



## [12] 发明专利说明书

专利号 ZL 200480001324.1

[45] 授权公告日 2009 年 4 月 22 日

[11] 授权公告号 CN 100481069C

[22] 申请日 2004.7.21

CN1323421A 2001.11.21

[21] 申请号 200480001324.1

US6594653B2 2003.7.15

[30] 优先权

US6584459B1 2003.6.24

[32] 2003.10.24 [33] US [31] 10/693,653

CN1325077A 2001.12.5

[86] 国际申请 PCT/US2004/023547 2004.7.21

审查员 郑守志

[87] 国际公布 WO2005/045563 英 2005.5.19

[85] 进入国家阶段日期 2005.5.20

[73] 专利权人 微软公司

地址 美国华盛顿州

[72] 发明人 A·D·米利根 C·R·里夫斯

J·B·帕尔汉姆

G·K·R·卡其瓦亚

L·A·布尔克 A·米尔斯

R·L·哈萨

[56] 参考文献

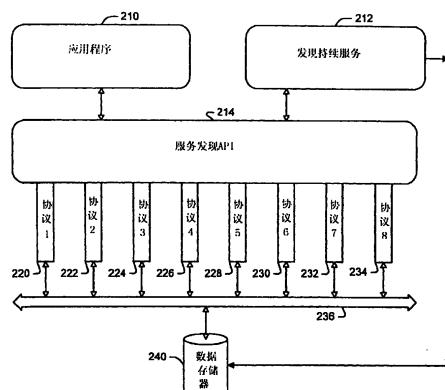
权利要求书 3 页 说明书 16 页 附图 19 页

[54] 发明名称

服务发现及发布

[57] 摘要

本文内容包括服务的发现和发布的系统和方法。应用程序描述了服务发现、发布和订阅(subscription)服务发现应用编程接口的要求。服务发现应用编程接口援用一个或多个低等级协议来满足该发现、发布和/或订阅的要求。从低级协议中得到的服务信息被编组成一个统一的数据模型，并返还给用户应用程序。此外，服务信息可被存储在一个由与服务发现 API 通讯相连的发现持续服务所管理的一个长期数据存储器中。



1. 一种用于发现在计算环境中可用的服务的方法，包括：

在一应用程序中：

    定义一发现范围；

    定义一发现过滤器； 和

    向第一应用程序接口发出一搜索请求；

在第一应用程序接口中：

    解析所述搜索请求；

    检索对应于所请求的发现范围和发现过滤器的服务信息； 和

    将所述服务信息返还给所述应用程序；

其中，所述第一应用程序接口提供了对于以在大范围的低层 API、协议、存储和网络环境上服务细节的发现和发布为目标的服务的一致的、高度的抽象。

2. 如权利要求 1 所述的方法，其中检索对应于所请求的发现范围和发现过滤器的服务信息包括执行针对至少一个低层 API 或协议的调用。

3. 如权利要求 1 所述的方法，其中检索对应于所请求的发现范围和发现过滤器的服务信息包括查询一个持续数据存储服务。

4. 如权利要求 1 所述的方法，还包括将所检索的服务信息格式化成一个一致的服务入口对象数据格式。

5. 如权利要求 2 所述的方法，还包括在持续数据存储中保存从所述至少一个低层 API 或协议接收的信息。

6. 一种用于发布在计算环境中可用的服务的方法，包括：

在一应用程序中：

    定义一服务入口对象；

    定义一发布范围；

    给所述服务分配一个唯一的键；

    分配一服务类型；

    定义服务的属性；

    定义服务的端点； 和

向第一应用程序接口发出一发布请求；

在第一应用程序接口中：

解析所述发布请求；和

执行至少一个低层 API 调用以发布所述服务；

其中，所述第一应用程序接口提供了对于以在大范围的低层 API、协议、存储和网络环境上服务细节的发现和发布为目标的服务的一致的、高度的抽象。

7. 如权利要求 6 所述的方法，还包括将所述服务信息存储在一个持续数据存储器中。

8. 一种用于删除在计算环境中已发布的服务的方法，包括：

在一应用程序中：

定义一服务入口对象；

指定一对应于所述已发布的服务的键；

定义一删除范围；和

对第一应用程序接口发出一删除请求；

在第一应用程序接口中：

解析所述删除请求；和

执行至少一个低层 API 调用来删除所述服务；

其中，所述第一应用程序接口提供了对于以在大范围的低层 API、协议、存储和网络环境上服务细节的发现和发布为目标的服务的一致的、高度的抽象。

9. 如权利要求 8 所述的方法，还包括从一个持续数据存储器中删除所述服务信息。

10. 如权利要求 8 所述的方法，还包括在一个持续数据存储器中登记所删除的服务信息。

11. 一种订阅在计算环境中的服务事件的方法，包括：

在一应用程序中：

定义一范围；

定义一过滤器；

定义一回调函数；和

对第一应用程序接口发出一订阅请求；

在第一应用程序接口中：

    解析所述订阅请求；和

    执行至少一个低层 API 调用来订阅服务事件；和

    将信息从服务事件返回给所述应用程序；

其中，所述第一应用程序接口提供了对于以在大范围的低层 API、协议、存储和网络环境上服务细节的发现和发布为目标的服务的一致的、高度的抽象。

12. 如权利要求 11 所述的方法，还包括将所检索的服务信息格式化成服务入口对象数据格式。

13. 如权利要求 12 所述的方法，还包括在持续数据存储器中保存从所述至少一个低层 API 接收的信息。

14. 一种管理有关在计算环境中可用的服务的信息的系统，包括：

    第一应用程序接口，用于接受来自应用程序的服务查询，其中所述第一应用程序接口接收第一服务查询协议中的服务查询、处理所述服务查询、且对第二协议至少发起一个相应的服务查询，其中，所述第一应用程序接口提供了对于以在大范围的低层 API、协议、存储和网络环境上服务细节的发现和发布为目标的服务的一致的、高度的抽象；

    发现持续服务，通信地连接至所述第一应用程序接口，其中所述发现持续服务从所述第一应用程序接口处接收服务信息并将所述服务信息存储在数据存储器中。

15. 如权利要求 14 所述的系统，其中所述第一应用程序接口提供接口给至少一个基于目录的协议和至少一个特别协议。

16. 如权利要求 14 所述的系统，其中所述第一应用程序接口发现在本地计算设备上可用的服务。

17. 如权利要求 14 所述的系统，其中所述第一应用程序接口发现在远程计算设备上可用的服务。

18. 如权利要求 14 所述的系统，其中所述第一应用程序接口实现一范围映射，且其中所述范围映射可由系统管理员配置。

---

## 服务发现及发布

### (1) 技术领域

本发明涉及数字计算，尤其涉及在计算设备和计算网络中的服务发现（service discovery）。

### (2) 背景技术

在计算设备和计算机网络上运行的应用程序，有时会需要使用由其他与计算设备或网络相连接的物理或逻辑设备所提供的服务。目前，应用程序使用大范围的应用编程接口（API）、协议、和用于发现、列举及描述服务和设备的对象模型，而这些服务和设备是在本地计算设备上或是在计算机网络中的多个设备上。即使所涉及的服务和设备在概念上是相似的，但用来发现、列举和描述服务和设备的机制相差很大。

