计算机科学2002Vol. 29№. 4

软件构件概念的变迁

The Development of Software Component Conception

马 亮 孙艳春

(北京大学计算机科学技术系 北京100871)

Abstract It has been more than thirty years since the appearing of the term software component. But there is still not a standard definition of it. Software component is the key element of software reuse. Different researchers are focused on different aspects of software reuse, which results in the different understandings about software component. This paper introduces the various viewpoints of software component conception during its history. Also this paper discusses the probable trends of future component conception.

Keywords Software component, Software reuse, Software engineering

1. 前言

软件工程作为一门独立学科出现的目标是为了解决60年代开始出现的软件危机,试图摆脱软件白手起家、手工作坊般的开发方式,提高软件生产率和质量。几十年来尽管软件工程的研究和实践取得了巨大的进展,但离人们期待的目标还有相当的距离。近年来研究人员逐渐认识到,要真正实现软件的工业化生产,达到软件产业发展所需要的软件生产率和质量,软件复用是一条现实可行的途径。软件构件(software component)则是软件复用的核心概念和基本单位。

软件构件的想法由来已久。早在1968年 NATO 软件工程会议上 McIlroy 就在题为"Mass-Produced Software Components"的论文中提出了软件构件、构件工厂等思想[1]。但 McIlroy 并没有给软件构件一个明确的定义,McIlroy 提到的软件构件概念只是从传统行业基于标准零部件组装的生产模式借鉴而来的一个想法。

70年代以及80年代初期是结构化程序设计流行的年代,子程序的复用是软件复用的主要手段。子程序概念提出的目的是为了节约机器资源,但是很快人们就认识到利用它可以提高软件开发效率。许多通用的子程序被集中到库中,几乎所有商业高级语言都带有一个庞大的子程序库。70年代还提出了模块(module)的概念,例如 MIL 语言中的模块由对外提供和需求接口以及内部功能实现组成[2]。80年代初期诞生的 Ada 语言则引入了程序包。模块和程序包都可以做为比子程序粒度更大的复用单位。80年代中后期 OOPL 兴起并逐渐成为主流编程语言,与子程序相比,OOPL 中的类具有更高的复用价值,在易用性方面也远胜于模块和程序包,类库在 GUI 领域获得了巨大的成功,大大推动了软件复用的发展。与此同时,构件的概念又重新受到研究人员的关注。几十年来随着计算机技术的不断发展,研究人员对软件构件的理解也经历了很大的变化。

2. 构件概念的发展

到目前为止还没有一个软件构件的统一定义,其根本原因在于构件被看作是复用的基本单位,而不同时期以及不同的研究人员所关注的复用类型是不同的,因此对构件的认识也就不一样。早期研究人员主要关注源程序代码的复用,将构件理解为某种可复用的程序代码片断。后来研究人员认识到复用的范围并不局限于编码阶段,而是可以贯穿于整个软件开发过程,因而对构件的概念有了很大的扩充。90年代以后随着分布对象、Web 等技术的发展和软件复用实践的深入,国际学术界对构件概念又有了新的认识。

2.1 早期的软件构件概念

7、80年代软件构件主要指可复用的程序代码片断,现在一般将它们称为代码件。这一时期如何利用已有的源程序代码、子程序库和类库来提高软件开发效率是研发人员首要考虑的问题。代码件主要包括以下几种形态:

子程序。作为构件的子程序包括程序设计语言提供的标准过程或函数,语言编译器提供的处理特定平台或硬件功能的标准子程序,专用领域的通用子程序(如数学函数库 MATLAB)以及开发人员在开发过程中积累的具有复用价值的子程序代码。

程序包。程序包将逻辑相关的实体或计算资源组成一个集合,是一种比子程序粒度更大的构件。程序包是 Ada 语言的一个重要概念,较好地体现了封装(信息隐藏)的思想。构件可以通过程序包来实现。

类。作为构件的类包括开发工具提供的类库中的通用类,以及开发人员积累的可复用的类代码。

模板。模板也称为参数化数据类型,是比函数或类更高一级的抽象。最新版本的 C++ 语言中包含大量标准模板函数和模板类。

代码件在90年代依然得到巨大的发展,并且成为4GL 的基础之一,Microsoft 的 MFC、Borland 的 VCL、Sun 的 Java 类库对软件生产率的提高都发挥了巨大的作用。高级语言中对构件概念强调最多的应该要算 Ada 和 C++,Ada 语言中 Booch 构件是最具有代表性的例子。Grady Booch 在80年代用 Ada 83设计了一组软件构件,分为工具、结构、支持三大类,提供通用算法和数据结构的抽象以及其它通用功能[3]。Alexander Stepanov 则在 1995 年使 STL (Standard Template Library)成为 C++ 标准的一部分,目标是建立一个通用高效的算法和数据结构模板库。

