基于构件/构架的软件开发方法

夏洁武　　钱乐秋

（吉安师专计算机系　吉安 343009）　（复旦大学计算机系　上海 200433）

摘　要　基于构件/构架的软件开发方法，以构件为中心，框架/构架为纽带，开发应用系统。本文从构件/构架的基本概念出发，对基于构件/构架的软件开发方法进行综述，讨论与此相关的主要问题的热点研究。

关键词　构件　框架　构架　基于构件/构架的软件开发

COMPONENT/ARCHITECTURE - BASED SOFTWARE DEVELOPMENT

Xia Jiewu

(*Computer Department of Ji'an Teacher's, Ji'an 343009*)

Qian Leqiu

(*Department of Computer Science, Fudan University, Shanghai 200433*)

Abstract　This paper gives the fundamental concepts, problems of Component/Architecture - Based Software Development(CaBSD). We also discuss current research efforts and results of CaBSD.

Keywords　Component　Framework　Architecture　CaBSD

1　引　言

软件重用是学术界和产业界一直致力于研究的问题。早期的软件重用主要以源代码与执行码为主，但由于对代码理解的困难，标准的不统一及传统程序设计语言本身对复用的支持力度不足等方面的原因，复用效率较低，所带来的功效也不显著。随着软件开发成本的上升，复用的概念也扩展到了分析与设计。九十年代，基于构件/构架的软件开发研究使重用步入了一个新时代。

利用可重用构件建立应用系统思想的提出，可追溯到 1968 年 NATO 会议。基于构件/构架的开发类似于硬件的“即插即用”技术，即能从所提供的具有一定标准的软件构件库中获得合适的构件，通过一定方式生成新系统。在过去的十年中，致力于相关方面的研究有很多[1]，如模块内部语言，模块接口规约和分析，特定领域软件构架，软件生成器，构架描述和配置等。

本文对基于构件/构架的软件开发方法进行综述，介绍构件/构架的基本概念，讨论与此相关的热点研究和应用前景。

2　构件，框架和构架

2.1 构件

在文献[1,2]中，对构件的概念作出解释。有观点认为在面向对象程序设计中，构件相当于对象类，在传统程序设计中，构件相当于过程和其所使用数据的封装体。普遍意义上的构件应从以下几个方面理解：

① 构件应是抽象的系统特征单元，具有封装性和信息隐蔽，其功能由它的接口定义。这样，重用者如果不想改变构件的功能的话，只需与构件接口打交道而不涉及其内容。

② 构件可以是原子的，也可以是复合的。因此它可以是函数，过程或对象类，也可以是更大规模的单元。一个子系统可看作是包含其它构件的构件。

③ 构件应是可配置的和共享的，这是基于构件/构架开发的基石，且构件之间能相互提供服务。

构件的定制以重用为目的，因此软件构件不仅需要较高的抽象和通用性，还应该可检索且易于组装，同时，软件构件总是寓于某种形式之中，有源代码的，目标代码的，也可以是规格说明层次上的，最终需要一种

夏洁武，讲师，主研领域：软件工程。

智能的互操作机制把它们连接起来，正是由于软件构件自身固有的特性，时至今日，对于构件定义的内涵和外延，尚无确切的界定。在软件系统级，软件构件和软件包本身并没有明确的界限，一个构件在某种情况下可以作为独立的软件运行，在另一种情况下又可以和其它构件集成在一起构成一个新的软件系统。

2.2 构架

Perry 和 Wolf 首次把软件构架定义为“元素、形式和约束的总和”，而在 1995 年 11 月的 IEEE 关于构架的专题讨论上，有六篇主题文献给予构架以不同角度的定义[6]。这些定义的共同之处包括：

- 构架是与设计的同义理解，是系统原型或早期的实现。
- 构架是高层次的系统整体组织。
- 构架是关于特定技术如何合作组成一个特定系统的解释。

总之，虽然构架的概念尚未得到统一认识，但都认为它是对软件系统的组成，系统结构及系统如何工作的较为宏观的描述。

一个软件构架应反映系统的功能性需求，非功能性需求，设计原理和构架风格（如管道、数据抽象、基于事件的隐式激发、库（包括数据库和背景系统）解释器、主程序和子路由等）。

2.3 框架

框架从重用意义来说，是一个介于构件和构架之间的一个概念。和前面两个概念一样，给框架下定义也很难，两个比较有代表性的定义是[2]：

- 框架是一个系统全部或部分的可重用的设计，是由抽象类、它们的实例及实例间内部作用方式组成的功能结构。
- 框架是一个能被应用开发者定制的应用骨架。

这两个定义并不矛盾，前者描述框架的结构与组成，后者描述框架在应用上的典型特征。框架变得越来越重要，象 Microsoft 公司的 OLE、IBM 和 Apple 公司的 OpenDoc 都是框架，Java 语言的兴起，又有如 AWT、Beans 等新的框架产生。有关框架的更多细节参见文献[1,2]。