例如，设想一个应用程序试图列举可用的打印机的情形。当程序运行在一个被管理的、团体环境下时，应用程序会需要使用简便目录访问协议（LDAP）来与 Microsoft ®（现用目录）目录服务存储器通信以发现注册的团体打印机、或使用 NetBT 以发现打印队列服务器、或使用蓝牙以发现私人领域的网络打印机。此外，应用程序或许必须调用设备管理 API 以发现直接与本机相连的打印机、或调用 UPnP™ API 以发现 UPnP 打印机。这些机制中的每一个都要求对特定的 API、协议和查询语义的理解。

用于发现、列举和描述服务的应用程序所需的 API 和协议的数量之大使得软件发展的任务复杂化了。

### (3) 发明内容

在说明书和权利要求书中所描述的具体实现通过提供一种将发现和发布任务简化的统一接口而解决了上述和其他问题。该统一接口允许发挥基础协议的优势，不再需要应用程序开发者去理解下层协议。该统一接口提供了一个对于服务以及以在大范围的低级 API、协议、存储和网络环境上发现和服务细节位目标的相关联操作的一致的、高度的抽象。

在一个示例实现中，提供了一种发现在计算环境中可用的服务的方法。该方

法包括：在一应用程序中，定义一发现范围，定义一发现过滤器，及向第一应用编程接口发出一搜索请求；及在该第一应用编程接口中：解析该搜索请求，检索与所请求的发现范围和发现过滤器相对应的服务信息，和将该服务信息返还给该应用程序。

在另一个示例实现中，提供了一种发布计算环境中可用的服务的方法。该方法包括：在一应用程序中：定义一服务条目对象，定义一发布范围，分配给服务一个唯一的键，分配一服务类型，定义服务的属性，定义服务的端点，及向第一应用编程接口发出一发布请求；及在第一应用编程接口中：解析该搜索请求，和执行至少一个低级 API 调用来发布该服务。

在另一个示例实现中，提供了一种删除计算环境中的一个已发布的服务的方法。该方法包括：在一应用程序中：定义服务条目对象，指定一个对应于已发布的服务的键，定义删除范围，及向第一应用编程接口发出一删除请求；及在第一应用编程接口中：解析该搜索请求，执行至少一个低级 API 调用来删除服务。

在另一个示例实现中，提供了一种订阅计算环境中服务事件的方法。该方法包括：在一应用程序中：定义一范围，定义一过滤器，定义一回调函数（callback function），及向第一应用编程接口发出一订阅请求；及在第一应用编程接口中：解析该搜索请求，执行至少一个低级 API 调用来订阅服务事件，及将服务事件的信息返还给该应用程序。

在另一个示例实现中，提供了一种管理有关计算环境中可用的服务的信息的系统。该系统包括：第一应用编程接口，其配置成接受来自应用程序的服务查询，其中第一应用编程接口以第一服务查询协议接收服务查询、处理该服务查询、和向第二协议发出至少一个相应服务查询；与该第一应用编程接口通信地相连的发现持续服务，其中发现持续服务从第一应用编程接口处接收服务信息，并将服务信息存储在一个数据存储器中。

#### (4) 附图说明

图 1 是一个示例计算设备的示意性说明；

图 2 是一个说明示例软件体系结构的结构图；

图 3 是阐明服务发现操作的流程图；

图 4 是阐明服务发布操作的流程图；

图 5 是阐明服务删除操作的流程图；

图 6 是阐明订阅服务事件操作的流程图；

图 7 是阐明具体范围和抽象范围之间关系的结构图；

图 8 是阐明如何使用 C# 编程语言利用现用目录协议上的 SimpleFilter 对象来定位那些每分钟能打印 50 页的彩色打印机的伪代码；

图 9 是阐明如何使用 C# 编程语言来定位网络服务的伪代码；

图 10 是阐明如何使用 C# 编程语言利用 SimpleFilter 对象和 UDDI 协议来查找支持特定的 tModel 接口的伪代码；

图 11 是阐明利用 SimpleFilter 对象和 UDDI 协议来查找支持特定的 tModel 接口 Visual Basic.NET 的使用方法的伪代码；

图 12 是阐明利用现用目录支持下的 RichFilter 来定位一个名如 Office Printer 的打印机的 C# 编程语言的使用方法的伪代码；

图 13 是阐明利用现用目录支持下的 RichFilter 来定位一个名如 Office Printer 的打印机的 Visual Basic.NET 的使用方法的伪代码；

图 14 是阐明利用 SSDP 协议来发布一个由特定的唯一的标识符所确定的特定类型的服务的 C# 编程语言的使用方法的伪代码；

图 15 是阐明利用 SSDP 协议来发布一个由特定的唯一的标识符所确定的特定类型的服务的 Visual Basic.NET 的使用方法的伪代码；

图 16 是阐明来删除一个来自 SSDP 协议的服务的 C# 编程语言的使用方法的伪代码；

图 17 是阐明来删除一个来自 SSDP 协议的服务的 Visual Basic.NET 的使用方法的伪代码；

图 18 是阐明利用 SimpleFilter 来注册那些使用 SSDP 协议的特定类型的事件的 C# 编程语言的使用方法的伪代码。这个注册了的回调函数将为每个与过滤器相匹配的事件调用，且相应的 ServiceEntry 对象将被提供给该处理器；和

图 19 是阐明利用 SimpleFilter 来注册那些使用 SSDP 协议的特定类型的事件的 Visual Basic.NET 的使用方法的伪代码。

## (5) 具体实施方式

这里描述的是用于服务发现和发布的示例性方法和软件体系结构。这里描

述的方法可被具体化为计算机可读媒体上的逻辑指令。当运行在处理器上时，这些逻辑指令使得通用计算设备被编程为执行上述方法的专用机。当处理器由逻辑指令配置成执行这里所表述的方法时，它就组成适于执行上述方法的结构。

#### 示例性运行环境

图 1 是可被用来实现根据所述具体实施方式的一个或多个计算设备的示例性计算设备 130 的示意性说明。计算设备 130 可被用来实现根据所述具体实施方式的各种不同实现。

计算设备 130 包括一个或多个处理器或处理单元 132、系统存储器 134、和耦合包括系统存储器 134 和处理器 132 在内的多个系统组件的总线 136。总线 136 表示若干类型的总线结构中任意的一种或多种，包括存储器总线或存储器控制器、外部总线、图形加速端口、以及采用多种总线结构中任意一种的处理器或本地总线。系统存储器 134 包括只读存储器 (ROM) 138 和随机存取存储器 (RAM) 140。基本输入输出系统 (BIOS) 142 包含帮助在计算设备 130 内的组件之间传输信息的基本例行程序，诸如在启动时，BIOS 被存储在 ROM 138 中。

