软件体系结构的形式化与面向状态的形式化风格*)

胡劲松　郭荷清　郑启伦
(华南理工大学计算机学院　广州510640)

Formalizing in Software Architecture and State-Oriented Style

HU Jing-Song　GUO He-Qing　ZHENG Qi-Lun
(Department of Computer Engineering and Science, South China University of Technology, Guangzhou 510640)

Abstract The paper introduces the definition, the essential and the fundamentality of formalizing, and discusses some aspects about formalizing in detail. After reviewing several formal styles and architecture description languages, we provide the state-oriented style and the state space formal description, which is a new formal description and can combine static properties with dynamic properties. The state space formal description has many merits, such as that the expressing forms of the systems and the opponents are consistent, the expandability and the refine-ability are fine and that mathematic analysis is convenient.

Keywords Formalizing, Architecture, State-oriented

1. 引言

软件体系结构的重要性已被越来越多的软件人员所重视，体系结构从全局的、整体的角度去理解和分析整个系统的行为和特性，在更高层次上把握系统各组件之间的内在联系，有助于解决当前开发复杂的大型软件所存在的困难，保证软件质量，提高软件可靠性、可重用性和可维护性。

尽管很多优秀的软件人员在设计时使用了体系结构方法，但它们只有在特定的语言习惯中才能理解，也只能应用在一些特定专业领域中。因此设计者不能充分开发系统结构中的共性、在各种的设计方案中作出最佳决策、把普遍的范例推广到特殊的领域，其他设计者也无法分享他们的经验和知识。形式化即对软件体系结构中诸多内容进行抽象化、规范化，是体系结构设计规范化的重要基石，为上述问题提供了解决方案。

2. 形式化的内容

2.1 形式化的定义

有关体系结构的文章绝大多数都提到形式化，但是很少有文章给出形式化的定义，在经典的体系结构书中[1]也没有，这里我们给出形式化的定义。

定义1(形式化)　就是通过概括一类实体的公共属性，忽略各个实体间不同的具体、特定的细节，并按照某种规范来描述这类实体。形式化从本质上来说就是抽象，从哲学的角度来说就是从一般到特殊，并且用规范的形式和符号集来表达这一过程。

2.2 形式化模型

很多软件尽管是从体系设计开始，但不是形式化的，因为不是直接用通用的符号描述系统结构的表达抽象，过于特殊化，涉及底层的基本数据单元，形式化方法能提供精确的、抽象的模型和基于模型的技术，为描述特定的工程设计提供规范化的符号，所以可用形式化规范语言描述一特定系统的体系结构。通过运用合适的形式化模型对体系结构的非形式化描述进行规范定义，从而得到软件体系结构的形式化规范描述，以使软件体系结构的描述精确、无歧义，并进而分析软件体系结构的性质，如无死锁性、安全性、活性等。分析软件体系结构的性质有利于在系统设计时选择合适的软件体系结构，从而对软件体系结构的选择起指导作用，避免盲目选择。例如对于一个具体的系统，在获得其功能模型之后，进行如下形式化步骤：

1)对每个功能模块进行形式化。

2)对每个接口进行形式化。

3)最后，集成各部件，建立部件间输入输出间的联系，组成一个单一子系统。

形式化模型主要是揭示通过相互关联的各部件规范化来对一个系统进行体系结构化。文[1]用Z符号描述，给出了一个数字示波器的形式化模型例子。这种方法适应复杂系统形式化。

2.3 形式化风格

软件体系结构风格(style)是指不同系统所拥有的共同的结构组织模式和语义特性[2]。也即一种体系结构风格决定了部件、连接件和一组如何将它们结合在一起的约束限制，包括拓扑限制、执行语义限制。实质上，一种体系结构风格决定了一种体系结构的框架，文[3]对体系结构风格进行了分类：

2.3.1 管道—过滤器(pipes filters)　其结构具有下列特点：过滤器的设计是独立的，不受前、后部件的影响；任何两个过滤器只要能保持它们之间传输数据的一致，均可进行连接。因而该结构能较好地支持软件系统一级的重用(reuse)；易于维护和不断完善；可以分析软件的某些特性，支持并发执行。

