计算机科学 1999Vol. 26 №. 2

对系统动态更新的探讨与设计

On Dynamic System Evolving

张升昕 董 淳

(复旦大学计算机系 上海 200433)

Abstract This paper introduces our research on dynamic software system evolving. A method of middleware-based dynamic software system evolving——we refer to it as 'software plug-and-play' is presented. We, at first, analysis the main problems for 'software plug-and-play' to solve, and list its advantages; Next, we describe the system architecture model. For the main steps, the corresponding models we built and the automated tools we designed are illustrated. In addition, the middleware works are discussed.

Keywords Dynamic system evolving, Architecture, Component, Middleware

软件系统,特别是大型商业应用系统常常需要不断地更新,以满足新的需求,对于基于构件的软件系统,只需更换有关的构件,即可避免修改整个系统。但对于有些系统如分布式系统,如果为了更换构件而将系统关闭,将是不可想象的,会付出巨大的代价。动态地系统更新——软件即插即用将是更好的选择。另一方面,要实现真正的构件市场,构件必须不受配置特性的限制而能通用,这样重用构件就可以由专业化的第三方来生产。中间件的出现给软件即插即用提供了一定的支持,基于中间件的软件即插即用将是未来较为可行的方法。

方法特点

基于中间件的软件即插即用方法并非是一种革命性的创造,它植根于早期的模块化结构方法,来源于如下的软件设计方法的发展道路:模块化结构→面向对象软件设计→基于构件软件设计→软件即插即用。当前对软件重用方法的研究虽然很多,但并没有形成一个受到广泛接受的实用方法,这其中的原因很多:它们往往只是对软件重用方法的某一方面的研究,重用的对象针对构件甚至整个系统的源代码,重用局限于某特定领域,或者重用的构件不能跨平台等等。比较之下,基于中间件的软件即插即用方法将更加易于接受,它使得对实时、分布式系统的更新高效、简捷而又代价较低,因为它具有如下特点:

· 集成化的过程描述。它包括软件重用的整个过程,首先从输入系统的需求分析规范开始,从需求文档中分析提取出系统的构架模型,定义构件的接口;接着对构件接口规范进行分析,绑定有关的构件;然后,按一定的标准将构件组装,成为可以即插即用的构件;最后,通过中间件动态地实现构件的即插即用。

· 形式化的模型。对软件重用的几个步骤,建立了较形式化的模型,给出较智能化的工具,这样对于此方法的正确性有一定的保证,同时最大程度地减少设计人员的参与。在定义系统的构架阶段,可以使用编译技术中的语法分析工具;在规范分析阶段,建立逻辑子类模型,使用自动推理工具;在构件生成阶段,使用基于知识的专家系统。

· 可执行的重用构件。和重用源代码的传统方法不同,软件即插即用方法重用的是二进制代码构件,摆脱了编程语言的限制。当然,这种构件必须经过组装,组装后的构件具有统一的接口,构件之间通过接口通信。

· 配置无关性。构件可以是在不同环境、不同地点、不同时间开发的,甚至可以从构件市场上购买,互相连接的两个构件可能具有完全不同的配置特性,这种配置特性同开发环境和运行环境相关。

· 地址透明性。互相连接的两个构件没有必要知道彼此的地址,即插即用的构件可以位于不同的进程或者不同的机器。

系统构架模型

虽然软件构架这个术语已经出现很久,对它的

研究目前也十分流行，但软件工程界对它并没有广泛而统一的观点，没有给出唯一的定义，不同的研究有不同的角度。一般而言，构架是指组成系统的构件的组合结构和构件之间的连接方式。在本文中，我们强调的是构件（客户和服务者）之间的通信。下面介绍软件即插即用构架模型的主要组成部分：

• 构件（Component）：组成系统的独立的功能单位，它可以是结构模式中的模块、面向对象模式中的类或者就是一组函数的集合等。构件分为原子构件和组合构件两类，组合构件由原子构件组合而成。构件是软件即插即用的基本单位。

• 客户和服务者：根据构件之间的实现关系可以将构件分为客户构件和服务者构件。客户构件通过调用服务者提供的服务实现自己的全部功能，而服务者要将自己实现的功能提供给客户。一个客户也可以是另一个客户的服务者，同样一个服务者可以是另一个服务者的客户。软件即插即用方法允许动态地更换服务者，而不影响客户的运行。