2.4 三者对重用的支持

基于构件/构架的软件开发，是以构件为中心，以框架、构架为纽带组成新系统。根据通用性和实现方式简单化的要求。在构件制作中完成第一级标准化，再通过框架/构架实现第二级标准化，然后生成高层次的系统整体组织，建立具有较强的重用支持特点的新系统。与传统的软件开发方法相比，这种开发方法着重于开发出可重用构件，并重用这些构件生成新系统。三者对重用的支持如下：

- 构件是基础，也是基于构件/构架软件开发重用的最小单元。构件重用是代码级的重用。使一个构件完成一个简单的任务很容易办到，但重用率并不高，而一个带有很多参数和功能选项的构件重用率要高，却由于限制太多难以构造，也很难学会使用。因此，当我们设计一个构件时，不得不在简单性和功能上作一个折中。
- 构件的功能是由其接口定义的，框架/构架为构件提供了一个可重用的上下文关系。框架可为构件提供一个标准方式来管理错误、交换数据、激活操作。存储在框架构件库中的构件使用统一的框架接口，使得由构件库构造一个应用系统更为容易。框架和构架提供分级的可重用的抽象算法和高层设计，为把一个大系统划分为小的构件及描述构件间内部接口提供了标准。
- 构件重用包括可重用构件的制作和利用可重用构件构造新构件或系统。因为应用是一直在变化的，满足今天需要的构件不一定满足明天的需要。所以无论多么好的构件库，都有一个更新的问题，构件、框架、构架的三级划分和标准化处理，使构件及其制作具有良好的通用性，降低了构件库更新的难度。
- 框架重用包括代码重用和分析设计重用，一个应用系统可能需要若干个框架的支撑，从这个意义上来说，框架也是一个“构件”同时，框架又是一类特定领域的构架[3]。
- 构架重用不仅包括代码重用和分析设计重用，更重要的是抽象层次更高的系统级重用[7]。对已有的系统，我们可以根据构架提供的上下文关系来决定哪些部分可重用，可重用部分的功能是什么及哪些部分能相互连接及其连接方式等问题，建立新系统时，可在继承旧系统的某些结构的基础上作适应性修改。软件构架必须考虑软件系统的功能方面和非功能方面，非功能方面重要的一点是可重用性，一个好的软件构架应有利于应用系统开发，促进系统功能需求的实现和支持重配置。具体的 SA 往往和特定应用领域有密切联系。然而，尽管框架和构架的重用层次更高，比构件更为抽象灵活，但也更难学习使用。

3 基于构件/构架开发所需支持

基于构件/构架的开发，一方面是从可重用构件库中选取合适构件，通过裁剪或组装，按框架或构架规约重新配置和扩展的过程；另一方面是通过重用框架、构架的设计、分析构造新的可重用构件的过程。因此，基于构件/构架的开发需要如下几方面的支持：

- *构件的获取与描述手段* 包括如何描述构件的功能、接口、形态、性能和对环境的要求；开发人员如何检索构件库，从中选取符合要求的构件及构件的验证。
- *构件的组装与裁剪工具* 在获得构件后，如何按

照相应框架或构架的描述,将构件裁剪、组装成一个系统。

• *框架、构架的描述方法* 包括框架、构架的开发环境,对构件间通信关系的描述。对构架清晰的描述,是基于构件/构架开发的关键之一。

4 与结构化方法(SA/SD)、面向对象方法(OOA/OOD)的比较

结构化方法(Structure Analysts/Structure Design,SA/SD)是基于功能分解的种种系统开发方法的总称,适用于许多问题领域,并能产生规范的文档。与最初的杂乱无序的软件开发相比,一经产生,便广泛应用于开发系统。结构化方法将软件生命周期划分为需求分析、设计、编码、测试、维护等五个阶段,在各阶段间具有明显的顺延性和依赖性。

结构化方法将目标系统的功能作为划分模块的标准,按照分析阶段所确认的数据流和加工的关系,先使得模块的调用关系与功能的抽象层次相对应,再考虑底层模块的共享可能,是一种自上而下的开发。由于从高层模块开始就与当前需求密切相关,并且使数据结构的设计服从于功能实现的要求,所以尽管可以很好地满足当前要求,但由于对未来预测支持不够,实际上已经为以后的扩充和重用设置了障碍。再者,结构化开发方法某阶段出现的总是要回复到其前期阶段,从而造成大量人力、物力、时间上的消耗,并且,由于SA/SD方法在分析阶段和设计实现阶段分别采用不同的图形工具表示,从分析到设计阶段不得不经过两类表示之间的转换,这种转换也直接导致了在分析阶段和设计阶段难以平滑过渡的弱项。