2.3.2 面向对象组织(object oriented organization)　在这种风格中，数据表示及相关的基本操作被封装在对象中。这种风格的部件就是一种称之为"管理器"(manager)的对象。对象既有模块的封装特性，又有进程的动态信息传递特性。在

*)国家自然科学基金(69783008)，广东自然科学基金(970525)资助项目。胡劲松　博士，研究兴趣：算法分析与设计，人工智能；郭荷清　博士生导师，研究领域：软件工程，软件体系结构；郑启伦　博士生导师，研究领域：智能计算机。

串行环境下，对象之间的引用是通过函数/过程的调用实现的，其语义是静态的；在并发环境下，对象之间的交互作用是通过通讯来实现的，其语义是动态的。这种风格有两种重要特点：一个对象保证其表示的完整性；这种表示对其它对象是隐蔽的。该结构也有一些缺点，一个对象要引用其它对象，须知道这些对象的身份，一旦一个对象身份改变，则必须修改所有与之相关的对象，进而可能带来副作用问题。

2.3.3 基于事件的隐式调用(event based implicit invocation) 这种风格的一个主要优点是有利于部件重用。因为部件之间的交互是通过事件的广播来实现的，所以只需简单地注册几个它感兴趣的事件，就可把部件添加到系统中。

2.3.4 分层系统(layered systems) 其结构非常清晰，由于是分层设计，因此对系统进行改动时，只需改动相应的那一层，因而便于系统的维护。另外，由于每一层都定义了规范及与相邻层之间的接口，因此可用不同方法实现。需要指出的是，并非所有系统都能以分层方式实现，对有些系统要精确地划分其抽象层次往往是十分困难的。

2.3.5 其它典型结构 除上述已介绍的体系结构风格外，还有一些较典型的结构，如仓库(repositories)、表驱动解释器、分布式进程、主程序/子程序结构。上述列出的种种体系结构风格均属于一般的、抽象的、通用的体系结构范围。除此之外，还有更多的专门的应用领域的软件体系结构及异质体系结构等。

一种软件体系结构风格在其产生时，其思想通常是简单的，并常常由软件设计师用非形式化的自然语言表示概念、原则。例如：客户机-服务器体系结构就是为适应分布式系统的要求，从主从式演变而来的一种软件体系结构。因此该阶段的描述常是用自然语言描述的，所以各种体系结构风格虽然是抽象的，但还不是形式化的，因为大多数应用方面的文章对体系结构风格的阐述是非规范的，没有使用统一的符号集和形式来描述。

非规范的风格有不少缺点。首先体系结构类型表达不清晰，不得不从系统细节的描述中推导出来。其次高水平的抽象可避免很多重要的设计问题。第三，体系结构的连接是隐含的，所以我们不能独立于系统规范检验或推导系统的拓扑逻辑特性。

形式化的体系结构风格主要有2个优点：

1)可以很方便地分析按某种形式化风格设计的系统，包括系统的各个方面的特性。

2)形式化的风格很容易具体化、特殊化、专业化、细化、求精。例如在基本类型上增加一些新的约束，就派生了新的类型。

2.4 形式化应用结构框架

应用框架是对领域内应用系统的部分或整体的可重用设计，将应用系统的控制流进行抽象，形成更为通用的功能组件和稳定的界面。应用框架，尤其是面向对象应用框架强调模块的独立性、可用性、灵活性，可以增强大型应用软件的可扩充性和代码重用性，缩短大型应用软件系统的开发周期，提高软件质量[4]。目前还没有一个严格和精确的应用框架(framework)定义，通常这样认为：应用框架是从特定域(domain)中提取出来的一组组件及其相互关系的可重用的体系结构。应用框架作为一种软件重用技术，实际上它提供的是设计层的重用，但它经常以程序的方式出现。由于应用框架和设计样本通常以图表、自然语言等非形式化的方式出现，缺乏精确的语义，因而缺乏系统化的方法来组织、选择并结合到系统设计中，使系统设计处于手工操作。

