COTS构件 系统开发 软件开发 软件构件 ②

计算机科学2000Vol.27№.1

6-8,20

基于COTS构件的系统开发*)

COTS-Components Based System Development

张世琨 王立福 杨芙清 TP311.52

(北京大学计算机科学技术系 北京100871)

Abstract With the process of software component techuology, a new trend emerges with a large amount of COTS products used in software systems, and software systems exhibit the characteristics of constructibility and evolvebility In this paper, we introduce the system development process based on COTS components, analyze the short-term and long-term factors concerned in COTS-based system development. Some relevant techuologies are outlined.

Keywords COTS components, Component-based system development, Open system, Domain engineering

1 引言

随着软件技术,特别是软件构件技术研究和实践的不断深入,当前系统开发的一个新趋势是大量使用COTS(Commercial-Off-The-Shelf)产品。软件表现出构造性和演化性的特点,促进了软件构件业和集成组装业的形成、软件构件市场已初见端倪。CORBA、COM 和 JavaBeans 等构件标准的出现,为各自的COTS构件市场的形成奠定了基础。

基于构件的系统开发(Component-Based System Development,简称CBSD),有时也被称作基于构件的软件工程(Component-Based Software Engineering),强调通过集成现成的软件构件,构造大型软件系统。采用这种方法可以快速组装系统,潜在地降低开发成本,增强系统的灵活性和可维护性,减少同大型系统支持和升级相关的维护费用。CBSD 体现了"造船不如买船,买船不如租船"的思想。

诸如这些原因:削减系统开发和维护成本的经济压力;不断提高的COTS产品的数量和质量;构件集成技术的出现(例如CORBA,COM,JavaBeans等);开发组织内部不断增加的可复用软件。所以,基于构件的系统开发正在逐渐成为一种趋势。CBSD把系统开发的重点从编程转移到系统的组装方面[1],并由此带来一系列同系统开发、维护和管理相关的新问题。

2 基于COTS构件的系统开发过程

传统的软件开发通常都是"白手起家",开发组织承担系统所有部分的开发,并拥有对整个系统的控制权。一般过程如下:

- 获取和定义用户需求;
- 确定满足需求规约的体系结构;
- 详细设计体系结构内部的每个子系统;
- 编码、测试和调试子系统,以符合给定的需求;
- 集成子系统,形成完整的系统;
- 集成测试和系统发布。

在基于COTS构件的系统开发中,系统开发的观念被组装和集成现存构件的观念所取代。正如前面提到的,在传统开发方法中,系统集成通常是实现过程的最后阶段;CBSD则强调以构件集成为中心,进行系统构造。因此,可集成性是决定构件获取、复用和创建的主要考虑因素。

基于构件的系统开发过程包括四个主要活动:构件认证、构件剪裁、构件组装和系统演化,如图1所示。

2.1 构件认证

构件认证是确定现存构件在新的系统环境中是否合用的过程。当竞争产品市场存在时,构件认证同时也是一个选择构件的过程。构件认证还包括对构件开发和维护过程的认证(例如,构件生产厂商是否通过ISO9000质量体系认证)。

构件认证包括两个阶段:获取和评估。在获取阶

*)本文得到了国家"九五"重点科技攻关项目的资助。

段，确定构件的性质，包括构件功能和构件接口，以及构件的可靠性、可预测性、可用性等质量方面的因素。在某些情况下，一些非技术性的因素也很重要，例如构件生产厂商的市场份额、过去的商业表现和过程成熟度等。

评估是从一组候选构件中选择最合适的构件的过程。除了采用国际标准化组织(ISO)对产品评估的一般准则[2]，以及其他针对特定应用领域所需要的技术[3]外，评估还涉及到对构件进行性能测试、原型试验和了解其他用户对构件的使用情况等。

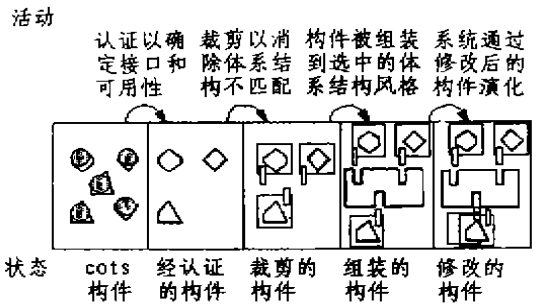

图1 基于构件的系统开发过程

2.2 构件剪裁

各个构件在生产时是为了满足不同的需求，并基于对环境的不同假设，因此构件在用于新系统时经常要进行剪裁。构件剪裁必须基于构件之间的冲突最小化原则，即消除体系结构的不匹配问题。针对不同的构件类型，可以采取不同的剪裁策略[4]。在CBSD中，构件一般可分为：

