计算机科学 1997Vol. 24№. 2

大粒度软件复用*)

张志华 全炳哲 金淳兆
(吉林大学计算机科学系 长春 130023)

摘 要 In terms of the granularity of the reusable component, software reuse can be divided into three different levels: small-scale reuse, medium-scale reuse and large-scale reuse. This paper discusses the current research state and the features of large-scale reuse, and the problems need to be solved in large-scale reuse. At last, we presents a method to facilitate the large-scale reuse.

关键词 Software reusability, Large-scale reuse (resue-in-the-large), Software architecture, Standard interface.

自八十年代早期已来，国内外的学术界和软件界都十分重视复用技术的研究和应用。软件复用是指在构造新软件系统的过程中，对已存在的软件开发知识(开发过程和技能)和软件开发各阶段的各种结果的重复使用。关于复用的分类标准有很多种。例如，按软件开发阶段、抽象级别、复用性质(产品和技能)等[6]，根据 Krueger 等人的分类标准，软件复用按复用粒度大小和抽象层次的不同，分为三类[5]：

①小粒度复用，即程序代码的复用。程序代码包括源代码和目标代码，这种复用主要表现为函数、子程序、Ada 语言中的包、面向对象中的类、方法的复用。

②中粒度复用，即软件设计结果的复用。进一步按复用粒度的大小，又分为两种：微体系结构的复用和框架应用程序(也称为宏体系结构)的复用。二者之间没有明显的分界。前者注重于如何对系统的(局部)行为进行概念建模和解释，而后者以微体系结构为基础，注重系统的全局结构的建立。在面向对象领域中，微体系结构由描述相关的类及其相互关系的设计和代码两部分组成。而框架应用程序的复用对象则是组成系统的各微体系结构及其相互关系。

③大粒度复用，即应用子系统的复用。复用对象是独立开发的应用程序或子系统。在复用过程中，它们不能作任何修改和扩充。通过一些标准协议，可使这些大粒度部件(应用程序)协同工作，共同解决某一问题。通过这种方式，大的、复杂的系统可以被快速而低成本地开发出来。

到目前为止，人们对小粒度复用进行了长期的研究和实践，发现代码复用并不如人们所期望的那样高，其中有技术原因，也有非技术原因。非技术原因主要表现在组织方面、管理方面、文化方面及经济、法律等方面。主要的技术原因有：开发一个软件所需的大量工作往往在分析和设计阶段，编码只占全部工作的一小部分；从部件库中检索部件的质量和效率不高；验证和确认也需要大量的工作；检索出来的部件的功能和接口与人们所希望的类似，但并不完全一样，若要使用此部件，必须对其作必要的调整，大大增加了复用代价。

由于上述种种原因。近年来，人们开始转向了中、大粒度复用研究。通过中粒度复用，软件设计者们在开发一个新软件系统时，可以利用已有的需求分析、设计的思想和结果。通过大粒度复用，人们可利用已存在的软件系统来组成新的目标系统。设计目标系统时，只需考虑到各子系统相互作用的框架结构，而不必关心设计和实现细节。本文将讨论有关大粒度复用的若干问题。

*)本文得到"95"攻关项目、国家自然科学基金和吉林大学"符号计算与知识工程开放实验室"的支持。
张志华 硕士研究生。全炳哲 副教授。金淳兆 教授。

一、大粒度复用的现状及其特点

自八十年代早期已来，人们主要致力于小粒度复用，只是在最近几年，才开始注意大粒度、领域相关的复用部件族的设计[9]，尽管在某些方面已取得了一些成果，但总的来说，大粒度复用还很不成熟，还有许多问题有待解决。在复用部件的体系结构，部件的描述机制以及复用部件的标准化等方面还不是很成熟，还需作进一步的研究。

目前，人们主要从领域分析、体系结构、部件描述方法、接口及互操作性以及组装和支撑系统等方面来研究大粒度复用。

与代码复用相比，大粒度复用更具有领域相关性，Jeff Peolin 在第六届软件复用工作会议上曾谈到，以存放大量的低质量或太一般化的部件的复用库为中心的方法，不象人们所期待的那样有效，但相对少而精心设计的领域相关的部件的复用，却取得了令人难忘的效果[9]。领域分析是系统的可复用体系结构设计和相关应用系统族的部件设计的起点。由于大粒度部件结构复杂，为减小复用成本，不应对其进行修改，因此领域分析更为重要。通过比较一个领域的各相关应用之间的共同点和不同点，设计未来的复用部件将更具有针对性。