由于具体系统的差异性和复杂性，使得软件重用限于编码层而停留在“拷贝与粘贴”阶段，而对于系统设计的重用研究甚少。形式化的方法能精确描述对象的行为和特性，是使系统设计走向工程化的重要步骤，能有效弥补非形式化的许多缺点。应用框架的研究还不成熟。尤其是没有系统化的标准和方法来选择哪些应用框架，应用框架的特性和特点不同人有不同的理解，其使用还处于无序的状态，这与应用框架的稳定性是相悖的。用形式化的方法来规范和描述应用框架，是其走向工程化的重要一步。

3. 软件体系结构标准 IEEE－1471[5]

除了抽象化，形式化的另一个重要特征是规范化，所以有必要建立一个统一的体系结构标准。尽管软件体系结构一词在计算机领域已越来越流行，但是它的使用存在很多矛盾，在软件工程中，各种概念都很不相同，在2000年，IEEE 的计算机协会批准了 IEEE－1471标准，5个核心概念和关系构筑了IEEE1471标准的基础。

1)每一个系统都有一个体系结构，但是一个体系结构并不是一个系统。

2)体系结构和体系结构本身的描述不是同一件事。

3)体系结构标准、描述及开发过程可以不同，并且可以分别发展。

4)多样性是体系结构描述本身所固有的。

5)把对象的观点概念从它的特殊性中分离出来，是建立体系结构描述标准的有效方法。

该标准主要是面向软件性比较强的系统和一般系统，如信息系统、嵌入式系统、子系统等。

IEEE 1471最重要的组成部分是：

1)一套关键词的定义集，如体系结构描述、体系结构视点、体系结构观点。

2)体系结构、体系结构描述及构造系统的标准的概念的分离。

3)满足描述一个系统体系结构的要求，体系结构描述要求包括：功能性、性能、安全性、可行性。

这不是一个强制性的、具体的标准，而是提供通用的规范的概念。

4. 形式化与重用、求精的关系

软件重用[6]是软件工程研究的重要课题，它是指在构造新的软件系统的过程中利用已有的软件成分。这里所说的已有的软件成分包括十分广泛的内容，如数据、文档、模式、体系结构，知识、方法和环境等多方面的内容。从广义上说，软件重用包括3个层次：(1)产品重用，如代码、数据、软件模式、体系结构等的重用；(2)知识重用，如方法、标准、经验、领域知识、软件过程等的重用；(3)环境重用，如系统程序、软件工具、开发环境等的重用。软件重用是提高软件生产效率、保证软件质量的关键手段之一。传统的观点认为，软件重用的对象仅为已有系统的程序代码，但是仅仅重用源程序代码等信息并不能有效地降低软件开发的成本。软件重用的对象应为整个软件开发过程、包括分析、设计、实现等各阶段中的所有信息。

抽象概念和软件重用是密不可分的，每一类抽象决定相关的可重用属性，每一可重用属性也可以决定一类抽象，即具

有这些属性的所有实体。形式化是有层次的，因为抽象是有层次的。抽象的层次性体现在对细节的取舍上，细节越多层次越低，细节越少层次越高。形式化层次的高低并没有好坏之分，而是由应用的目的决定的。形式化层次高，应用范围就广，但是细化和求精的工作量就大，形式化层次低则相反。

最有效的软件重用是在软件体系结构层的重用。软件体系结构重用是指将软件的框架组织、全局结构等作为一个整体建议重用，与软件逻辑结构相比，软件体系结构更着重于系统与各子系统，各子系统之间的相互关系，而非数据结构和算法。与应用生成器相比，均是重用系统设计，但应用生成器一般只适用于特定应用领域，隐含重用体系结构的信息，而可重用软件体系结构则通常是显式重用软件体系结构，并可以通过集成其它体系结构，建立新的更高层次的体系结构。软件体系结构的抽象直接来源于应用领域，可以用领域语言描述。从领域语言描述到实现可以全部通过自动映射来实现，开发者可通过选择特定体系结构来适应不同应用的需求。