软件产品的复用可以分为间接复用和直接复用两类。间

马 亮 博士生,主要研究方向为软件复用和软件构件技术。孙艳春 博士,主要研究方向为软件工程、CSCW。

接复用主要针对文档型知识的复用，复用者在分析理解文档后手工实现，无法提供自动工具支持。直接复用主要针对系统实现过程中的复用，在自动工具的支持下完成。Barhe 将直接复用分为黑盒复用和白盒复用两类[4]。黑盒复用是指将构件不加修改地应用到新系统中，白盒复用是指将构件进行适当修改之后再应用到新系统中。Barhe 认为大多数情况下对构件的复用都是白盒复用。Boehm 研究了对开发过程中对代码件修改所需要花费的代价[5]，认为随着所需修改部分所占比例的增加，修改构件所花费的代价将以非常快的速度增长，例如，修改一个构件20%代码所花费的代价将相当于重新开发这一构件所需花费代价的90%。

2.2 90年代初期对软件构件的理解

一个应用系统的开发过程通常包括分析、设计、实现、测试四个阶段，每个阶段都将产生相应的开发文档或产品。90年代开始研究人员逐渐认识到软件复用不应局限于源程序代码，分析、设计文档以及测试数据和方案等对开发活动有用的信息都具有复用的价值，因此软件构件应当包括分析件、设计件、代码件、测试件等多种类型。其中有代表性的构件定义是91年 NATO 制定的软件复用标准中的定义[6]：

可复用软件构件(RSC)是可以被复用的软件实体，它可以是设计、代码或软件开发过程中的其它产品，RSC 有时也称为"软件资产"。

同早期的认识相比，这一定义将构件概念的外延大大拓宽了。

作为基本复用单位的类与整体系统之间依然存在着巨大的鸿沟，90年代中后期陆续提出的若干新概念试图去填补这一鸿沟，这些概念一度被看做分析件和设计件的基本形态，主要包括：

设计模式(design pattern)[7]。记录了软件设计时经常使用的一些被实践证明是成功的思想、结构和方法。目前对设计模式的研究主要集中在 OO 领域，OO 设计模式是一种比单个对象和类粒度更大的结构，描述了对象或类之间如何组织和交互来解决特定的问题。

框架(framework)。是一个可复用设计，它是由一组抽象类及其实例间协作关系组成[8]。框架构件的典型例子是 Taligent 公司发布的一套支持快速应用开发的工具集 CommonPoint，其中包括上百个面向对象框架。

软件体系结构(Software Architecture)。软件体系结构[9]力图实现比框架粒度更大的系统级的复用。

与此同时一些学者探讨了软件构件应该具备的基本属性，例如 Will Tracz 认为可复用构件应具备以下属性[10]：①有用性(Usefulness)：构件必须提供有用的功能；②可用性(Usability)：构件必须易于理解和使用；③质量(Quality)：构件及其变形必须能正确工作；④适应性(Adaptability)：构件应该易于通过参数化等方式在不同语境中进行配置；⑤可移植性(Portability)：构件应能在不同的硬件运行平台和软件环境中工作。

代码件之所以能够取得巨大成功主要原因是高级语言已经研究得非常透彻，有一个完善的便于机器实现的形式化标准，因而能够吸引众多开发商参与编程工具或通用子程序库、类库的开发和竞争，从而促进了代码件的大规模应用。但是系统分析、设计以及测试过程由于人为的因素较多，这些过程中的产品难以形成一个可以让机器理解的形式化描述，因此目前分析件、设计件和测试件的复用主要以间接复用为主。从软件复用实践的角度来看，这类构件并没有为软件生产效率的提高带来多大价值，到目前为止依然难以从学术研究的殿堂走到实际应用中去，实现级的构件依然是学界和业界关注的重点。

2.3 近年来对软件构件的最新认识

90年代中期以后随着分布对象、Internet、JAVA、Client/Server 计算、4GL 和可视化开发工具等技术以及基于构件组装的软件开发模式(CBSD)的发展，国际学术界对构件的认识逐渐又有了新的变化。研究人员相继提出了若干新的构件定义，其中有代表性的包括：