和结构化方法相比,面向对象的软件开发与它的一个本质区别在于,OO是自底向上的设计。这种差别主要反映在对模块的认识上。OO方法将类看作是一种基本的模块,通过继承性加以功能和结构的扩充来建立模块间基于共享的联系,通过把能否形成类库中的可重用部件作为设计优劣的标准来控制模块设计的通用性。因此,OO方法从保证基本模块的通用性和可装配性入手,使得设计结果既能满足当前需求,又有利于以后的扩展及其它设计的重用,其各阶段采用一致的标记,实现了无缝的开发。因此,采用OO方法开发的系统较为稳定,当需求变化时,所做的修改小而简单。然而,也正是由于它的自底向上的构造方式,造成对系统整体性认识缺乏的缺陷,对未来系统行为的分析和预测的支持力度也不够。

基于构件/构架的软件开发是自底向上和自上而下的结合,因而在很大程度上结合了SA/SD方法与OOA/OOD方法的优势,克服了两者的缺陷,各阶段工作在系统的不同层次上进行。高层注重系统结构和约束,低层则强调构件的可重用性,构件根据框架/构架进行组合而生成新系统,因此人们可能通过对框架/构架的分析来预测未来系统的性能。整个开发各阶段皆以重用为准绳,从重用角度来看,结构化方法和面向对象方法重用层次较低,粒度也较小,基于构件/构架的软件开发的重用粒度已从类重用扩展到构件、框架、构架等更高层次的重用。

5 热点研究和现有工作

5.1 构件模型和构件描述语言

构件是基于构件/构架开发的中心,对可重用构件作一个清晰的描述是很有必要的,目标是使重用者很容易明白构件的功能及其它属性。一般认为描述可重用构件的途径有两条:构件模型和构件描述语言[3],且后者是基于前者的。在学术界和产业界都提出了一些构件模型[3~5],其中比较有代表性的是W.Tracz提出的3C模型[4],REBOOT项目中提出的REBOOT模型[5],OMG的CORBA/OM模型,MICROSOFT的OLE/COM模型。CORBA/OM和OLE/COM模型是由产业界提出的比较具体的模型标准,使用时有一定的限制,而3C模型和REBOOT模型是学术界提出的指导性模型,抽象层次比较高,用户可根据不同的问题域来扩展模型。

3C模型由构件的三个不同方面的描述组成:概念描述"构件做什么";内容描述构件的具体实现;上下文描述构件和周围环境在概念级和内容级的关系。REBOOT是一个基于刻面(facet)的分类模型,它所考虑的刻面包括:依赖、抽象、操作及操作对象,一个构件可以认为是刻面中的项的联合。两个模型的主要区别在于目标不一样,3C模型为构件制造者所欣赏,而REBOOT模型则更适合于已经建造好的构件。

目前,构件模型比较少,构件描述语言却比较多一些。较为典型的有LIL,ACT TWO,CDL,MELD,CIDER,LILEANNA,RESOLVE。这几种典型的构件描述语言各有特点,实际运用中,当用户选择一种语言描述构件时,必须综合考虑这种构件描述语言的几个因素:a)构件概念的描述;b)接口和实现要分开;c)结构和语法简单明了;d)设计灵活性的表达;e)语言实现机制。在文献[3]中,对这几种语言作了详尽比较。

无论是构件模型还是构件描述语言都离不开合适的开发工具,尤其是支持构件语义检查的工具,同时,可重用构件以各种形式存在于构件库中,因此,完善的构件库管理工具也是必不可少的。

5.2 框架的设计、描述和使用

有关框架的设计,描述和使用的研究有很多。已

有大量的商业用户界面[2]，如 OWL、ZAPP、OPENSTEP 和 MFC 等。VLSI 路由算法，超媒体系统，结构化图形编辑器，操作系统，网络协议软件等领域均有应用，并不限于用户界面。面向对象的抽象类、子类的概念和继承机制为框架的实现提供了有力支持。目前很多框架都用面向对象程序设计语言来实现[2][8]。

无论现有的还是未来对框架的研究，都面临着许多挑战，其中关键的一个是：一个框架中的构件哪些是可变的，哪些是固定的[10]。从开发角度看，在实现一个框架之前，我们得确信它包含的领域和设计是足够通用的，它必须精确到可编程，并具有推理能力。这需要许多参数化技术，如面向对象技术或特定构架技术。从重用角度看，首先必须使框架清晰易理解，最好有形式化规约和模式语言来描述，再者，框架的搜索也是一个重要问题。文献[8]开发的一个“框架浏览器”努力达到上述目标，依然没有解决搜索的完备性问题。

