维多利亚式"，中国的"园林式"、"宫廷式"等不同结构风格的建筑。同样在软件设计上，经过多年的理论探索和工程实践，也逐渐形成了一系列不同结构风格的软件体系结构。如UNIX 操作系统中的管道-过滤器(pipe-filters)风格、分布式系统中典型的客户机/服务器(client/server)风格以及通信系统中的分层(layer)系统等等[1]。

软件体系结构是20世纪90年代软件工程领域出现的一个研究热点。与传统软件技术研究的不同之处在于，软件体系结构把原来从不同角度和不同观点出发建立的分散、非系统、非形式化的关于软件构成的思想经过研究分析和归纳综合，建立统一可用于软件系统设计、分析、构造、实现的符号和概念体系[2]。根据1999年 IEEE AWG/P1471的定义，所谓软件体系结构，它是软件系统的高层抽象，描述整个系统的结构和行为模型，标识了主要的系统组件(component)、组件之间的交互——连接件(connector)，以及组件和连接件如何结合在一起的约束与配置关系。

研究软件体系结构的首要问题是如何表示和描述体系结构。目前许多体系结构描述方法往往是非形式化(如图表法)和半形式化的(如模块连接语言 MIL、软构件描述法等)，不利于对体系结构进一步分析和开发；为解决这一问题，我们在文[3]中提出一种基于可执行时序逻辑语言 XYZ/E 的软件体系结构形式化描述方法，采用 XYZ/E 描述了体系结构的组件、连接件等基本元素以及几种重要的体系结构风格。为了更好地支持基于体系结构的软件分析和开发，目前体系结构描述语言(ADL)已成为软件体系结构研究的核心问题[4,5]。ADL 通常以一形式化语言作为它的底层语义模型，并为软件系统的概念体系结构建模提供具体语法和概念框架，还有一系列基于底层语义的工具支持体系结构的表示、分析、演化和设计；ADL 不但是形式化描述软件体系结构的基本工具，而且也是对软件体系结构进行求精、验证、演化和分析的前提和基础；目前常见的 ADL 有 Wright、Rapide、Unicon、Aesop、Darwin、SADL…等。

对于规模和复杂性均不断增大的大型软件系统，在对软件体系结构进行抽象描述之后，一个十分重要的工作就是如何将其分层细化、逐层求精为具体的可实现的软件体系结构，即所谓的体系结构求精(refinement)。这里逐层求精是指上层与下层之间存在一种形式上的抽象映射，低层通过这种映射应保留高层的属性。

2. 软件体系结构求精方法概述

软件体系结构的求精与早期的程序逐步求精不仅在实现细节的层次级别上不同，而且两者在实现方法上前者也要比后者复杂得多，这是因为抽象的软件体系结构(风格)通常是简单易理解的，而具体的软件体系结构因为要考虑实现的问题而很复杂，因而在不同抽象层次的软件体系结构之间的映射通常没有明确的规定，即使有，一般也是不完备的，因此很难检查相邻两层体系结构之间求精过程的语义一致性，也难以对体系结构的性质(如连接的安全性、公平性等)进行分析和验证。目前对软件体系结构求精方法的研究尚处在起步和探索阶段。

2.1 基于行为替代的体系结构求精

行为替代是传统的逐步求精方法。Stanford 大学研制开发的软件体系结构语言 Rapide 即采用此方法对体系结构进行求精：如果能找到一个有效的从系统 X2到 X1的抽象映射，则 X2可以代替 X1。这种方法需要对每个具体的系统的每一步求精定义抽象映射再验证一致性。

*)本研究得到国家自然科学基金(No:60073020)、重庆市应用基础研究项目、中国科学院计算机科学重点实验室开放课题基金资助。戎 玫 博士，主要研究领域为电子商务、软件工程。张广泉 博士后，主要研究领域为软件体系结构、形式化方法。

行为替代方法的特点是下层的具体表示不应产生上层抽象表示不能表示的任何行为和性质，即求精不能破坏系统的基本结构。例如：如果在抽象软件体系结构中两个组件之间没有连接件，则就断定它们之间不交互，因而求精时就不能引入连接件使之交互。