软件体系结构的重用吸收了其他软件可重用对象的优点，是目前最理想的可重用软件对象。建立一个完备的软件体系结构库，以及用于支持、管理体系结构构件的软件开发环境，形成一种新的基于软件重用的软件开发范型。将对今后的软件开发产生重要的影响。

5. 形式化描述方法

研究软件体系结构的首要问题是如何表示软件体系结构，其主要研究对象是体系结构描述语言 ADL(Architecture Description Language)。它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，目前已成为软件体系结构的研究热点[2]。

国际软件工程界现已提出多种 ADL，它们大多是面向特定领域的，如 Aesop 支持应用软件体系结构描述风格；Adage 支持对电子导航系统体系结构框架的描述；Meta-H 为实时电子控制软件的设计者提供明确的指导；Unicon 支持异构型部件和连接，并针对体系结构有一高层编译器；Rapide 专门为系统体系结构建立快速原型设计的，这是一种基于事件的、并发的、面向对象的语言；Wright 支持定义和分析部件间的相互作用。为了精确描述体系结构，现有 ADL 通常包含一形式化语言(如 Petri 网、Z、OBJ 等)。如 Wright 采用 CSP，Rapide 采用偏序事件集合等。我们知道，这些形式化语言往往只适合描述系统行为的某些方面，如 CSP 较适合描述和分析软件体系结构的动态行为特性，如连接件在一定条件下满足无死锁性等；而 Z 较适合描述软件体系结构的某些静态性质，如风格、子风格的定义，但难以描述和分析系统的动态性质。模式互联语言(MIL)不支持重要的体系结构元素的描述，体系结构描述语言(SDL)仅支持静态的软件体系结构的描述，该语言仅支持形式化符号系统，不支持证明系统。

6. 形式化的面向状态的风格

近年来，相对于计算机硬件生产的低成本和高生产率，计算机软件生产的高成本和低生产率已严重地制约了软件的迅速发展，软件工作者从各个角度研究提高软件生产率的方法，软件重用技术就是其中之一，对应于已获得巨大成功的硬件集成电路(IC)，软件研究人员也提出了相应的软件 IC 的概念，软件 IC 是可重用的相对独立的软件部件。但是软件有其不同于硬件的特点，软件所面临的问题领域千变万化，完全的软件 IC 库的建立十分困难，目前进展不大。一个主要的原因是目前的体系结构风格难以适应软件 IC 的要求。

面向对象的风格被认为是当今最理想的软件重用技术，在这一风格中，对象的内部数据处理和过程对外部是隔离的、屏蔽的、非耦合的，和外部数据的交换通过接口参数传递进行，这一要求是比较严的，一方面可使对象容易重用，另一方面严重限制了对象模块的大型化。因为通常模块越大，和外部数据的交换越多，越难以隔离和屏蔽。打个简单的比方说：一个电阻是一个两个引脚的分立电子元件，它非常简单，使用范围广，就像对象的复用；超大规模集成电路有几百个引脚，一般使用在特定领域，使用的方法也比较复杂，但是可大大降低成本，组成的系统容易维护。由此我们想到，要组成大规模软件 IC 必须突破面向对象的风格的限制，即允许对象的内部数据处理不独立，可以和外部有耦合。由此提出面向状态的风格。

下面给出一些定义：

定义2(时序数据) 我们认为所有的数据都是有时序的，所谓时序就是数据的生存时刻、发生作用的时刻，对于寄存器、总线、网络、管道、端口中的数据是不断变化的，所以对于数据，必有时序。

定义3(组件) 所有组件，乃至整个计算机系统都可以看作时序数据处理机。

定义4(关键数据点) 组件的重要变量、端口。

定义5(状态) 组件的状态就是其关键数据点的时序数据。我们用 $X(k)$ 表示在第 k 个时刻，关键数据点 X 的数据状态。称 $X(k)$ 为状态变量。此处 X 是向量，可代表多个点。