1996年 ECOOP 会议上提出的定义[11]：

软件构件是一个具有规范接口和确定的上下文依赖的组装单元。软件构件能够被独立部署和被第三方组装。

Szyperski 在1998年给出的定义是[12]：

软件构件是可单独生产、获取、部署的二进制单元，它们之间可以互相作用构成一个功能系统(functioning system)。

CMU/SEI 的 Felix Bachman 等人在2000年5月的一份关于基于构件的软件工程的报告中给出如下定义[13]：

构件：①是一个不透明的功能实现；②能够被第三方组装；③符合一个构件模型。

这些定义包含以下几个共同的因素：构件是二进制功能单元、符合构件模型(或具有规范接口)、允许不同构件开发商开发的构件进行组装。这些因素体现了研究人员对构件如何真正促进软件产业发展的新的认识。首先，要想使 CBSD 成为主流开发模式离不开大量软件开发商的参与和构件市场的形成，有效保护构件开发商的智力投资是非常重要的，构件源码的公开显然不利于智力投资的保护，因此二进制代码就成为构件形态的首选。另一方面二进制代码隐藏了构件内部实现细节，也有利于减轻构件理解的难度，避免由于内部的修改而可能带来的错误。其次，符合同一构件模型是构件制作和组装的基础，构件接口是复用者理解构件和构件组装的桥梁，一般来说构件接口是构件模型的一个组成部分，因此符合一个得到业界广泛认可的构件模型是构件走向实用化的一个不可缺少的条件。第三，CBSD 希望构件具有可插拔(plug & play)性，便于替换，基于构件的系统不能依赖于某个固定构件开发商，因此必须允许不同开发商开发可以相互组装的构件，这也有利于构件市场的形成。

目前符合上述定义并且在业界得到广泛应用的构件主要包括以下三类：

CORBA 构件。CORBA 是分布对象技术的鼻祖，尽管分布对象技术最早提出是为了解决网络环境下异构平台间的互操作问题，但在其应用的过程中却体现出了巨大的复用价值。CORBA 构件表现为具有 OMG IDL 接口的对象或对象组。

COM 构件。COM 是从 OLE 和 VBX 技术发展而来的。对象模型包括 COM、DCOM 和最新发展的 COM+三种。ActiveX 控件是 COM 构件最常见的形态。

JavaBean/EJB 构件。JavaBean 将可复用的 Java 代码包装成为构件，JavaBean 已经成为 Java 2的一个组成部分。

2.4 构件概念的未来发展

软件构件技术研究的目的是为了在实际软件开发的活动中得到应用，而不是停留在论文或研究报告中，构件概念的发展也正体现了这种观点。90年代初期出现了所谓 COTS(commercial-off-the-shelf)构件。COTS 不是个技术术语，它是指制成的、可以在市场上购买到的产品。COTS 构件的出现使人们

看到了构件市场的端倪。COTS 构件从一开始就强调内部细节的不透明性、组装过程的可插拔性等特点，早期的 COTS 构件包括通用子程序库、通用类库甚至一个完整的应用子系统，近年来则以 COM 和 JavaBean 构件为主。注重构件技术的实用性是未来构件概念发展的趋势，今后将有可能沿着以下三个方向发展：

(1)引入网络服务的思想　2000年6月和2001年2月微软和 SUN 公司分别推出了 NET 框架和 ONE 体系结构，其共同特点是强调网络服务，网络服务使软件可以通过租赁的方式通过 Internet 提供给用户使用，从而避免了盗版、用户使用频率、版本更新等一系列问题。网络服务构件将是未来构件的一种重要类型。

(2)构件与编程语言相结合　目前的高级程序设计语言并不直接支持构件的概念。CBSD 想要真正得到大规模的应用需要开发工具提供更强大的支持，将构件概念引入程序语言是国际上一些学者的研究方向。例如 M. Flatt 做了大量有关面向构件的编程语言的研究[14]，他认为基于构件的开发应当从程序语言级支持构件的定义、交互、编译和链接。商业语言中 Java 已经使 JavaBean 成为自己的一个组成部分，微软也在 VC 中提供 ATL 以方便 COM 构件的制作。尽管需要解决的问题还很多，成为高级语言中的一个组成部分也将是未来构件概念的一个发展方向。

(3)强调构件的领域专用性　软件领域的广泛性是 CBSD 难以实施的一个重要原因。相对来说某一特定传统行业的领域范围则狭窄得多，因而有利于标准构件的研究和生产，软件系统的开发则需要面向社会的各行各业。近年来人们逐渐认识到复用的重点应当转向特定领域，软件构件应当分为产品专用、领域专用、领域通用三个层次。大量的面向领域的构件的出现是 CBSD 成功的关键因素之一。