• 中间件（MiddleWare）：中间件是位于其它软件之间的一层软件，它将应用软件与系统软件和底层运行环境的其它技术或特有属性分隔开来。中间件提供了软件即插即用方法的技术基础。

• 接口：每个构件都有与其相连的构件接口，接口描述该构件的功能集合，接口分为'提供'接口和'要求'接口两种，一个构件可以有多个'提供'接口和'要求'接口。'提供'接口描述一个构件向其它构件提供的服务或功能实现，'要求'接口描述一个构件为了实现自己的功能必须从其它构件得到的服务。'要求'接口对于软件即插即用方法十分重要，客户构件不必和某个特定的服务者固定地链接在二进制代码中，只要一个构件能够提供'要求'接口中指定的服务，它就有可能成为服务者。

• 绑定（Binding）：如果一个构件能够满足另一个构件的要求，或者说两个构件能够构成客户/服务者关系，就可以将它们连接起来以实现客户的功能，这种连接通过绑定来实现。绑定将客户的'要求'接口和服务者的'提供'接口清楚地连接在一起，并形成一个组合构件。

```
component Client
begin
  requires requiredFunction;
end;
component Server
begin
  provides providedFunction;
end;
component Binding
begin
  Client clientInstance;
  Server serverInstance;
  bind clientInstance. requiredFunction to serverInstance
       providedFunction;
end;
```

图 1　构件绑定

• 配置特性：描述构件实现关于开发环境（编程语言、对象模型等）和运行环境（平台、中间件等）的依赖。在将客户和服务者封装为可执行的二进制代码时必须要用到这些配置特性。对于必须的配置特性，可以缺省。每个构件可以对应多种配置特性组合。在下图的例子中，构件 Client 的配置特性 CPP Client 指出它是用 C++ 语言编写的、运行在 Windows NT 上；构件 Application 的配置特性 CPPApplication 指出，它的一个实例 clientInstance 用 CPP-Client 描述其配置特性，并可以使用名称 Application 通过底层通信中间件的名称管理服务来访问它的这一配置特性。

```
configuration CPPClient of Client{
  property OS NT;
  property language CPP;
};
configuration CPPApplication of Application{
  CPPClient clientInstance;
  property name Application;
};
```

图 2　配置特性

图 3 描述了基于中间件的软件即插即用方法的具体过程，下面分别讨论其中的每一步骤。

抽取规范

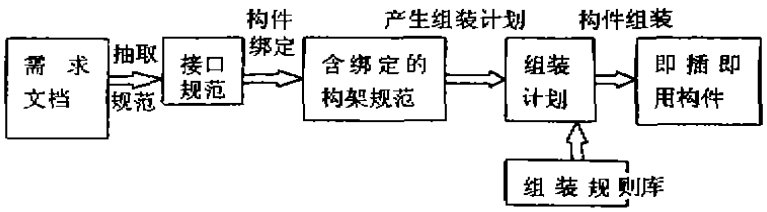

图 3　基于中间件的软件即插即用方法的全过程

从系统需求文档抽取出构件规范是软件即插即用方法的第一步，具体的工作包括定义系统的构件组合，每个构件的接口规范，比如'要求'和'提供'接口、配置特性等，最后的规范以形式化语言表达。此软件即插即用方法专门开发了CDL(构架规范语言)用以描述系统构架规范。抽取规范的具体实现可以手工完成，也可以利用有关的编译工具。

客户和服务者的绑定

客户和服务者的绑定，也是对构件规范的分析过程，以判断客户的要求能否被服务者的服务去实现。规范分析是一个十分繁琐而又必须的工作，应尽量采取形式化的方法，减少设计人员的参与。下面先介绍逻辑子类(Logic Sub Type)的概念系统，它是此形式化规范分析方法的基础。

逻辑子类：描述两个接口规范之间的实现关系。如果一个构件能实现规范 S_i 就能实现规范 S_k，则称 S_i 是 S_k 的逻辑子类。那么，实现 S_i 的构件就能透明地代替用来实现 S_k 的原构件。逻辑子类分为结构子类和行为子类两种，结构子类是行为子类的基础。首先给出结构子类(记为$\leqslant_s$)的递归定义：