5.3 构架实现

对软件构架的研究始于九十年代初，研究软件构架具有重要意义。首先，区分出通用的结构对于理解系统的上层关系非常重要，尤其是可以在重用旧系统的某些部件的基础上构造新系统。第二，错误的软件构架可能导致灾难性的后果。第三，软件构架的形式描述对于复杂系统的高层性质的分析和表示起重要作用，如果原始的设计结构能够被清晰、准确地表示出来，有助于对软件的理解，同时也方便软件设计师之间进行交流。第四，对软件构架的深入理解便于软件工程师在多种设计中作出合理的选择。目前，SA 最活跃的研究领域归纳起来有以下几方面：

- *SA 形式基础的研究*[7] 致力于开发对软件构架的形式化描述和分析工具。包括构架中的构件接口，构件间的事件流描述，构件的行为抽象约束的表达，可执行模块和事件模式的描述。对构架的描述可以是图形化的模型如 Kruchten 的“4 + 1VIEW MODEL”，也可以是基于事件及约束的语言如 Rapide，PCL。大部分描述语言是基于 C，C ++，Smalltalk 或 Ada 的扩展。
- *构架实现的工具和环境集成*[1] 其目的在于给不同类型或角色的构件连接入一个系统的构架提供一个集成化环境。通常包含基于构件的构架模型，用以提供设计思想和构件的连接所需界面；构架规约分析工具，主要用于对构架决策验证；构件集成工具，把抽象的构架规约转换成可执行代码。
- *特定领域软件构架(DSSA)*[9] DSSA 是针对一个应用领域的软件构架，带有参考需求和领域模型及支持它的基础结构和将其实例化的过程。DSSA 通过参数化、继承机制及复本机制来支持构架的可变性和延展性。有关 DSSA 的研究包括不同领域 DSSA 应用，领域工程中的领域分析，建模工具和方法环境，参考构架的建立。不同的领域有着各自特殊的需求，对所有领域都适用的软件构架是不存在的，因而，DSSA 的研究备受关注。Ruben prieto Diaz 早在 1994 年就预言，特定领域的可重用工业，领域工程在 2000 年前将成为软件重用的主流问题，而现今国际上的重用研究热点无疑正在验证他的预言。

5.4 其它

实际上，对构件、框架和构架的研究并不是孤立的，基于构件/构架的软件开发必然要解决高层的框架、构架设计和底层的构件管理问题。因此，对框架、构架的描述包含了对构件的描述，配置框架构架的过程即为选取适当构件，合理配置的过程。与每一个问题相关的支持工具的开发，也是研究者颇为关注的热点。鉴于这一领域的新颖性，尚未有完整的、得到广泛认同的体系。

6 结束语

基于构件/构架的软件开发使软件象硬件一样能通过标准的组件来组装(组件可以是构件、框架)，是一种增量式开发。应用这种方法使开发人员能提高软件开发效率，其重用度的提高降低了开发成本，软件产品的维护工作量也相应减少，且为后续开发提供了基础。因此，围绕着它的各项研究进展正在预示一个软件开发新时代的来临。

参 考 文 献

[1] Andersen Consulting, Toward Software Plug - and - Play, Software Engineering Notes, 1997/May, pp. 19 ~ 29.

[2] Ralph E. Johnson, “Components, Frameworks, Patterns”, Software Engineering Notes, 1997/May, pp. 10 ~ 16.

[3] Ben Whit. T. le, “Models and Languages for Component Description and Reuse”, Software Engineering Notes, Vol. 20, No. 2. 1995/April, pp. 76 ~ 86.

[4] Implementation Working Group Summary, Reuse in Practice Workshop. Pittsburgh, Pensylvania, 1989/Jul.

[5] Weighted Term Spaces for Related Search, Inproceedings of the 1st International Conference on Information and Knowledge Management(CIKM'92), 1992/Nov. pp. 5 ~ 8.

[6] M. Boasson, The Aritstry of Software Architecture, IEEE Software, 1995/Nov. pp. 13 ~ 16.

[7] J. Davis and Roger B. Williams, Software Architecture Characterization., Software Engineering Notes, Vol. 20, No. 3, 1997/May, pp. 30 ~ 33.

[8] Hafedh Mili, Representing and Querying Reusable Object Framework, Software Engineering Notes, 1997, pp. 110 ~ 117.

[9] Will Tracz, Software Development Using Domain - Specific software Architectures, Software Engineering Notes, Vol. 20, No. 5, 1995/Dec., pp. 27 ~ 37.

[10] Pree. W, Design Patterns for OO Software Development, Addison - Wesley, Reading MA, 1994.