·白盒构件：可以访问构件的源代码，允许直接修改构件；

·灰盒构件：构件的源代码不能修改，但构件提供了扩展语言或应用程序接口；

·黑盒构件：只有构件的二进制可执行代码可用，且没有扩展语言和应用程序接口。

针对以上各类构件，其剪裁策略和方法各有优缺点。但需要强调的是，白盒方法由于修改了源代码，会引起严重的维护和演化问题。脚本语言、包裹器(wrappers)和桥接器(bridges)是目前常用的剪裁灰盒和黑盒构件的技术。

2.3 构件组装

构件必须通过某种定义良好的体系结构进行集成，该体系结构提供了把不同的构件组装成系统的机制(类似于计算机主板的作用)。例如，基于COTS构件开发系统，可以采用以下几种体系结构风格：

·数据库：系统构件之间通过中央数据控制共享信息；

·消息总线：各构件具有分离的数据存储，通过消息通知构件的变化；

·对象请求代理(ORB)：ORB提供了与语言无关的接口定义、对象定位和激活机制。

每种体系结构具有自身独特的风格。当前，最活跃的研究和产品开发集中在符合CORBA规范的ORB上。

2.4 系统演化

在基于构件的系统中，构件是系统变化的基本部件。为了修正系统的一个错误，可以用修改后的构件替换一个有缺陷的构件，即把构件作为可插换的部件。类似地，当需要附加功能时，可以把具有该项功能的新构件加入系统中。

然而，使用一个构件替换另一个构件通常是一件既费时又费力的工作，因为新构件不可能同旧构件完全相同，包裹器通常需要重新实现；而且要对新构件进行彻底测试，包括隔离测试，以及同系统其他部分的集成测试。

3 采用CBSD时需考虑的因素

由于COTS构件本身所具有的特性，例如：大多数的COTS构件是黑盒构件，系统集成者无法了解构件内部的细节；个别的系统集成者对构件的维护和演化的影响很小；构件的行为并不都象构件开发者所声称的那样。因此当实现基于COTS构件的系统时，需要考虑以下几方面的因素。

3.1 短期考虑

一个开发组织在采用基于构件的系统开发方法时，应考虑以下短期因素：

·开发过程：系统集成不再处于开发阶段的最后，而必须在前期进行计划，并在整个开发过程中加以控制。

·计划：在构件集成实际开始之前许多问题难以预测，因此估计开发进度和资源需求将是十分困难的。

·需求：构件的开发通常是针对特定的一组需求，并对环境做出假设。在系统开发中，用户实际需求和构件的使用常常出乎构件开发者的预料，因此需要在用户需求和当前可用构件之间进行折衷考虑。

·构件标准：如果一个组织选择基于构件的系统开发方法，同时还希望保持系统的开放性，那么必须把接口标准作为构件认证的准则。一个软件构件在何种程度上满足特定的标准，极大地影响系统的互操作性和可移植性。

·体系结构：标准和构件的选取需要坚实的体系结

构基础，这也是系统演化的基础。当从遗产系统迁移到基于构件的系统中时，体系结构尤其重要。

·构件认证：在构件认证方面存在的困难是，目前对构件的关键质量属性和构件度量还没有取得共识。针对这一问题的一些工作包括："SAAM：一种软件体系结构属性的分析方法"[5]，美国空军 PRISM 项目定义的在特定应用领域的认证过程，以及生产线资产支持。

·现存构件的复用：基于构件的系统开发强调使用可复用的构件。然而，即使在大量可复用构件存在的情况下，也经常需要对这些构件进行剪裁，以便集成到新系统之中。剪裁并不意味着对构件进行修改。常用的技术包括脚本语言、包裹器、桥接器和其它类型的连接件等。