· 基本结构子类包括有关基本数据类型的子类关系：Short$\leqslant_s$ Long，Float$\leqslant_s$ Double。

· 结构化数据类型(A、B)的子类关系定义为：如果对于结构化数据类型B中的任意类型为T的字段a，A中存在一个字段名也为a且类型为T的结构子类的字段，则 A$\leqslant_s$ B。

· 操作(f、g)的子类关系定义为：如果f的所有输入参数都是g的'相应'输入参数的超类(Super-Type)，并且g的所有输出参数和返回类型都是f的'相应'输出参数和返回类型的超类，则 f$\leqslant_s$ g；在这里，'相应'是指参数在参数表中具有相同的参数名，或在相同的位置。

· 接口规范(A、B)的子类关系定义为：如果对于接口规范B中的任意类型为T的操作a，A中存在一个操作名也为a且类型为T的子类的操作，则 A$\leqslant_s$ B。

结构子类定义只根据结构来判断，特别是要求同名匹配会引起规范分析的困难和错误。行为子类定义从语义角度来弥补结构子类定义的不足。下面给出行为子类的定义：

· 操作(f、g)的子类关系定义为：如果操作g的前置条件蕴涵操作f的前置条件，并且操作f的后置状态并上操作g的前置条件蕴涵操作g的后置状态，则操作f是操作g的子类。

· 接口(A、B)的子类关系定义为：如果接口A的不变部分等价于接口B的不变部分，并且接口B的协议是接口A的协议的子集，即，接口B所能接受的操作序列包括接口A所能接受的操作序列，则接口A是接口B的子类。

在图4中，接口 Policy1 是接口 Policy2 的结构子类，但不是接口 Policy2 的行为子类。

```
Interface Policy1{
    States{ UnAuthorized(init); Authorized};
    Void Authorized (in string agent, in string password)
        Postcondition (Authorized;);
    Void ChangePremium (in string client name, in long
        year, in double percent)
        Precondition{Authorized;};
};
Interface Policy2{
    Void ChangePremium (in string password, in string
        client_name,in short year,in float percent);
};
```

图4 逻辑子类

结构子类定义和行为子类定义的一个本质区别是行为子类关系是个不可判定的关系，因为证明操作条件之间的逻辑蕴涵关系并没有可行的方法，好在实际的规范分析方法中只需要大致的判断就可以了。

根据以上定义，我们采用自动推理技术实现规范分析器 Matchable。如果可以构成一对客户/服务者，就将它们绑定起来。在绑定过程中，还需要用到一个适配器(Adapter)。适配器提供客户和服务者之间接口的转换，它的接口和客户的'要求'接口相同，而用服务者的'提供'接口来实现。适配器可以在两种情况下发生作用：

· 当规范分析成功时，如果客户和服务者的接口并非完全相同，它们仍然不能直接通信，可以用适配器连接；

· 当规范分析失败时，产生一个适配器模板，它包含客户可以被服务者提供的服务，而将未被服务者提供的服务留给设计者去填写。

对中间件的分析

在客户/服务者模型中，中间件介于客户和服务者之间，客户和服务者并不直接通信，而是使用统一的标准接口通过中间件来交换数据。这样，客户和服务者可以保持相对的独立性，使得构件即插即用成为可能。当前较流行的中间件标准有 OMG 的 CORBA 和 Microsoft® 的 OLE 等，下面以 OLE 为例，分析中间件的构架模型和功能特点。

OLE(对象链接与嵌入)是以 COM(构件对象模

型)作为其底层构架基础,在 COM 的支持下,OLE 提供了较高级的各种服务包括:混合文档、数据交换和其他的应用程序间的互操作等。COM 使得不同地点、不同语言、不同平台的二进制软件构件相互连接和通信,具体而言,COM 具有以下相关的功能特点:①定义了构件互操作的二进制标准;②与编程语言无关;③得到许多平台的支持(Microsoft® Windows ®, Windows95, Windows NT™, Apple ® Macintosh®);④允许动态地装入和卸下构件;⑤客户可以动态地发现其他服务者的接口;⑥服务者通过调用计数可以在合适的时候退出系统;⑦通过 GUID(全球唯一标号)唯一地确定任意一个构件和接口。