大粒度复用部件的体系结构的设计、描述和标准化也是同代码复用不同的地方。大粒度部件具有复杂的结构。尽管在复用中，大粒度部件不能修改和扩充，但很可能在协同合作的部件间产生体系结构不匹配的问题(有关体系结构不匹配的问题，将在下一节专门讨论)。为了避免此类问题，必需研究复用部件的体系结构，将其标准化、明显化。T. J. Mowbray 从系统适应性和软件集成的角度由低到高提出了软件体系结构的六个层次，讨论了适用于复用的体系结构[6]。文章认为，在系统集成方面，面向对象的体系结构比非面向对象的体系结构要优越。在此六个层次中，第 1 至 3 层为非 OO 体系结构；第 4 层虽为 OO 体系结构，但仅在编码阶段才考虑复用和集成问题，因此，复用率不高；第 5 层在设计时就考虑了这些问题，建立了 Design Pattern 和 Design Framework；第 6 层体系结构是指在系统中建立了标准化的子系统(具有标准的接口)，用于多个系统间的相互作用。另外，为了提高各部件的独立性，Gisi 又提出了一种软件总线式的体系结构，一个部件所需的数据和控制直接从"总线"上取，而忽略了其他部件的存在。为了减少系统的复杂性，提高复用度，Philips 研究室的 Linder 和 Müller 等人针对远程通信系统提出了一种层次式的体系结构，复用的基本单位是"构件(Building Block)"。所谓"构件"，是指构造系统的软件部分的基本单位，每个部件具有封装的内部结构和外部接口。这些部件按其通用性分为若干层，构件之间的相互作用通过接口和回调函数实现。同时，M. Shaw 和 Deline 等人提出了一种软件体系结构的抽象描述方法及相应的支持工具[7]。

为了描述部件，需要确定部件模型和部件描述语言。一般来说，部件模型是在给定的领域内加以考虑的，是对特定领域的部件的抽象描述[10]。当前比较著名的部件模型有 3C 模型、REBOOT 模型等。部件描述语言用来捕获给定领域的部件的重要属性，它建立在此领域的部件模型之上。目前，部件描述语言主要有 LIL，CDL，CIDER，LILEANNA，RESOLVE 等[10]。大粒度部件同其他部件相比有其独特之处。例如，具有复杂的结构、完整的进程边界、可能会引起互操作性和体系结构不匹配等问题。因此，为了充分有效地利用大粒度部件，必须针对其特点开发描述方法。Haikuan Li 提出了一种描述大粒度部件(LSC)的方法并提出了设计框架、设计实例和领域资源等概念[4]。设计实例即是对当前部件及其体系结构的描述，设计框架由设计实例经泛化和参数化获得，在框架中，没有具体的部件，取而代之的是具有相应功能的虚设的角色(actor)，领域资源是指当前系统中对每一特定角色的可用资源的集合。这种方法采用层次结构，将设计框架、设计实例、领域资源结合在一起，有利于部件的检索和复用，也使体系结构得以明显化。

独立开发的大粒度部件具有完整的进程边界，可以单独运行。如果想使其在同一系统中协同工作，必需的接口和互操作能力是必不可少的。Microsoft 公司为达到系统集成的目的而提出了 OLE(对象链接与嵌入)技术。OLE 是一个以面向对象为基础的系统间相互作用的框架，它定义了若干标准接口，应

用程序的相互作用完全在接口中进行，减少了隐含操作。在对象化操作系统 Windows95 中，很多部件已成功地使用了 OLE 技术。Kevin J. Sullivan 在其技术报告[8]中，探讨了利用 OLE 来进行大粒度复用的问题。与之类似的还有 IBM、Apple 公司的 OpenDoc，OMG 的 CORBA 也为应用程序的互操作提出了一种标准。