计算设备 130 还包括用来从硬盘 (未示出) 中读取数据和向硬盘写入数据的硬盘驱动器 144、用来从可移动式磁盘 148 (未示出) 中读取数据和向可移动式磁盘 148 写入数据的磁盘驱动器 146、和用来从可移动光盘 152 如 CD ROM 或其他光媒中读取数据和向可移动光盘 152 写入数据的光盘驱动器 150。硬盘驱动器 144、磁盘驱动器 146 和光盘驱动器 150 通过 SCSI 接口 154 或其他适合的接口与总线 136 相连接。这些驱动器和及其相关联计算机可读媒体给计算设备 130 提供对计算机可读指令、数据结构、程序模块和其他数据的非易失性存储。尽管这里所述的示例性运行环境使用了硬盘、可移动磁盘 148 和可移动光盘 152，但本领域的技术人员应认识到其他类型的能够存储计算机可存取数据的计算机可读媒体，如磁带、闪存卡、数字影碟、随机存取存储器 (RAM)，只读存储器 (ROM) 等都可被用在该示例性运行环境中。

众多程序模块可存储在硬盘 144、磁盘 148、光盘 152、ROM 138，和 RAM 140 上，包括操作系统 158、一个或多个应用程序 160、其它程序模块 162 和程序数据 164。用户可通过输入设备，如键盘 166 和定位设备 168 向计算设备 130

输入命令和信息。其它可用的输入设备（未示出）可以是麦克风、操纵杆、游戏手柄、圆盘式卫星电视天线、扫描仪等。这些或其它输入设备通过耦合至总线 136 上的接口 170 与处理单元 132 相连接。通过诸如视频适配器 174 的接口，监视 172 或其它显示设备同样与总线 136 相连接。除监视器之外，个人计算机通常包括其他外部输出设备（未示出），诸如扬声器和打印机。

计算设备 130 通常运行在利用与一个或多个远程计算机如远程计算机 176 的逻辑连接的网络环境下。远程计算机 176 可以是另一台个人计算机、服务器、路由器、网络 PC、对等设备或其它公共网络节点，而且远程计算机 176 通常包括上述与计算设备 130 相关所描述的元件中的多个或者全部，尽管在图 1 中只示出了一个存储器 178。图 1 中所描述的逻辑连接包括局域网（LAN）180 和广域网（WAN）182。这种网络环境在办公室、企业范围计算机网络、企业内部互联网和因特网中都很常见。

当在 LAN 网络环境中使用时，计算设备 130 通过网络接口或适配器 184 连接至本地网络 180 接。当在 WAN 网络环境中使用时，计算设备 130 通常包括调制解调器 186 或其它手段，用于在广域网 182 如因特网上建立通信。调制解调器 186 可以是内置的或外置的，它通过串行接口 156 连接至总线 136。在网络环境中，与计算设备 130 相关描述的程序模块或其中的部分，可以存储在远程存储器设备中。将会认识到，所示出的网络连接是示例性的，其它在计算机间建立通信连接的方法也可以使用。

一般，计算设备 130 中的数据处理器通过在不同时期存储在计算机的各种计算机可读存储媒体中的指令来编程。程序和操作系统通常是分布在在例如软盘或 CD-ROM 中的。它们被从软盘或光盘上安装或装载至计算机辅助存储器。当执行时，它们至少会被部分地装入计算机主要电子存储器中。这里所描述的发明包括这些或其它类型的计算机可读存储媒体，如果这些媒体包含以下用于实现结合微处理器和其它数据处理器所描述步骤的指令或程序。当计算机被根据下述方法和技术编程时，本发明还包括计算机本身。

#### 示例性软件体系结构概述

图 2 是在可以驻留在图 1 的系统存储器 134 中的用于服务发现的示例性软件体系结构 200 的结构图。在这种实现中，系统存储器 134 可包含多个应用程

序 210。在网络化环境中，应用程序可以是客户程序的作用，而在 PC 环境中应用程序可以是单机程序。应用程序的特别性质是无关紧要的。

应用程序 210 调用服务发现 API 214 以发现在计算机环境中可用的服务。服务发现 API 214 提供高级文法来用于表达发现查询。这种文法可以用 Opath ——一种用于表达发现查询的自然查询语言来实现。这种高级文法向软件开发者提供一种更概念化的机制来表达开发者所寻找的服务，而不是要求由协议 220-234 所要求的颗粒度更高的和协议特定的表达。开发者可以用高级文法来构建一个查询，其可随后或者被转给一个特定的协议集，称之为众多“具体的范围”，或者给一个“抽象的范围”，其为预定义的或配置的具体范围的集合，。除了支持服务发现，系统还支持服务发布/删除和对事件的监视。

服务发现 API 214 进而调用在图中用 Protocol 1 220 到 Protocol 8 234 所表示的一个或多个基础协议 (underlying protocol)。基础协议的确切数目是不重要的。协议 220-234 中的几个是目录支持协议，比如 LDAP、通用描述 (Universal Description)、发现和集成 (UDDI)、和域名系统 (DNS) 服务器 ()。其它的协议可以是特殊的协议，诸如 Bluetooth (蓝牙)、UpnP、和 NetBT。协议 220-234 中的一个或多个使用通信连接 236 来与计算环境中可用的其它组件或服务进行通信。

响应于发现请求，服务发现 API 返回一组表示在本地机器或网络上发现的匹配服务的 ServiceEntry 对象。ServiceEntry 对象是一个通用数据结构，它能够代表许多由系统支持的协议所返回的相关细节。每个 ServiceEntry 对象对应于一个单一的服务实例。在一种实现中，ServiceEntry 对象提供描述性的和辨识性的属性，包括：(1) 服务名称；(2) 服务描述；(3) 端点，其通常包含服务的网络地址；(4) 键值，其标识该服务实例；(5) 属性，例如，服务或设备特性的名称-值对可扩展表；和(6) 供应商，例如，一个标识提供该服务的实体的标识符。

发现持续服务 212 与服务发现 API 214 通信。其中，发现持续服务 212 登记在特别协议上的声明事件。当声明事件被检测到时，即通知发现持续服务，发现持续服务将服务声明的相关信息复制到数据存储 240 中的存储单元。在存储单元中存储服务细节使得目前不可用的服务也能被发现。例如，即使一个打

印机现在被关闭电源，打印机的有关细节可被登记在存储单元内，且可被发现。除此之外，服务查询不局限于与服务通信的协议。此外，查询存储单元的性能可远远好于处理一个广泛的网络发现查询。