COM 作为构架模型主要是描述构件之间的连接和通信,它要求在内存中建立一个标准的 Vtable (虚拟函数表),客户使用标准的方法(Vtable 指针)来调用服务者的方法。这样,任何语言只要允许通过指针调用函数(如 C、C++、Visual C、Smalltalk、Ada、BASIC 等)都能用来编写 COM 构件。在 COM 构架模型中,应用程序之间通过构件接口相互通信。

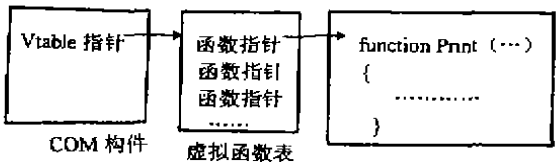

图 5 虚拟函数表 Vtable

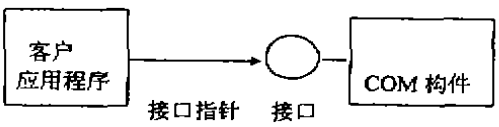

图 6 COM 构件通过接口与客户相连

构件生成

一旦分析了客户和服务者的接口规范,就要将匹配的构件绑定,再按统一的接口标准将绑定的构件组装为可执行的二进制代码,成为可重用构件,这个转化过程就是构件生成。

构件生成过程要依据构件的配置信息,首先产生一个组装计划,然后按组装计划实施组装。我们专门开发了构件生成器来产生组装计划,它是一个基于知识的专家系统,该系统包含一个规则库,规则库中存储二类规则。第一类规则定义各种生成工具(规范描述语言处理器、编译器、链接器等)的功能和限制;第二类规则定义用户的偏好,当有多个组装计划可供选择时,专家系统凭借此偏好作出决定,规则库可以根据需要(比如新的生成工具的出现)扩充。生成器分析客户和服务者的配置信息,决定接口之间的互连应选择哪些配置选项,然后进行必要的调整,最后产生组装命令来将客户和服务者相连。作为专家系统生成器的推理基础,目标运行环境的互连能力和各种编程语言同运行环境的匹配都必须被形式化地描述,然后作为规则加入规则库中。组装计划以命令文件的形式给出,执行命令文件就完成了组装过程。

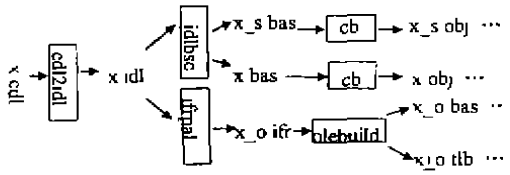

图 7 一个组装计划的例子

图 7 是对一个包含一个基于 CORBA 的 Visual Basic 服务者和一个基于 OLE 的 Delphi 客户的系统产生的组装计划。矩形框中列出的是生成工具。x.cdl 是用统一构架规范语言 CDL 描述服务者 x 的文件,x.idl 是 x 的 CORBA 的接口描述语言文件,x-o.tld 是 x 的 OLE 类型的库文件。cdl2idl 是从 cdl 到 idl 的适配器,idlbsc 是从 idl 到 Basic 的适配器,cb 是 Basic 的编译器,tfrpal 创建接口库文件,olebuild 创建 OLE 类型的库文件和 CORBA 对象的 OLE 代理。这样实施该组装计划后,用 CDL 描述的服务者 x 就转化为符合 OLE 接口标准的二进制代码构件了。

构想:通过中间件实现构件即插即用

对于一些大型的实时系统以及分布式系统,为了更换一个构件而将整个系统关闭是十分困难而且代价高昂的,最好是能在系统运行中动态地卸下一个旧的构件再插入一个新的构件。经过前面几个步骤以后,我们已经得到了适合即插即用的构件,现在我们通过中间件(仍然以 Microsoft®的 COM 为例)来动态地将新构件插入到运行中的系统中去。

在上节的客户/服务者例子中,分别将它们封装为符合 COM 接口标准的可执行构件,可以用新服务者 x 来代替系统中的旧服务者。首先,旧服务者的所有客户停止对它的调用,当它发现自己的调用计数等于零时,可以将它退出系统,从虚拟函数表

(下转第 26 页)