由于在实际求精过程中，有时用户并不会把一些细节和性质（如无死锁性、活性等）描述出来，而是把它们放到更低的实现层中，因此行为替代方法有时并不太确切。

针对上述不足之处，文[6]提出"求精模式（refinement pattern）"的方法。即每步求精采用已定义的求精模式，用模式定义中的具体模式代替抽象模式。如何找出所有的求精模式是该方法的关键。

2.2 基于风格的体系结构求精

该方法由 D. Garlan 于20世纪90年代中期提出，即在不同的体系结构之间定义一组求精规则，若两个不同的体系结构（风格）满足这些规则，则认为其中一个是另一个的求精。这样可以用其中一个风格的实例代替另一个风格的实例。

基于风格的求精方法的优点是，在风格这一层次的分析比实例层的分析更强大，因为在风格上所做的一些分析可以为同一风格的多个实例复用，避免了一些复杂的细节，如公式证明等；但在另一方面，该方法也有很大的局限性，如不能运用一些与具体实例有关的特性来分析系统的性质。为解决这个矛盾，D. Garlan 在1996年又提出一种折衷的方法[7]，把一种具体的体系结构风格分成若干个子风格（substyle），由于子风格更接近具体的实例，可以为它们建立一些更为特殊的求精规则，这样做就不需一个风格的所有实例都满足同样的规则，因而降低了求精规则的难度。

如何确定子风格和相应的约束条件以及根据这些约束条件定义求精规则是该方法的关键。例如：对于一个管道-过滤器风格，可以将"管道线（pipeline）"定义为其子风格，对于这个子风格，可以定义一些简单的求精规则。图1通过两个函数来刻画这种方法，其中第一个函数确定给定系统（stylel）是否是一个特定子风格（style la）的成员；第二个函数确定抽象（子）风格和具体风格成员（style2）之间的求精关系。

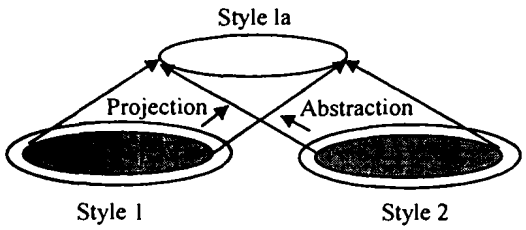

图1 子风格和求精

3. 基于组件的体系结构求精方法

该方法从体系结构的基本元素——组件入手，每次求精不是针对体系结构（风格），而是对组件进行求精。首先将系统作为一个抽象的组件，根据功能需求描述组件的规范（即静态语义），然后，选择合适的体系结构风格，对抽象组件进行分解，生成若干组件以及它们之间的交互，对这一层包含的组件作进一步的分解，形成第二层的设计，就这样一层一层分解下去，直到最后不可分解为止。

在该方法中，组件由两部分组成：外部界面与内部结构（见图2）。外部界面用组件的规范来表示，它刻画了组件"做什么"；内部结构也就是实现这个组件的逻辑功能的体系结构，它包括下层的组件以及它们之间的交互，它刻画了组件"怎么样"。由前者向后者转换，即构成软件开发过程的一步过渡，这样的逐层过渡就构成了软件开发的逐步求精过程。

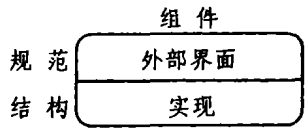

图2 组件的构成

基于组件的求精方法较适合于具有分层嵌套结构组件的软件系统。下面我们通过一个实例，进一步说明如何应用此方法。

SZRTOS 实时操作系统是航天部771所开发的一个为航天应用程序提供底层服务支持的实时操作系统平台。系统最上层是应用程序；第二层是系统为应用程序提供的接口，包括系统调用、I/O 驱动模块和容错管理模块；第三层由中断处理、进程管理、进程通信、同步互斥等功能模块组成；最底层是硬件；其中第二层和第三层就是 SZRTOS 实时操作系统内核。从体系结构风格的角度，SZRTOS 实时操作系统内核可看作是一个分层系统结构（见图3）。