定义6(功能的实现) 组件、模块或系统对时序数据进行变换，由初始状态经过一系列中间状态，到我们指定的某种状态。

定义7(面向状态) 以状态为核心，模块、组件就是某些状态的集合，系统是全体状态的集合。

例 如图1，A、B、C 是三个功能模块，R 是 A 的输入，它不是单个数据，而是一个数据组，同理 X_1、X_2、X_3、Y 也都是关于输入输出的数据组。现在要把 A、B 封装成一个模块，如虚线所示，显然面向对象的风格是不行的，只有使用面向状态空间的风格。下面我们使用形式化的状态空间描述法来描述。

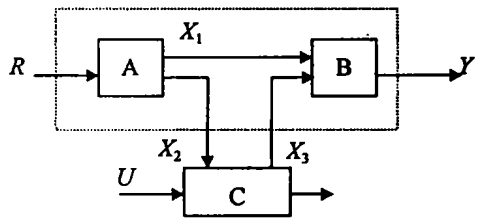

图1 模块组合

形式化的状态空间描述法：

关键数据点 X_1、X_2、X_3、Y，我们用状态空间矩阵表示 A、B 的合成模块：

$$\begin{bmatrix} X_1(k+1) \\ X_2(k+1) \\ X_3(k+1) \end{bmatrix} = \begin{bmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & a_{32} & a_{33} \end{bmatrix} \cdot \begin{bmatrix} X_1(k) \\ X_2(k) \\ X_3(k) \end{bmatrix} + \begin{bmatrix} b_{11} & 0 \\ b_{21} & 0 \\ 0 & b_{32} \end{bmatrix} \cdot \begin{bmatrix} R(k) \\ U(k) \end{bmatrix}$$

(下转第149页)

势所在。

由此可见，RISC 风格的协作构架具有很好的伸缩性，它的设计思想是通过微协议或者微构件的组合来实现系统目标。但需要指出的是，正如 RISC 不能完全代替 CISC(复杂指令集)一样，RISC 风格的协作构架也有自己的局限性。根据我们的经验，某些对实时性要求极高的系统是不适合采用 RISC 风格的体系结构的。

结论 目前我们的工作集中在可靠多播协议上。可靠多播协议有许多种类。我们主要是通过在开放式源代码中提取形式规范、并根据需要进行一定程度的修改，然后通过 PVS 验证该规范的正确性，这属于协议功能分析的范畴。实时任务关键系统所需的概率多播协议的性能分析也是非常重要的，该工作也在同步进行。任何工作都需要实际系统的验证，下一步的研究工作主要是要在实际系统验证所选用的多播协议，远期的工作目标是形成一定规模的协议库，并将其集成到通用的软件开发工具之中。

参 考 文 献

1 Floyd S, et al. A reliable multicast framework for light-weight sessions and application level framing. In: Proc. of ACM SIGCOMM [C], 1995

2 Huh S Y, Rosenberg D. A Change Management Framework: Dependency Maintenance and Change Notification. Journal of System and Software [J], 1996, 34: 231～246

3 Schmidt D C, Levine D L, Mungee S. The Design and Performance of Real-Time Object Request Brokers. Computer Communications [J], 1998, 21: 294～324

4 Thuraisingham B M, Maurer J A. Information Survivability for Evolvable and Adaptable Real-Time Command and Control Systems. IEEE Transactions ON Knowledge and Data Engineering [J], 1999, 11(1): 228～238

5 Ravindran B. Engineering Dynamic Real-Time Distributed Systems: Architecture, System Description Language, and Middleware. IEEE Transactions Software Engineering[J], 2002, 28(1): 30～57

6 Garcia-Molina H, Salem K. Main memory database systems: An Overview. IEEE Transactions on Knowledge and Engineering [J], Dec. 1992. 509～516

7 Batory D S, Leung T Y, Wise T E. Implementation Concepts for an Extensible Data Model and Data Language. ACM Transactions on Database Systems [J], 1988, 13(3): 231～262