前，可用的保留风格主要有如下三种：

①Fixed Filter(FF)。在一个接口上所设置的每个FF保留的Filterspec仅由单一发送者组成，所设置的有效Flowspec是从该接口上接收特定发送者的全部FF保留请求中的最大值。FF Resv消息的Flowspec将单点发送给这个特定发送者的前网段，这个Flowspec是在该路由器上为这个特定发送者而设置的所有保留的最大Flowspec。

②Wildcard Filter(WF)。在一个接口上所设置的每个WF保留的Filterspec是通配符，并可与上游的任何发送者匹配。所设置的有效Flowspec是从特定接口上接收的全部WF保留请求中的最大值，每个WF Resv消息的Flowspec将单点发送给前网段上游。这个Flowspec是在该路由器上设置的所有WF保留的最大Flowspec。

③Shared Explicit(SE)。在一个接口上所设置的每个SE保留的Filterspec都含有一个特定的来自上游发送者的集合，并且是由这个接口上接收的每个SE保留请求的各个Filterspec联合设定的，所设置的有效Flowspec是从这个特定接口上接收的所有SE保留请求中的最大值。一个SE Resv消息的Filterspec将从一个接口输出单点发送给前网段上游的所有发送者联合，并且这些发送者被包含在该路由器上至少一个SE保留的Filterspec中。同样，这个SE Resv消息的Flowspec是由所有SE保留的最大Flowspec设定的。这些Filterspec至少包含一个经过该接口的前网段发送者。

SE和WF适合于会议应用，在这类应用中，某一时刻只有一个发送者是主动的，应能保存发送者音频和视频的保留请求和发送带宽，以便允许一定的超量。

结束语 RSVP通过在发送者、路由器和接收者建立一条路径提供基于不连接协议(UDP)的端到端QoS控制，以支持多媒体会议之类的多媒体应用。实现RSVP的关键技术是：

·路由器对RSVP的支持能力。这是RSVP技术的核心问题，其中路由器的QoS编码方案、资源调度策略、可提供的RSVP会话数量等都将对RSVP性能产生直接的影响。

·基于RSVP的应用开发问题。尽管RSVP规范描述了基本的RSVP API，但它并未在操作系统上实现。目前，开发RSVP应用的主要手段是WinSock v2，但支持RSVP的应用软件还很少。

由于RSVP会话与ATM虚路径概念相吻合，这意味着RSVP将在LAN-ATM体系结构中发挥更大的作用。通过RSVP可以改变目前LAN-ATM体系结构中不支持QoS的状况，使采用LAN-ATM体系结构的企业网或园区网能够充分支持多媒体应用。

参考文献

1 Beaden R. Resource Reservation Protocol(RSVP)-Version 1 Functional Specification. 1996. Available at: http://www.ietf.org/html.charters/intserv-charter.html

2 Paul P. RSVP and Integrated Services in the Internet: A Tutorial. IEEE Communications Magazion, 1997, 35(5), 100～106

3 Busse I. et al Dynamic QoS of Multimedia Application based on RTP Computer Communication, 1996, 19 49～58

4 Coulson G. et al. Supportion the Real-time requirements of Continuous Media in Open Distributed Processing. Computer Network and ISDN System, 1995, 27: 1231～1246

5 IETF home page. Available at: http://www.ietf.cnri.reston.va.us

(上接第85页)

(Vtable)中删除指向旧服务者的函数指针；装入新服务者构件，向虚拟函数表插入指向新服务者的函数指针，一旦有某个客户需要调用相关的服务，将会动态地发现这个新服务者的接口，就可以通过Vtable指针调用新服务者的新的服务了，这样客户在并不知道它所调用的服务者已经更换的情况下，继续得到服务，系统照常运行。

由于即插即用构件已被封转为二进制代码，构件之间的通信通过指针，客户通过服务者的接口得到服务，而不能访问服务者的内部实现，这种即插即用的过程在合适的配置环境下就可以进行，几乎不受编程语言、系统平台以及构件分布地址的影响。

小结 基于中间件的软件即插即用方法以中间件为基础，基本支持系统动态更新的几个过程，它具有如下特点：集成化的过程描述、形式化的模型、可执行的重用构件、配置无关性和地址透明性。我们设计了规范分析器Matchable和构件生成器Generator。随着分布式中间件和INTERNET版的中间件的发展，软件即插即用方法应该在INTERNET上得到进一步的实现。