#### 示例性操作

在示例性实现中，服务发现 API 214 提供用于服务发现、服务发布和订阅服务事件通知的方法。图 3 是阐明服务发现的操作 300 的流程图。在操作 310 处，应用程序定义了一个范围，在操作 315 处应用程序定义了一个过滤器，在操作 320 处应用程序发出搜索请求。服务发现 API 214 收到了搜索请求，在操作 325 处，服务发现 API 214 解析这个搜索请求。在可任选操作 330 处，服务发现 API 214 确定这个搜索请求是否可利用存储在发现持续服务 212 中的信息解决。在一种实现中，由发现持续服务 212 所管理的信息包括一个生存时间指示器，其指定在发现持续服务 212 中的信息的生存期限。取决于控制和配置，服务发现 API 214 会查询发现持续服务 212 来确定这个发现请求能否利用发现持续服务 212 在数据存储器 240 上管理的那些信息去满足。如果该发现请求可利用发现持续服务 212 解决，那么控制传给操作 350，且从发现持续服务 212 中检索到的服务条目对象被返回给应用程序。

相反，如果发现请求是利用发现持续服务 212 没有解决或不能够解决的，那么控制传给操作 335，服务发现 API 214 执行为实现该发现请求所要求的低级 API（一个或多个）调用。在操作 340 处，从低级 API 调用中返回的服务信息被格式化成服务条目对象，在可任选操作 345 处，服务条目对象被传给能够在数据存储器 240 上储服务条目对象的发现保存服务。在可任选操作 347 处，可以执行进一步的对服务条目结果的处理和过滤，诸如冗余检测和去除。在操作 350 处，服务条目对象被返回给应用程序进行在操作 355 处的进一步处理。由应用程序执行的进一步处理的精确细节是不重要的。

图 4 是阐述服务发布操作的流程图。在操作 410 处，应用程序为服务发布定义服务条目对象。在操作 415 处，应用程序给服务发布定义范围。在操作 420 处，应用程序给服务发布分配一个唯一的键，在操作 425 处，应用程序给服务发布分配一个服务类型。在操作 430 处，应用程序给服务发布定义端点，在操作 432 处，应用程序为服务发布定义属性，在操作 435 处，应用程序生成一个

发布请求。根据将要发布的信息的细节以及将会使用的低级 API，所执行的步骤可有所变化。

服务发现 API 214 接收到发布请求，并在操作 440 处，解析该发布请求。在操作 450 处，服务发现 API 214 执行低级 API 调用来执行服务发布请求。在可任选操作 455 处，服务发布被存储在发现持续服务 212 中。

服务发现 API 214 的服务发布工具也可用来删除一个已发布的服务。图 5 是阐述服务删除操作的流程图。在操作 510 处，应用程序定义服务发布的服务条目对象。在操作 515 处，应用程序给服务指定一个唯一的键。在操作 520 处，应用程序给服务删除定义范围。在操作 530 处，应用程序生成一个服务删除请求。

服务发现 API 214 接收删除请求，并且在操作 540 处，解析该删除请求。在操作 550 处，服务发现 API 214 执行低级 API 调用来执行服务删除请求。在可任选操作 555 处，服务发布被从发现持续服务 212 中删除。

服务发现 API 214 也可被用来允许向应用程序通知服务事件，如一项新的服务或特定类型的设备的到来和离去。图 6 是阐述订阅服务事件操作 600 的流程图。在操作 610 处，应用程序定义一个向监视器指定特定低级协议的范围。在操作 615 处，应用程序定义一个指定事件类型的过滤器。在操作 620 处，应用程序定义一个回调函数，它将在匹配事件发生时接收 ServiceEntry 细节。在操作 625 处，应用程序生成一个订阅请求，其被传给服务发现 API 214。

服务发现请求 API 214 接收该订阅请求，并且在操作 630 处，解析该订阅请求，在操作 635 处，服务发现请求执行实现订阅服务所要求的低级协议调用。当服务事件发生时，低级协议会向服务发现 API 提供事件的通知。在操作 640 处，事件的通知被格式化成服务条目对象。在可任选操作 645 处，服务条目对象可被存储在发现持续服务 212 中，在操作 650 中，该服务条目对象利用先前指定的回调函数返回给应用程序。在操作 655 处，应用程序对服务条目对象进行进一步的处理。应用程序的进一步处理的精确细节是不重要的。

系统的组件和操作在以下做更详细的说明。

## 过滤器

过滤器是借以评估服务描述的一组规则，得到的结果是真（即服务描述与过滤器相匹配）或假（即服务描述与过滤器不匹配）。一个过滤器可以表达成一个指定具体属性的简单过滤器，或表达成一个利用更有表达力的文法的复杂过滤器。不论是表达为简单过滤器还是复杂过滤器，查询能够通过不止一个的协议遵从所用协议的能力来指定和执行而不需修改。服务发现请求 API 214 将高层查询重新表达成用于基础低级协议的正确格式。例如，服务发现请求 API 214 可以接收一个用于特定服务类型的查询，并且利用为现用目录服务的 LDAP 和为 UDDI Web 服务注册服务的 UDDI 协议来表达和评估该项查询。不要求应用程序开发者直接处理个别的协议。

在一示例性的实现中，服务发现请求 API 214 要求发现模块支持简单过滤器，提供给出标准的精确的匹配语义、和含有用 OPath 文法表达的查询的复杂过滤器。将会理解，各自也会支持附加的“天生的”过滤器类型。不同的发现模块可能拥有协议专用的天生的过滤类型，如，UPnP 可使用 XPath 过滤器，现用目录可以天生地使用 LDAP 过滤器，UDDI 可以天生的使用 UDDI 过滤器。

模块上的 OPath 过滤器功能的底层进一步将应用程序与基础发现协议隔绝。过滤器类用模块间共享的方式展露了其他的解析和解释过滤器的方法。

一个简单过滤器通过指定服务类型、服务接口、和/或属性而规定查询的表达。这些设置的任意组合可在在一个搜索查询中提供，只有在每个标准都精确地匹配时，服务才会被包括在所得服务条目集之中。

服务类型可以被实现为一个指定必须与服务实例相匹配的类型的串。一个常用的服务类型集在服务发现请求 API 214 中被预定义。当有在协议和存储器中的键实体被标识到时，这个集合可以被扩展。例如，对于现用目录中的打印机，其可指定为：filter.ServiceType = CommonServiceTypes.Printer。

服务接口可被实现为一个指定用于服务必须匹配的接口的标识符的串集合。作为一个例子，对于在 UDDI 中的 web 服务，下述 tModel 标识符可以被指定：

```
filter.ServiceInterfaces.Add("uuid:ac104dcc-d623-452f-88a7-f8acd94d9b2b");
filter.ServiceInterfaces.Add("uuid:4d2ac1ca-e234-142f-e217-4d9b2f8acd9b")
```

属性可在指定服务必须匹配的服务特性的属性字典中实现。作为一个例子，对于在现用目录中的打印机，下述属性可被指定：filter.Properties.Add("printcolor", "TRUE"); filter.Properties.Add("pagesperminute", "50")