结论　软件构件概念虽然已经出现三十多年了，但是到目前为止依然没有形成一个能够被广泛接受的定义。不同的研究人员对构件有着不同的理解，而构件概念本身也在不断发展着。构件概念的歧义和发展与软件复用息息相关，构件概念的歧义是由于研究人员所关注的复用领域的不同，构件概念的发展源于软件复用在技术上进步和实践活动的深入。总的来说，目前对构件存在广义和狭义两种理解，广义的理解强调构件的复用价值，认为凡是应用系统中可以明确辨识且具有复用价值的构成成分都可以称为构件。而狭义的理解则注重构件在实际应用系统开发的作用，将构件限制于目前业界流行的 CORBA、COM 和 JavaBean 等技术上。然而从现状来看 CBSD 离大规模普及依然有很大的距离，构件概念仍然需要进一步发展，以进一步提高对构件制作、构件组装和构件市场形成的支持。

参考文献

1 McIlroy M D. Mass-Produced Software Components, Software Engineering Concepts and Techniques. In:1968 NATO Conference on Software Engineering, Van Nostrand Reinhold, 1976. 88～98

2 Deremer F, Kron H. Programming in the Large Versus Programming in the Small. IEEE Transaction on Software Engineering, June 1976. 321～327

3 Booch G. Software Components with Ada - Structures, Tools and Subsystems. Menlo Park, CA.: The Benjamin/Cummings Publishing Company, Inc., 1987

4 Barhes B H, Bollinger T B. Making reuse cost-effective. IEEE Software, 1991, 8(1):13～24

5 Boehm B W. Megaprogramming, Keynote speech. ACM Computer Science Conf. Phoenix, Ariz., Feb. 1994

6 NATO Standard for Development of Reusable Software, NATO Communications and Information Systems Agency Components, 1991

7 Gamma E, et al. Design Patterns. Addison-Wesley, 1995

8 Johnson R. Ralph Johnson's Framework Homepage. http://stwww.cs.uiuc.edu/users/johnson/frameworks.html

9 Shaw M, Garlan D. Software Architecture. Prentice Hall, 1996

10 Tracz W. Confessions of a Used Program Salesman- Institutionalizing Software Reuse. Addison-Wesley Publishing Co., New York, NY: April 1995

11 6th International Workshop on Component-Oriented Programming. http://ecoop2001.inf.elte.hu/workshop/WCOP-ws.html

12 Szyperski C. Component Software. Addison-Wesley, 1998

13 Bachman, et al. Technical Concepts of Component-Based Software Engineering:[CMU/SEI-2000-TR-008]. 2000/5

14 Flatt M. Programming Languages for Reusable Software Components:[PhD thesis]. Rice University, Department of Computer Science, 1999

(上接第85页)

8 Guo L, Matta I. Search Space Reduction in QoS Routing:[Technical Report NU-CCS-98-09]. Northeastern University, 1～18

9 ATM Forum Technical Committee. Private Network-Network Interface Specification Version 1.0. af-pnni-0055.0000

10 Murthy S, et al. Loop-Free Internet Routing Using Hierachical routing Trees. IEEE INFOCOM'97, 101～108

11 Behrens J. hierachical Routing Using Link Vectors. Sun Microsystems, Inc.

12 Vogel R, et al. QoS-Based Routing of multimedia Streams in Computer Networks. IEEE JSAC, 1996, 14(7):1235～1244

13 Crawley E, et al. A Framework for QoS-based Routing in the Internet. Internet draft, 1998

14 Shin K G, et al. A Distributed route-selection scheme for establishing real-time channels. HPN'95, 319～329

15 Hou C J, et al. Routing Virtual Circuits with Timing Requirements in Virtual Path Based ATM Networks. IEEE INFOCOM'96, 320～328

16 Rouskas G N, et al. Multicast Routing with End-to-End Delay and Delay Variation Constrains. IEEE JSAC, 1997, 346～357

17 Zhu Q, et al. A Source-Based Algorithm for Delay-Constrained Minimum-Cost Multicasting. IEEE INFOCOM'95, 377～385

18 Waitzman D, et al. Distance Vector Multicast Routing Protocol. RFC 1175

19 Moy J. Multicast Extensions to OSPF. Internet draft, 1992

20 Matta I, et al. On Routing Real-Time Multicast Connections. Northeastern University