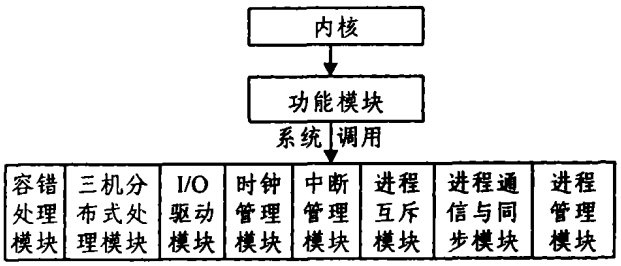

图3 SZRTOS 实时操作系统内核结构示意图

文[8]基于 XYZ/E 语言描述了 SZRTOS 实时操作系统内核的规范和具体实现算法，并在时序逻辑的框架内证明其设计的正确性。本文从软件体系结构的角度出发，简要分析一下其求精步骤和设计过程：

（1）分析内核设计需求，抽取内核规范

由于内核是一个开放系统，需要与外界环境进行通信，因此内核的数据结构应用包含两个变量分别存储从环境接收的事件信息和向环境返回的事件处理结果；内核的主要功能为向应用任务提供服务和管理硬件，其中，应用任务由一组应用进程组成，进程在内核中用进程控制块来表示，而硬件与内核需要通过端口来交互，因此内核规范的数据结构也应该存在变量刻画出内核的进程控制块和端口信息；此外，内核数据结构中还应该包含记录系统时间和当前运行进程 ID 号的变量。

SZRTOS 操作系统内核处理的变量定义如下：变量 event 和 result 分别记录内核最近一次从环境接收到的事件消息和向环境发送的事件处理结果；变量 runProsID 和 systime 分别记录系统当前运行进程标志 ID 号和系统启动后总共运行时间；数组变量 ports 记录系统中各个输入输出端口存储的数值；数组变量 prosArray 记录进程控制块信息，包括各个进程的优先级、运行状态、对信箱和互斥临界区资源的访问关系等信息。变量 event、result 和 prosArray 分别对应的 InputEventType、OutputEventType 和 AProsStruct 三种结构数据类型，其具体定义详见文[8]。

我们将 SZRTOS 实时操作系统内核看作一个抽象的组

件，根据功能需求给出它的规范描述；然后简单分析一下操作系统内核组件的内部结构，划分出相应的子组件(模块)。

SZRTOS 操作系统内核的规范用自然语言可描述为：如果操作系统内核启动时的输入参数 initProsID 和 initProsPriority 满足关系 InitConstraint，且内核与环境交互的交互过程中一直满足时序逻辑式□InputConstraint，则操作系统内核在运行时必须满足系统调用性质□SystemCallSpecification、安全性性质□ScheduleProperty、时间管理模块性质□TimeManagerProperty。其中刻画内核安全性性质的时序逻辑式为□ScheduleProperty，逻辑式 ScheduleProperty 定义如下：

```
ScheduleProperty=def chout!→RunProsIDProperty
RunProsIDProperty=def∀ id∈1..MAX_PROS:
ProsArray[id].status=PROS_READY
→runProsID≠0 ∧ RunProsPriority(runProsID)≤RunProsPriority
(id)
```

其直观含义就是内核每次处理完接收到的系统调用或中断事件后向环境输出处理结果时，或者当前没有进程在运行，或者当前所有处于就绪态的进程的运行优先级不高于当前运行进程的运行优先级。

内核规范可用如下 XYZ/E 语言形式化描述，这里内核对应于一个 XYZ/E 语言的进程元素。对于逐步求精过程中的各个抽象程序，我们在 Where 部分中给出了它们必须满足的性质。

```
{InitConstraint ∧ □InputConstraint}
Kernel(%CHN/chin(env:NM,*):InputEventTyp;%CHN/chout
(*,env:NM):OutputResultType;
%INP/initProsID:int;%INP/initProsPriority:int)
Where[
□SystemCallSpecification ∧ □ScheduleProperty ∧ □TimeManagerP-
roperty
]
```

SZRTOS 实时操作系统内核的结构从功能角度可划分为八个模块(见图3)：进程管理模块、进程通信与同步模块、进程互斥模块、中断管理模块、时钟管理模块、I/O 驱动模块、三机分布式处理模块和容错处理模块。各个模块的功能描述详见文[8]。