通过设定 Query 串属性，一个复杂过滤器提供一种利用例如 OPath 文法表达显著复杂的查询语义的机制。作为一个例子，对于 UDDI 中的 web 服务，Query 串会指定所需的名称和所需的支持的接口：filter.Query="WebService [ name='Fabrikam' and ServiceInterface='uuid:ac104dcc-d623-452f-88a7-f8acd94d9b2b']"

作为一个在没有 A4 纸可用时查找现用目录中有能力每分钟打印 25 页纸以上的打印机的更有表达力的例子：

```
filter.Query="Printer[printPagesPerMinuter>20 + 5 and not  
( printmediaReady='A4')]"。
```

由于基础协议和存储器的能力很不相同，从基本的 NetBT 到复杂的现用目录查询语义，使用具有更强表达力的 OPath 结构的能力将取决于所选择的范围（协议）。

## 范围

范围标识一个可以被搜索的查询域，通常是粗略的且由网络位置或管理权限所及给出。发现查询被发往一个或多个范围，且查询结果包括这些范围内的服务的子集，即，查询结果是所有在此范围内匹配给定过滤器的服务的子集。示例性的范围包括工作组、本地机器和域。

服务发现 API 214 提供具体的范围和抽象的范围。具体的范围以三个部分来指定查询域：一个标识具体协议的协议（Protocol）标识符，例如映射到单个发现模块，诸如 ConcreteScope.NetBtProtocol. 或 ConcreteScope.ADProtocol；一个地址（Address）（可任选）标识符，其指定一个在此范围上向其指引操作的服务器，诸如用于企业内部互联网 UDDI 服务器的“<http://intrauddi/uddi/inquire.asmx>”；和一个路径标识符（可任选），其标识模块名空间的分割，诸如一个 LDAP 搜索基，其可被设置成“CN=joedev, CN=Computer, DC=corp, DC=fabrikam, DC=com”，或者一个 UPnPv2

范围名。

服务发现请求 API 214 将具体范围传给模块。服务发现请求 API 214 不排除模块执行其它在具体范围上的间接操作，比如以有线方式传输具体范围给另一台机器和将具体范围传给该另一台机器上的相应的 API。

抽象范围是一个或多个具体范围的别称，也可以是其它抽象范围的别称。抽象范围提供一种在逻辑预定义的或配置了的具体范围集上定标查询的机制。这提供了允许开发者来定标例如，“enterprise” 范围的附加抽象，不需要清楚的协议、地址和特定目录服务器的连接细节。

将抽象范围映射到具体范围是全机器范围的，且是可配置的。例如，抽象范围 AbstractScope. Enterprise 可以映射以包括表 1 中两个具体范围。

Protocol=ConcreteScope. ADProtocol address="ldap://dev.corp.fabrikam.com" path=null
Protocol=ConcreteScope. UddiProtocol address="http://uddi.fabrikam.com/inquire.asmx" path=null

表 1

图 7 是阐明具体范围和抽象范围之间示例关系的结构图。具体范围 730-750 给出在其上评估查询的域的规格。具体范围 730-750 包括协议标识细节，及所要求的要用的存储器或服务器的具体情况，具有进一步在该存储器或服务器内进行范围确定的潜能。在服务发现 API 214 中，这些分别在协议、地址和路径属性中指定。

抽象范围 710-725 在具体范围之上提供更高层次的分级抽象。抽象范围被设置成包括组成它们的具体范围或抽象范围。这一范围映射系统管理员可用的，他们能够精确地配置如何解析例如 AbstractScope. EnterpriseScope。

具体范围和抽象范围都能够被服务发现 API 214 的用户所使用。在提供了抽象范围的场合，服务发现 API 214 会将其沿等级向下分解为一定数目的具体范围。

抽象范围允许应用程序 210 的程序员能在较高层次上工作，并且在代码中可以包含诸如“AbstractScope. Enterprise”的范围标识术语。这样，例如，程序员就不必将特定 UDDI 服务器的具体情况硬代码写入他的代码中。这种抽象提供了更

大是代码重用性和可移植性。同一代码段可以不经修改和重新编译地用于各种各样的企业环境中。只有抽象范围的配置会在环境改变时改变。

在抽象范围到具体范围的映射中可以有多重等级。在图 7 中，AbstractScope.LocalMachine 不能映射进 AbstractScope.All，尽管它的所有组件都包括了。

在一个示例性实现中，范围映射配置能由系统管理员通过组政策进行操控以控制企业中服务发现 API 214 的使用。作为例子，一个管理员能定义一个或多个可用于企业计算环境中或部分的企业计算环境中的抽象范围。这使得系统管理员能够调控应用程序的发现和对资源的使用。

#### ServiceEntry 服务条目结果

应用程序员能够选择合适的范围和过滤器表达，这些可以设置成服务发现者对象的属性。于是应用程序就可用 FindOne 或 FindAll 方法来执行发现请求。FindAll 方法返回所有匹配所提供的标准的服务，而 FindOne 方法只返回单个匹配的服务。这些方法可以应用同步或异步的调用模式来执行。

假定在指定的范围内有匹配所提供的过滤器的服务，FindOne 或 FindAll 方法会返回一个或一组服务条目对象。服务条目对象是对各种基础协议可提供的服务的表示法的抽象。每个服务条目对象对应于一个服务的单个实例，照此提供包括在表 2 中所述那些在内的描述性和标识性的属性。

属性	注释
名称	标识服务实例
描述	服务实例的描述
端点	服务实例可在其处服务的端点的集合
键	服务实例的标识键
范围	实体从其中发现或向其中发布的范围
证书	指定在发布该服务时将用到的证书
提供商	引用服务提供商（容器），如果存在的话
终止时间	服务条目根据生存时间将终止的时刻

表 2

提供了一公共空白 (public void) Save () 函数来创建或更新在范围集

之中所指定的范围内的服务条目表示。

公共空白 Delete () 方法将 ServiceEntry 对象从在范围属性中所指定的范围内去除。如果服务尚未发布，则造成异常。

#### 伪代码

图 8-19 所示是用于各种服务发现、发布和订阅函数的伪代码。

图 8 是阐述利用现用目录协议上的 SimpleFilter 对象如何使用 C# 编程语言来查找每分钟打印 50 页的彩色打印机的伪代码。

图 9 是阐述利用 UDDI 协议上的 RichFilter 对象如何使用 C# 编程语言来查找实现 uddi-org: inquiry\_v2 接口且名称为 Fabrikam 的 web 服务的伪代码。

图 10 是阐述利用 UDDI 协议上和 SimpleFilter 对象使用 C# 编程语言来查找支持特定的 tModel 接口的服务的伪代码。

图 11 是阐述利用 UDDI 协议上和 SimpleFilter 对象使用 Visual Basic. NET 来查找支持特定的 tModel 接口的服务的伪代码。