3.2 长期考虑

除了以上的短期考虑因素外，还应考虑以下的长期影响：

·外部依赖性：当在系统开发中集成 COTS 构件时，系统开发者就失去了一定的自主性，增加了对 COTS 构件生产者的依赖性。构件生产者通常是基于现有构件的缺陷、市场需求、竞争对手的情况，确定何时以及怎样升级其构件，而不是根据每个开发组织的个别需求进行升级。新的构件版本要求基于构件系统的开发者决定是否把新构件加入系统之中。回答"是"意味着面临重写包裹器代码和进行系统测试的相当大的工作量；回答"否"意味着系统依赖于技术可能过时的老版本构件，而且不被构件的提供者充分支持。这就是为什么基于构件的系统方法有时被认为是一个风险转移，而非风险减少的方法。

·系统演化性：系统演化不是简单的即插即用。当系统中的许多构件是黑盒构件时（实际情况往往是这样），系统的集成者并不知道构件同环境交互作用的细节，替换一个构件常常在整个系统中引起波动效应。使情况进一步恶化的是，一个构件的新版本常常要求其他构件的增强版本，甚至在一些情况下，升级版本同系统中的其他已有构件不兼容。

4 相关技术

4.1 开放系统技术

开放系统技术的基本原则是在系统开发中使用接口标准，同时使用符合接口标准的实现。这为系统开发中的设计决策，特别是系统演化，提供了稳定的基础。同时，也为系统间的互操作提供了保证。开放系统技术具有在保持甚至改善系统性能的前提下，降低开发成本、缩短开发周期的可能。对于稳定的接口标准的依赖，使得开放系统更能适应技术进步和市场变化。

与开放系统技术相结合，可以增强基于 COTS 构件的系统开发能力，提高系统的可移植性和可维护性，减少对特定构件厂商的依赖性，从而在市场竞争和产品选择方面处于有利的位置。

4.2 领域工程

领域工程是为一组相似或相近系统的应用工程建立基本能力和必备基础的过程，它覆盖了建立可复用软件构件的所有活动。其中"领域"是指一组具有相似或相近软件需求的应用系统所覆盖的功能区域。

领域工程对领域中的系统进行分析，识别这些应用的共性和特性，形成领域模型；依据领域模型，设计面向该领域的软件体系结构（DSSA），并以此为基础识别、开发和组织可复用构件。当开发同一领域中的新应用系统时，可以领域模型和面向特定领域的软件体系结构为框架，确定系统需求，并进行体系结构设计、以此为基础选择可复用构件或开发新的构件进行系统组装。

通过领域工程产生的领域模型，确定了该领域系统的公共功能和数据；面向特定领域的软件构架，规定了构件接口、构件之间的关系和应用环境。以此为基础，有助于在基于 COTS 构件的系统开发中，识别潜在的可复用构件，并指导系统的集成、测试和维护。

结论 从早期的 UNIX 操作系统的管道/过滤器，到后来的 ODBC、VBX，以及 COM、CORBA、JavaBeans 等构件标准的出现，促使 COTS 构件市场在迅速扩大。基于 COTS 构件的系统集成将改变传统的软件生产方式，是软件工程走向成熟的一个重要标志。

参考文献

1 Clements P. From Subroutines to Subsystems: Component-Based Software Development. In: Component-Based Software Engineering: Selected Papers from the Software Engineering Institute. IEEE Computer Society Press, 1996

2 Information Technology-Software Product Evaluation-Quality Characteristics and Guidelines for their Use. Geneve, Switzerland: International Standards Organization/International Electrochemical Commission, 1991

3 IEEE Recommended Practice on the Selection and Evaluation of CASE Tools (IEEE Std. 1209-1992). New York, NY: Institute of Electrical and Electronics Engineers, 1993

4 Valetto G, Kaiser G E. Enveloping Sophisticated Tools into Computer-Aided Software Engineering Environments. In: Proc. of 7th IEEE Intl. Workshop on CASE. July 1995

5 Kazman R, et al. A Method for Analyzing the Properties of Software Architectures. In: Proc. of ICSE 16. May 1994

6 Brown A W. Preface: Foundations for Component-Based Software Engineering. Same to[1]

（下转第20页）

7. 若 E=f(E1,E2,…,En)且程序中有 f(x1,x2,…,xn)=f-expression,则 RE=Reduce(E'),这里 E' 是 f 中的静态参数 xi 用相应的 Reduce(Ei)替换所得;

8. 若 E=f(E1,E2,…,En),则

①通过调用 Reduce,计算 f 中的静态参数元组;

②通过调用 Reduce,计算 f 中的动态参数,会得到一个表达式表;

③RE=call $f_{statvalues}$(Dynvalues);