(2)系统调用的规范描述

在 SZRTOS 实时操作系统中，内核和应用程序之间的交互是通过系统调用实现的。这里系统调用可看作是连接件。内核中一共存在16个系统调用，文[8]详细描述了它们的规范。内核处理应用程序(进程)发出的系统调用的描述是对各个系统调用描述的交，可以用一时序逻辑式□SystemCallSpecification 来表示，其中逻辑式 SystemCallSpecification 定义如下：

```
SystemCallSpecification=def
chin?event ∧ (#akernel=akernel) ∧
(event.eventtype=SYSTEMCALL ∧ event.runProsID=runProsID)
→
∧i∈1..16∧j∈0..c(i):((event.SystemCallNo=i)→
(#SystemCall_Prei,j→SystemCall_Posti,j))Δ(chout!result)
```

(3)基于组件的逐层过渡及语义一致性验证

当 SZRTOS 实时操作系统内核规范给出后，就可以对其进行逐步分解，并可验证每一次过渡的语义一致性问题。

如首先我们将在规范数据结构上定义的 XYZ/E 程序 Kernel 求精到在实现代码数据结构上定义的 XYZ/E 程序 Kernell。在 Kernell 中，我们除了将映射关系 Imp 作用于定义在规范数据结构中的逻辑式外，还引入了在前面给出的实现代码之间的冗余关系 Var_Relation。求精后得到的 XYZ/E 程序如下：

```
{InitConstraint ∧ □InputConstraint}
Kernel1(%CHN/chin(env:NM,*):InputEventType;%CHN/chout
(*,env:NM):OutputResultType;
%INP/initProsID:int;%INP/initProsPriority:int)
Where[
  Imp·(□SystemCallSpecification ∧ □TimeManagerProperty) ∧
□(chout!→Var_Relation)
]
```

XYZ/E 程序 Kernel1是程序 Kernel 的求精，它们之间的语义一致性证明如下：

证明：

```
Var_Relation→Imp·RunProsIDProperty
→□(chout!→Var_Relation)→Imp·ScheduleProperty
/*ScheduleProperty≡chout!→RunProsIDProperty*/
→Imp·(□SystemCallSpecification ∧ □TimeManagerproperty) ∧
  □(chout!→Var_Relation)
→Imp·(□SystemCallSpecification ∧ □TimeManagerProperty ∧
  □ScheduleProperty)
→LFKernel1→LFKernel /*由程序语义定义得到*/
```

证毕。

其余求精过程及其语义一致性验证详见文[8]。

在上例中，我们采用可执行时序逻辑语言 XYZ/E 描述 SZRTOS 实时操作系统内核的体系结构，XYZ/E 的基本特征是可以在统一的形式框架下表示从形式规范到可执行程序的不同抽象层次的系统描述，即既能表示程序的动态语义，又有表示规范的静态语义。XYZ/E 的这一重要特色在一定程度上弥补了现有许多 ADL 存在的不足，它不仅适于对软件体系结构进行精确描述，还利于对不同抽象级体系结构进行求精，在统一的时序逻辑框架下，便于论证求精过程的一致性。

参考文献

1 Shaw M, Garlan D. Software Architecture: perspectives on an emerging discipline. America: Prentic Hall, 1996

2 万建成，卢雷. 软件体系结构的原理、组成与应用. 北京：科学出版社，2002. 8

3 Rong M, Zhang G Q. A Software Architecture Description Approach Based on XYZ/E. In: Proc. of 7th Inter. Symp. on Future Soft. Tech. ISFTS-2002, 2002. 10

4 张广泉，唐稚松. 一种新的软件体系结构描述语言. 见：中国博士后学术大会论文集，北京：科学出版社，2001. 2

5 张广泉. 基于 XYZ/E 的软件体系结构描述语言研究. 计算机科学，2000, 27(9. 专辑)：155～157

6 Moriconi M, Qian X, Riemenschneider R. Correct Architecture Refinement. IEEE Tran. Soft. Eng., 1995, 21(4): 356～372

7 Garlan D. Style-Based Refinement for Software Architecture. SIGSOFT96 Workshop, San Francisco CA USA, 1996

8 唐稚松，等. 时序逻辑程序设计与软件工程(下册). 北京：科学出版社，2002. 5