图 12 是阐述利用现用目录下的 RichFilter 使用 C# 编程语言来查找名字如 Office Printer 的打印机的伪代码。

图 13 是阐述利用现用目录下的 RichFilter 使用 Visual Basic. NET 来查找名字如 Office Printer 的打印机的伪代码。

图 14 是阐述利用 SSDP 协议使用 C# 编程语言来发布由特定的唯一的标识符所标识的特定类型的服务的伪代码。

图 15 是阐述利用 SSDP 协议使用 Visual Basic. NET 来发布由特定的唯一的标识符所标识的特定类型的服务的伪代码。

图 16 是阐述使用 C# 编程语言来删除一个来自 SSDP 协议的服务的伪代码。

图 17 是阐述使用 Visual Basic. NET 来删除一个来自 SSDP 协议的服务的伪代码。

图 18 是阐述利用 SimpleFilter 使用 C# 编程语言来登记使用 SSDP 协议的特定类型事件的伪代码。已登记的回调函数会为每一个匹配过滤器的事件而调用，且相应的 ServiceEntry 对象会被提供给该处理器（handler）。

图 19 是阐述利用 SimpleFilter 使用 Visual Basic. NET 来登记使用 SSDP

协议的特定类型事件的伪代码。

### 示例性的 OPath 语法

表 3 给各种发现函数提供了示例性的 OPath 语法。

Opath	说明
Printer	寻找所有打印机和打印队列。
Printer[ name = 'Upstairs Printer' ]	寻找所有名为 Upstairs Printer 的打印机。
Printer[ printPagesPerMinute > 20 + 5 and not( printmediaReady=' A4' ) ]	寻找所有能够每分钟打印超过 25 页且没有可用的 A4 纸的打印机。
Printer[ Properties.name like 'Pri' and (printPagesPerMinute > 10 or printMediaReady=' letter' ) ]	寻找所有名字以 Pri 开头的、且每分钟打印超过 10 页或有可用的信纸的打印机。
Print[ supportsColor = true && (printName like 'Home' or name like 'Work') ]	寻找所有支持彩色打印且名字开头为 Home 或 Work 的打印机。
Service[ serviceInterface = ServiceConnectionPoint ]	寻找所有是 ServiceConnectionPoint 对象的服务。
Service[ (serviceType = 'Printer' or ServiceType = 'Computer') and name like 'Work' ]	寻找所有名如 Work 的是打印机或者 是计算机的服务。
Computer[ operatingSystemVersion like '%3790%' ]	寻找所有运行其操作系统版本号中有 3790 的计算机。
Computer[ operatingSystem = 'Windows Server 2003' ]	寻找所有运行特定操作系统的计算机。OperatingSystem 属性不包括在全局目录中。

表 3

表 4 包含可以在 UDDI 协议上使用的 OPath 语法的示例。

OPath	说明
WebService[ name = 'Fabrikam' ]	寻找所有名为 Fabrikam 的 web 服务。

WebService[ name = 'UDDI%' && ServiceInterface = 'uuid:ac104dcc-d623-452f-88a7-f8acd94d9b2b' ]	寻找所有名称开头为 UDDI 且支持所标识的接口（即 tModel uddi-org: inquiry_v2）的 web 服务。
--	--

表 4

表 5 包含了可以在 NetBT 协议上使用的 OPath 语法的示例。

OPath	说明
Workstation	寻找所有的工作站。
Service[ ServiceType = 'PrintQueueServer' ]	寻找所有的类型为 PrintQueueServer 的服务。
Computer[ serviceInterface = 'DomainController' and ServiceInterface = 'TerminalServer' ]	寻找所有作为终端服务器运行的域控制器计算机。

表 5

### 发现持续服务

如上文概要描述，发现持续服务 212 为服务信息管理一个服务信息的持久数据存储器。周期性地，或当某些预定事件发生时，比如启动，发现持续服务会登记以接收特别的设备/服务通知。作为一个例子，当一个新的 UPnP 设备被引入时，它会生成一个由 UPnP 协议模块处理的设备通知。该 UPnP 协议模块会将此事件（设备及其服务）通过服务发现 API 214 传给发现持续服务。

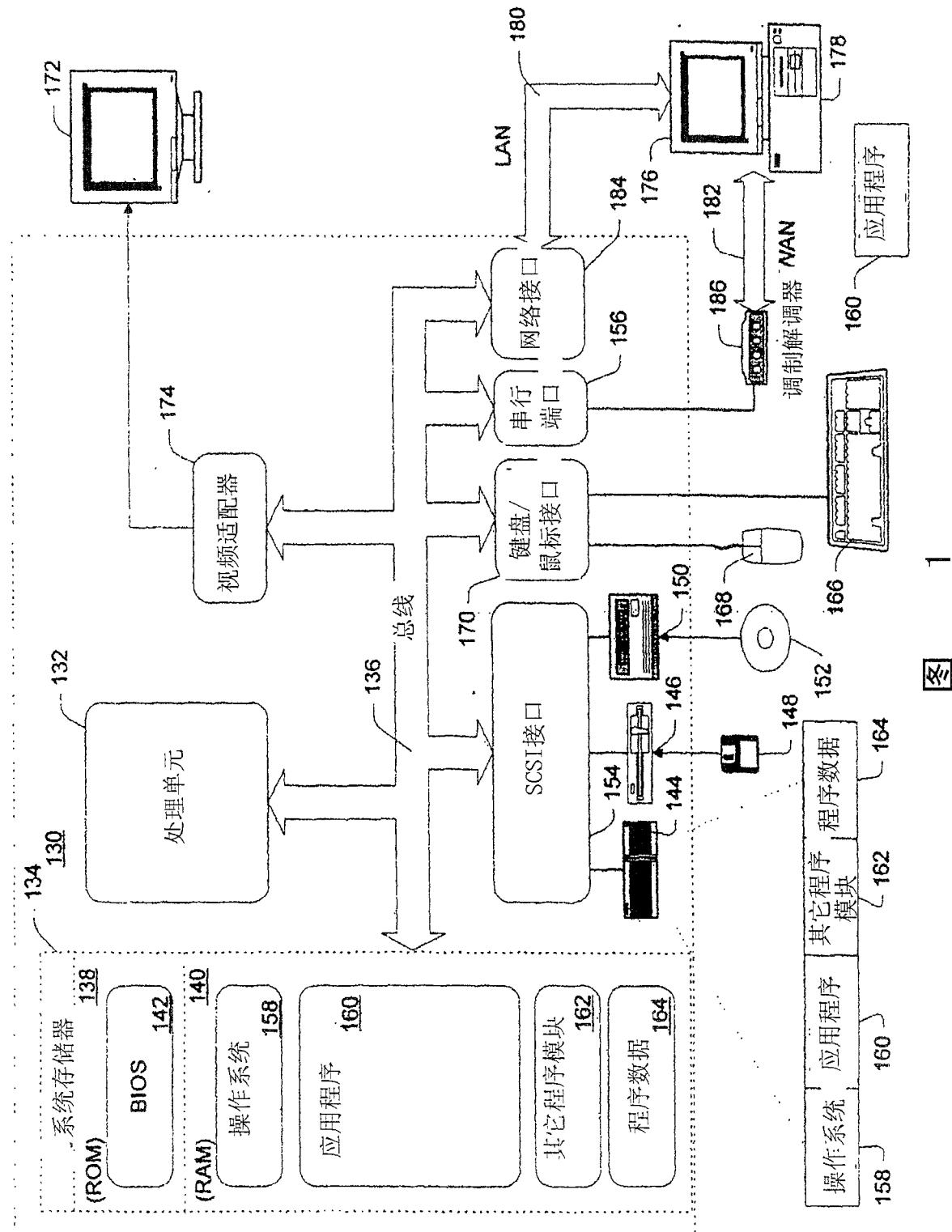
利用其持久数据存储器，发现持续服务随后确定这是一个新的设备/服务还是一个回归的设备/服务。如果是新的设备/服务，那么此设备及其服务的细节会被登记在持久数据存储器中。当另一个服务发现 API 214 的消费者试图查找服务时，服务发现 API 214 将能够返回特别的设备/服务的服务，即使该设备当前不可用。对于上述例子，在设备/服务当前可用的情况下，根据所指定的范围，UPnP 协议模块和持久数据存储器模块都会返回该设备的结果。除了 UpnP 之外，这种功能性也适用于其他特别的发现机制。