在大粒度部件的组装和支撑系统方面，Hewlett-Packard 开发了一种称为"领域相关工具箱(domain-special kits)的方法[9]。一个典型的工具箱包括：①部件。这些部件有完整的文档说明，经过严格测试，按照相容的部件种类被打包；②框架。在此框架中，兼容的部件可与其他的工作产品结合起来；③用来结合部件和增加功能的胶合语言；④可以随时运行的通用应用程序；⑤构造环境；⑥执行环境。

HP 正在定义和开发一系列这样较小规模的领域相关工具箱。HP 实验室 Pankej Garg，Mehdi Jazayeri 和 Michadl Greech 等人认为，这样的工具箱能够将关于一个应用领域的一般知识整理在一起，这样，创建一个新的应用程序时，可以通过对这个工具箱的定制来完成，而不必对其完全重新进行程序设计[9]。

如上所述，大粒度部件具有如下特点：①能够独立执行并完成一方面的任务，解决某一特定的问题；②具有复杂的系统结构；③由于开发这样粒度的可复用部件的成本较高，为了达到实用化的目的，降低复用成本，不应对其进行修改(或仅对其做少量修改)。同时，可复用部件必须满足三个条件：①完整的进程边界，每个部件能够独立工作，具有良好的封装特性；②部件的完整的接口说明，所谓"完整的接口(界面)"是指跨越部件边界的所有行为(过程调用，消息发送等)都在文档中明显地表示出来。③组成同一系统的部件间应具有相容的体系结构，以避免体系结构的不匹配。

二、实现大粒度复用需要解决的问题

我们认为，提高软件质量和软件生产率的未来突破将依赖于我们结合已存在的软件以产生新的应用的能力。通过利用可复用部件来构造新的系统将比大多数软件开发者使用的从头作起的方法要快得多，而且使创建大的、高质量的软件系统成为可能。所以，在过去的几年中，人们对这种组合方法从各种角度，例如，部件的相互作用的工业标准，领域相关的软件的体系结构和工具集等进行了研究和开发，但是，尽管如此，离大粒度复用的实用化仍相距甚远[2]。所遇到的主要困难既有技术上的，也有非技术上的。非技术方面的原因主要表现在管理、法律、经济等方面，主要的技术原因有两个：一个是互操作性的问题；另一个是上面提到的体系结构不匹配的问题[2]。

引起"互操作性"问题的原因主要有：不同的编程语言造成部件间的会话能力低下；不兼容的操作平台；数据库模式的不匹配等。此类问题属于低层次的不匹配，经常出现在编码阶段。这种问题必须解决，若不解决，大粒度复用则更无从谈起。令人可喜的是，近年来的研究表明，在解决互操作性的问题上，已取得了很大进步[2]。例如，分布式对象技术、CORBA、OLE 等标准的提出及开放系统的研究等。

然而，即使解决了互操作性的问题，大粒度复用还可能不会成功，因为还有一类更为深刻、更为普遍的体系结构不匹配的问题。所谓"体系结构不匹配"是指在大粒度复用中，各复用部件对系统中的其他部件及系统结构的所做的假设的不匹配[2]。D. Garlan 将引起体系结构匹配的假设分为四类：部件的性质、连接器的性质、全局体系结构和构造过程[2]。对每种情况来说，在独立使用的情况下，各部件所做的假设都是合理的，但组合进一个系统中，就会出现上述各种不匹配。例如，一个面向对象数据库提供了一个标准对象类的扩展库，以使一般意义下的编程容易些。但是，如果我们建造有特定目的的数据模型而仅需使用其中很少几个类的话，显然，会有大量的代码冗余并造成运行性能的下降，这属于部件性质的基础结构(Infrastructure)的不匹配。再比如一个系统中，一个部件的跨越进程边界的数据是基于 C 语言的，类型是结构和数组，而另一个部件所用(或所提供)的数据是 ASCII 字符串。这样，将引起一个连接器性质假设中的数据模型的不匹配。这些部件所做的假设都是隐含的，因此，在实际构造系统前很难分析出来[2]。部件的体系结构以及与全局体系结构的不匹配问题是大粒度复用中普遍存在的现象，是大粒度复用的主要困难。

解决体系结构不匹配的关键途径就是复用部件的标准化，为此需要开展下列研究：①领域相关的可复用部件的体系结构，使系统中各部件、连接器和全

局体系结构明显化、标准化；②领域相关的部件的接口标准化和部件间数据通信的规范化；③同时，必须建立一种描述标准化部件及其体系结构的方法，以便于复用和组装这些标准化部件。