④如果 $f_{statvalues}$即不在 Seenbefore 中又不在 Pending 中,则把它添加到 Pending 中去。

3. 可移动代码的优化

设在异构网络环境的服务器中有一代码 P:

```
P=f(n,x)=if n=0 then I
    else if even(n) then f(n/2,x)↑2
        else x*f(n-1,x)
```

若有静态输入 n=5,则对 P 中的各变量进行分析,暂时不能计算的表达式作下划线标记,可得:

```
P=f(n,x)=if n=0 then I
    else if even(n) then f(n/2, x)↑2
        else x*f(n-1,x)
```

用上节的部分计值算法则可得:

$P_5=f_5(x)=x*((x\uparrow 2)\uparrow 2)$

显然,当可移动代码 P 的部分参数为已知(或相对固定不变)时,迁移代码 P_5到客户机比迁移代码 P 到客户机所需的通讯量要少得多,且在客户机中的运行效率要高得多,而且这种经部分计值后的代码并没有损失其异构性。进一步地,由于在异构环境中,可移动代码在客户机中大都是解释执行的,因此其运行效率较低,我们可以用部分计值的自作用能力,在语义上生成其编译代码,这样可提高其运行速度,为了说明这,我们设 I 是一个程序语言的解释器,c^I、p、d 分别为 I-语言的编译器、程序、输入数据,则有:

$$c^I(p)(d)=I(p,d) \quad (5)$$

由(1)和(2)可得:

$$f(k,u)=\alpha(f,k)(u) \quad (6)$$

(6)式作代换{I/f,p/k,d/u}可得:

$$I(p,d)=\alpha(I,p)(d) \quad (7)$$

(6)式作代换{α/f,I/k,p/u}可得:

$$\alpha(I,p)=\alpha(\alpha,I)(p) \quad (8)$$

(6)式作代换{α/f,α/k,I/u}可得:

$$\alpha(\alpha,I)=\alpha(\alpha,\alpha)(I) \quad (9)$$

因此有:

$$c^I(p)(d)=I(p,d)=\alpha(I,p)(d)$$
$$=\alpha(\alpha,I)(p)(d)=\alpha(\alpha,\alpha)(I)(p)(d)$$

可见 α(α,I)就是 P 的编译器,即由解释器可生成语言的编译器,若一个程序在客户端要多次使用,显然,编译程序比解释程序的运行要快得多,可以极大地提高程序的运行速度。

结论 随着计算机网络的不断发展,可移动代码的优化问题将成为人们研究的热点之一,本文提出的部分计值技术可应用到可移动代码中,且在并行计算中若大部分输入数据为已知时,用部分计值技术也可对数据的分布进行优化,达到较好的负载平衡,但对不同的程序设计语言构造出其相应的部分计值器将是一个复杂而有趣的工作,这也是我们进一步的工作。

参考文献

1 Futamnura Y, Nogi K, Takano A. Essence of generalized partial computation. Theoretical Computer Science, 1991,90:61~79

2 Kelsey R, Rees J. A tractable scheme implementation Lisp and Symbolic Computation, 1995,17(4)

3 Outerhout J K. Tcl and the Tk Toolkit. Addison-Wesl, Reading, MA, 1994

4 White J E. Telescript Technology: the Foundation of the Electronic Marketplace. General Magic White Paper, General Magic Inc., Sunnyvale, CA, 1994

5 Gosling J, McGilton H. The Java Language Environment: A White Paper. Sun Microsystems White Paper, Sun Microsystems, 1995

6 Jones N D, Gomard C K, Sestoft P. Partial Evaluation and Automatic Program Generation, Printice Hall, 1993

7 Jones N D. An introduction to partial evaluation. ACM Computing Surveys, 1996,28(3):480~503

(上接第8页)

7 Brown A W, Wallnau K C. Engineering of Component-Based Systems. Same to[1]

8 Brownsword L, et al. The Opportunities and Complexities of Applying Commercial-Off-the-Shelf Components. CROSSTALK. April 1998

9 Vidger M R, et al. COTS Software Integration: State-of-the-Art[online]. Available at: URL: http://wwwsel.iit.nrc.ca/abstracts/NRC39198.abs

10 Whitehead E J Jr, et al. Software Architecture: Foundation of a Software Component Marketplace. In: Garlan D, ed., Proc. of the First Intl. Workshop on Architectures for Software Systems. April 1995