8 Chaudhuri S, Weikum G. Rethinking Database System Architecture: Towards a Self-tuning RISC-style Database System. In: 26th Intl. Conf. on Very Large Databases[C], Cairo, Egypt, 2000

9 杨武军. 基于用户级锁的业务对象的并发控制机制:[硕士论文]. 西安电子科技大学, 2002

10 Macker J P, et al. Reliable Multicast Data Delivery for Military Networking. IEEE MILCOM'96[C]

(上接第146页)

$$Y(k)=[0 \quad c_{12} \quad c_{13}]\cdot\begin{bmatrix}X_1(k)\\X_2(k)\\X_3(k)\end{bmatrix}$$

其中，$k+1$表示下一步的状态，k 表示当前状态，$R(k)$、$U(k)$ 表示输入数据或触发事件，在系数矩阵中对角系数 a_{11}，a_{22}，a_{33}，为自相关系数或子矩阵，表示该数据点当前状态与下一状态的关系，其它系数为互相关系数或子矩阵，表示各数据点之间的相互关系。此处 a_{11}～a_{33}可以是数，也可以是带某种程序运算符的数，这种运算符可把 a 和 X 结合起来，表示某种运算或过程。关于相关系数的矩阵称之为关系矩阵。

实现：该模块的实现就是关系矩阵的细化、求精、程序化。

软件 IC：基于面向状态的风格，我们可以建立具有复杂功能组合的大型模块/组件，因为不需要考虑数据独立和屏蔽，所以状态和功能可以比较自由地组合。在上例中，假设 C 模块经常要根据顾客的要求修改，我们就把它拿出来，只组合 A、B。当模块与外界无耦合时，面向状态的风格就变成了面向对象的风格，面向对象的风格是它的一个特例。

重用：在系统设计时，使用软件 IC 的思想与类、函数等小模块是不同的，应以功能的组合为目的、以软件 IC 模块为中心，围绕这个中心配置外围程序。

系统描述：整个系统的状态空间也可以同样表示，系统与组件的不同在于其全部元素都为子矩阵。

状态空间描述是一种形式化的方法，它并不限于状态空间风格，也可以对其他风格进行形式化，它具有如下优点：

1、形式一致。系统、所有的组件其表达形式都是一致的，易于重用。

2、良好的可扩充性，可求精性。每个子矩阵都是可扩充的可细化的。

3、关系表达清楚。系统整体结构及全部组件的关系都在矩阵中表示出来，无需用复杂的图形表示，静态特性好。

4、动态特性好，因为包含了时序、事件。

5、便于数学分析。所有基于描述语言及结构图的分析都是定性的、表面的，只有基于数学模型的分析才是定量的深刻的。

6、可以利用已有的数学分析工具，如李雅普洛夫稳定原理、系统解耦。

总结 形式化是体系结构设计规范化的重要基石，可以充分开发系统结构中的共性，分析软件体系结构的性质，有利于在系统设计时选择合适的软件体系结构，从而对软件体系结构的选择起指导作用，避免盲目选择。形式化是软件重用的最有效手段。本文提出的面向状态的形式化风格可以构建大型软件 IC，并具有兼顾动态和静态特性等诸多优点。

参 考 文 献

1 Shaw M, Garlan D. Softare architecture: Perspectives on an emerging discipline. 北京：清华大学出版社, 1998

2 张广泉. 软件体系结构：概念、风格与描述语言. 重庆师范学院学报(自然科学版), 2000, 17(3): 1～5

3 Garland, Shawm. An introduction to software architecture [R]. CMU-CS-94-166, 1994

4 姜东胜，张苓，陈莘萌. 应用框架的形式化研究. 武汉大学学报(自然科学版), 2000, 46(3): 285～288

5 Maier M W, Emery D, Hilliard R. Computer, 2001, 34(4): 107～109

6 李也戈，张然. 软件重用的实现. 计算机工程, 1995, 21(4): 37～40