三、一种复用部件的标准化方案

为了探索大粒度复用的实用化，我们基于 Sullivan、Haikuan Li 等人的工作，提出了一种复用系统的体系结构，它是基于协调者的 Client/Server 结构的扩展。在这种体系结构中，有两种部件：功能部件和模板部件（协调者）。功能部件是指具有特定功能的可完成一定任务的部件，作为功能的提供者出现，是我们所要复用的部件。功能部件具有多个标准化接口，每个接口完成某类功能。一个接口由若干个标准方法组成，这些方法由其他部件调用。模板部件（协调者）是一个用于连接功能部件的框架，作为功能的调用者出现。模板部件的内部结构决定系统的行为。模板部件有若干个槽，每个槽扮演特定的角色。槽是模板部件与功能部件的连接部分，即功能部件嵌入的位置。槽同功能部件一样，由若干个接口组成，通过槽的接口与功能部件的接口相互作用从而达到模板部件与功能部件相互作用的目的，如图 1 所示。

这样设计可以提高功能部件的独立性。各功能部件间不可以直接相互作用，相互之间是不可见的，只有协调者知道到底有哪些部件加盟进本系统。这样能够在一定程度上避免体系结构假设的冲突。当一个功能部件需要另一个功能部件提供服务或请求数据时，将消息发送到协调者的相应接口，协调者根据角色选择另一功能部件并调用其方法，或以一标准化的方式向此功能部件索要数据。然后将此数据放在协调者的数据接口中，供原功能部件读取。

协调者中的注册信息在系统实例化（即组装）时确定，实例化时，为协调者中的每个槽决定功能部件，然后将所选中的功能部件的信息注册进协调者中。当执行协调者的初始化过程时，激活相应的功能部件，并对其初始化。

我们可以通过一个协调者和多个功能部件组成一个新的系统，它具有单层的结构。在一个大的、复杂的软件系统中，可能具有多层结构，也就是说组成这样的系统的单元可能是功能部件，也可能是由协调者和功能部件组成的子系统。因此，应使协调者具有象功能部件那样的外部接口，供外界使用，如图 1 所示。

这种层次的 IC 插件式体系结构，提高了部件的独立性，减少了部件间的假设冲突。这里需要指出的是，协调者应具有容错的能力，即根据不同的功能部件的接口能力来对其在功能、性能方面作出适当的调整。这样，在选择功能部件时，会提高灵活性，扩大可选范围。

从上述复用系统的体系结构中，我们可以看到，部件间的相互作用完全是通过接口（功能部件的接口和槽的接口）实现的，因此，部件标准化的一项重要工作就是对接口进行标准化和数据规范化。

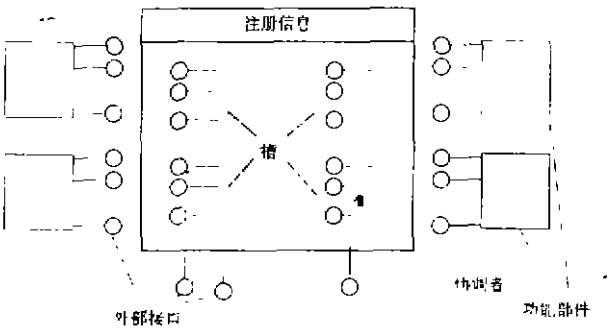

图 1　复用系统的体系结构

所谓“接口标准化”，是指建立一些标准化的接口，每个接口完成一定的功能，并且使接口间的相关度达到最小。接口标准化包括两个方面：一是，对单独的接口进行标准化，即此接口所要完成哪些功能，接口内有哪些方法，这些方法应是固定的，并完成固定的功能。例如，数据传送接口应负责（只负责）数据传送的功能等。在这方面已经取得了很好的成果。例如，Microsoft 公司的 OLE，就是一套标准化的接口协议。二是，对领域相关的部件建立标准接口组，这一部分是接口标准化的关键。对于给定的领域，应进行领域分析，在领域分析的基础上确定部件的功能需求，从而确定此部件的标准接口集。

组成同一系统的各部件（协调者和功能部件）有时要不可避免地传送数据。为使相互传送的数据能够理解，需要对数据规范化，领域无关的数据规范化是不可能的，只能是领域相关的，在领域分析的同时，确定所需的数据类型以便做好数据的规范化。

为了使系统的体系结构为其他人所了解，以便达到复用的目的，我们设计了一个对复用部件及其

体系结构进行描述的框架系统。此框架系统由框架模板和框架实例两部分组成。框架模板用于描述模板部件(协调者),框架实例用于描述模板部件实例化后的系统。框架模板由设计框架和部件资源两部分组成,设计框架用于描述系统的体系结构及接口,分为图形和形式化文本两部分。框架实例仅由设计框架组成。详细情况将另文介绍。

结束语 尽管大粒度复用还很不成熟,但许多人认为,大粒度复用是提高软件质量和软件生产率的最有效的途径之一。本文讨论了大粒度复用的当前研究现状和所具有的特点。目前,实现大粒度复用还有很多困难需要克服,除了互操作性问题之外,最主要的就是体系结构不匹配的问题。本文认为,复用部件标准化是使大粒度复用通往成功的关键途径。实现复用部件的标准化,就必须研究领域相关的体系结构,对复用部件进行接口标准化和数据规范化。本文提出了一种复用部件的标准化方案,讨论了复用系统的体系结构以及接口标准化和数据规范化等问题。尽管还需为实现大粒度复用付出更多的努力,但我们相信,我们会从中获取更大的好处。

参考文献

[1] Brockschmidt. K., Inside OLE, Second Edition, Microsoft Press, Redmond WA, 1995

[2] David Garlan, et al., Architectural Mismatch or Why it's hard to build systems out of existing parts. the 17th Intl. Conf. Software Reuse, 1995

[3] Richard Lajoie et al., Design and Reuse in Object-Oriented Framework: Patterns, Contracts, and Motifs in Concert, Centre de recherche informatique de Montreial, PQ H3A 2N4. Canada, 1994

[4] Haikuan Li, Reuse-in-the-large: Modeling, Specification and Management, Advances in Softwrae Reuse, 1994

[5] Hafedh Mili et al., Reusing Software: Issues and Research Directions, IEEE Trans. on Soft. Eng. ,21(6)1995

[6] Thomas J. Mowbray, What OO architecture benefits are you missing?, Object Magazine, 5(7)1995

[7] Mary Shaw et al., Abstractions for Software Architecture and Tools to Support Them, IEEE Trans. on Soft. Eng. ,21(4)1995

[8] Kevin J. Sullivan et al., Assessing an Architectural Approach to Large-Scale Systematic Reuse, Technique Report, Department of Computer Science University of Virginia Charlittesville, VA22903

[9] Ware Myers, Workshop esplores large-grained reuse, IEEE, Software, 11(1)1994

[10] Den Whittle. Models and Languages for Component Description and Reuse, Soft. Eng. Notes. 20(2)1995

(上接封底)

用户可以根据 Mach3.0 中关于核心与 pager 的 RPC 界面约定,编写自己的 pager 或 default_pager,实现特殊的文件系统或数据管理系统,满足应用的需求。

结束语 Mach3.0 代表的微内核思想为操作系统领域带来了许多新的研究课题,如何进一步减少线程切换及消息通讯开销,如何合理地划分操作系统功能,如何从硬件、体系结构上获得支持,使更多的操作系统功能放到核心外的服务器中实现。通过解剖和分析 Mach3.0,我们期望能实现更完善更高效的微内核机制,研究微内核结构下服务器的开发技术,将微内核思想和微内核技术应用于 MPP 系统研究中。

参考文献

[1 — 4] Open Software Foundation and Carnegie Mellon University, 《Mach 3 Kernel Principles》,《Mach 3 Kernel Interfaces》,《Mach 3 Server Writer's Guide》,《Mach 3 Server Writer's Interface》

[5] Accetta M., Baron R., Mach: A New Kernel Foundation for UNIX Development, In Proc. Summer 1986 USENIX Conf., 1986

[6] Golub, D. B., and Draves, R. P., Moving the Default Memory Manager Out of the Mach Kernel, Proc. the 2nd USENIX Mach Symposium, Nov. 1991

[7] 邓胜兰,基于 Mach 的虚拟共享存储服务器,计算机工程与科学,1994 年第 2 期(总 60 期)