如此，发现持续服务 212、服务发现 API 214 和本地数据库存储器 240 给用于设备和服务发现的各种低级协议提供了一个抽象层。这个附加的抽象层建立了一个应用程序员在开发应用程序时可使用的通用的、改进了的搜索语义。

此外，发现抽象服务 212、服务发现 API 214 和本地数据库存储器 240 给本地机器、家庭网络、企业网络和因特网上的服务和设备提供了一个统一的发现模型。如此，通过编写一个统一的 API，应用程序开发员能够在范围广泛的位置中发现服务。

### 结论

- 尽管所描述的内容是用一种针对结构特征和/或方法性操作的语言描述的，但应该认识到，在所附权利要求中定义的主题内容不必局限于所描述的具体特征和操作。相反，这些具体特征和方法是作为实现权利要求中的本发明优选形式来揭示的。



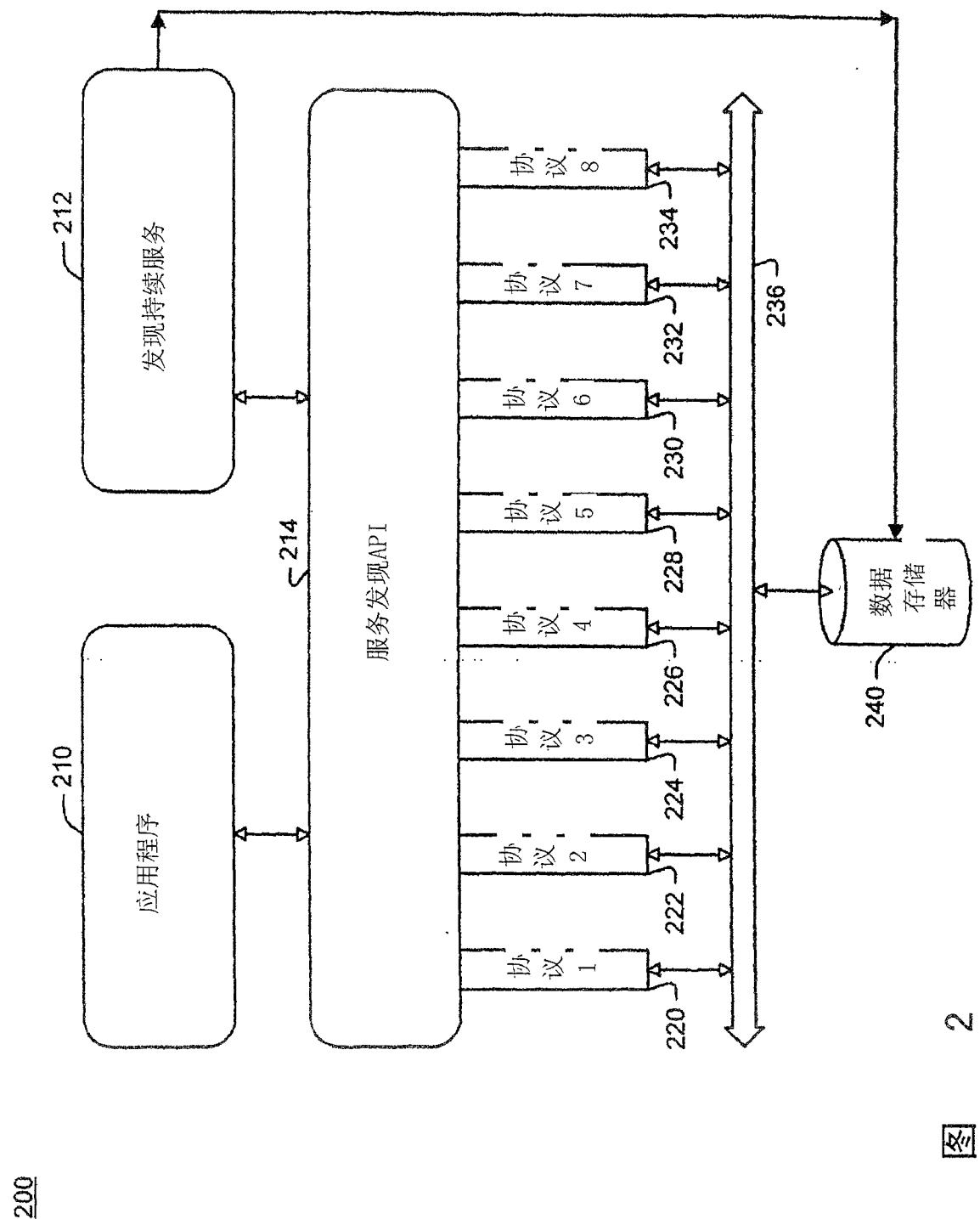


图2

200

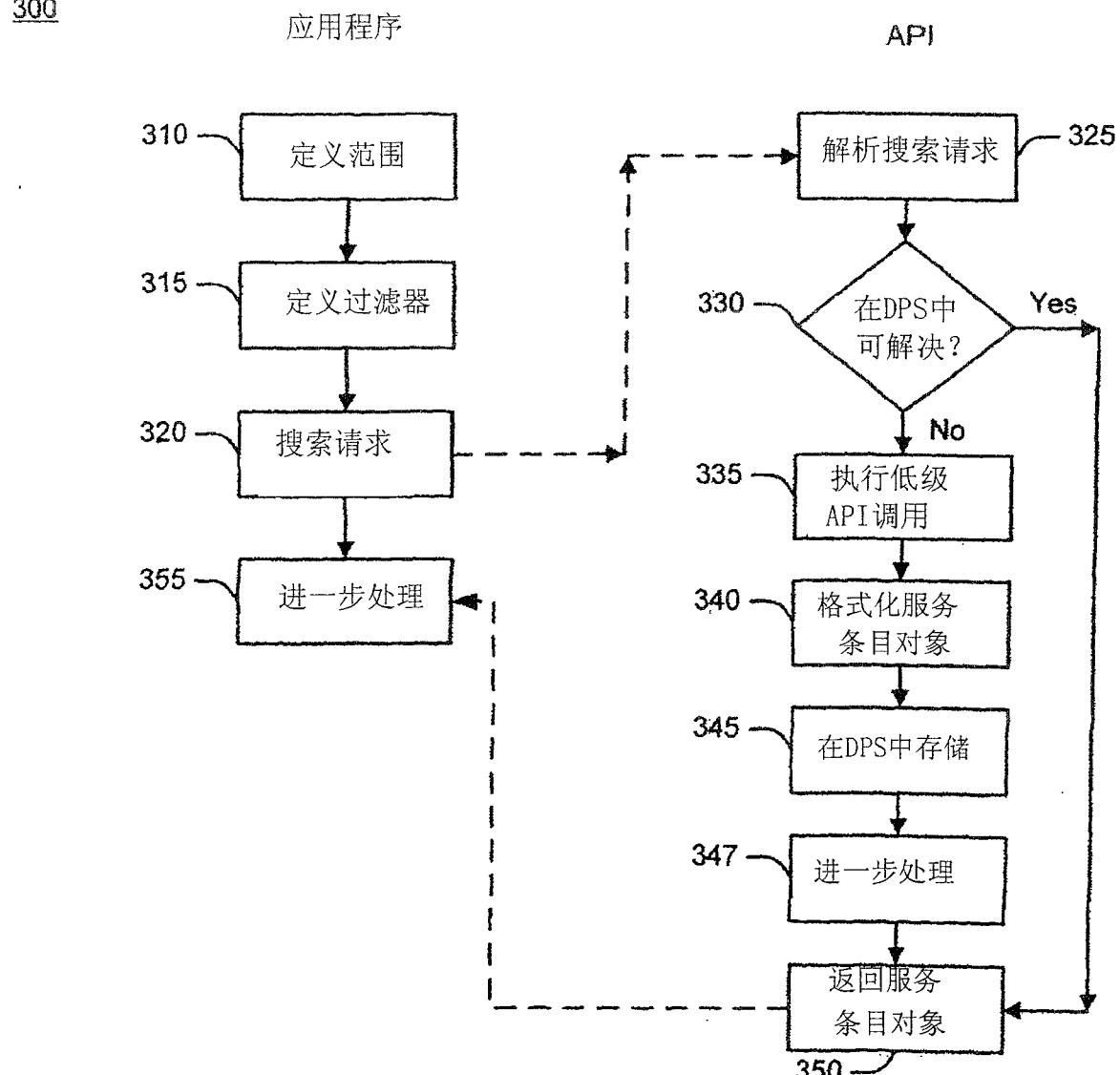
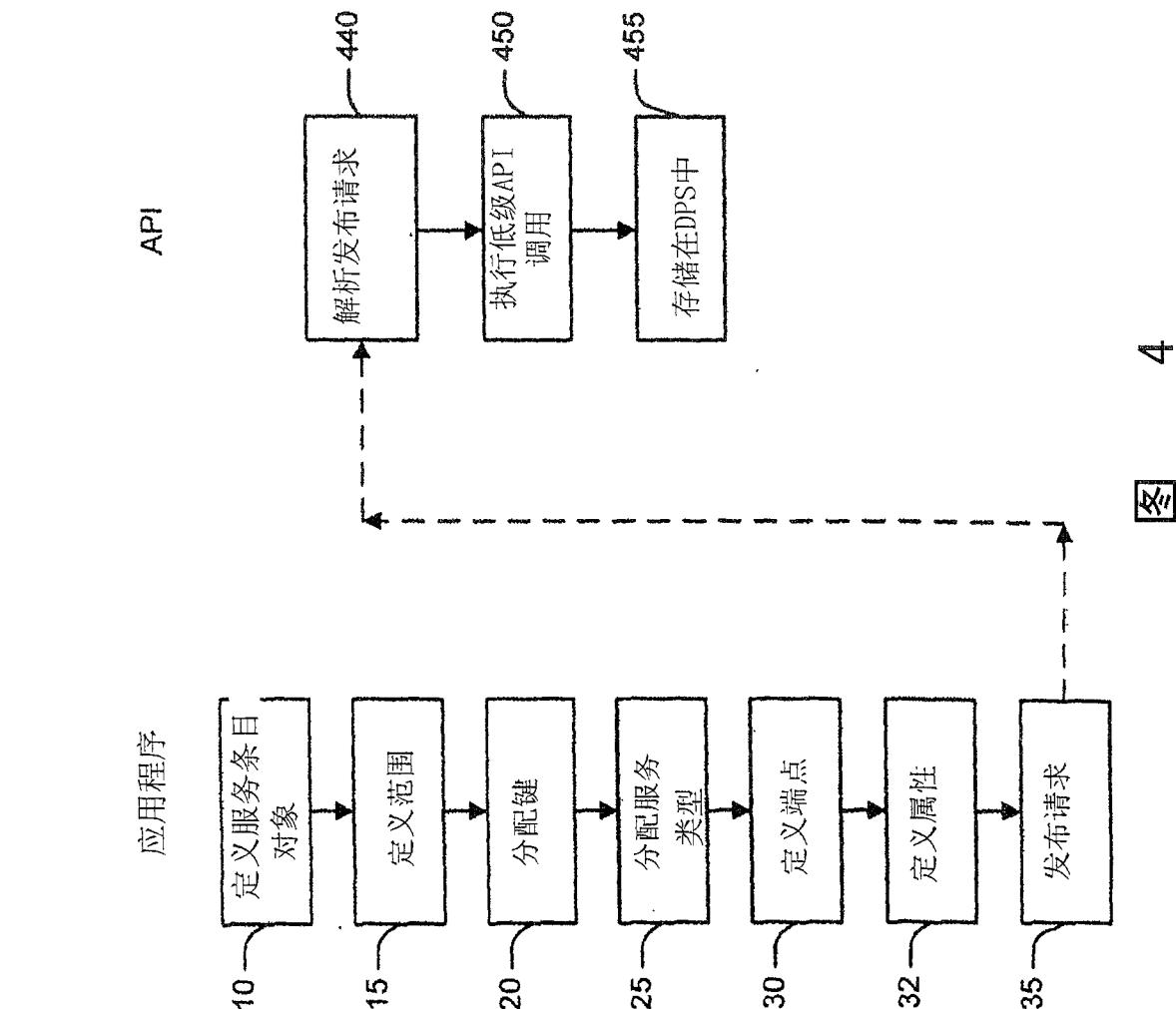
300

图 3



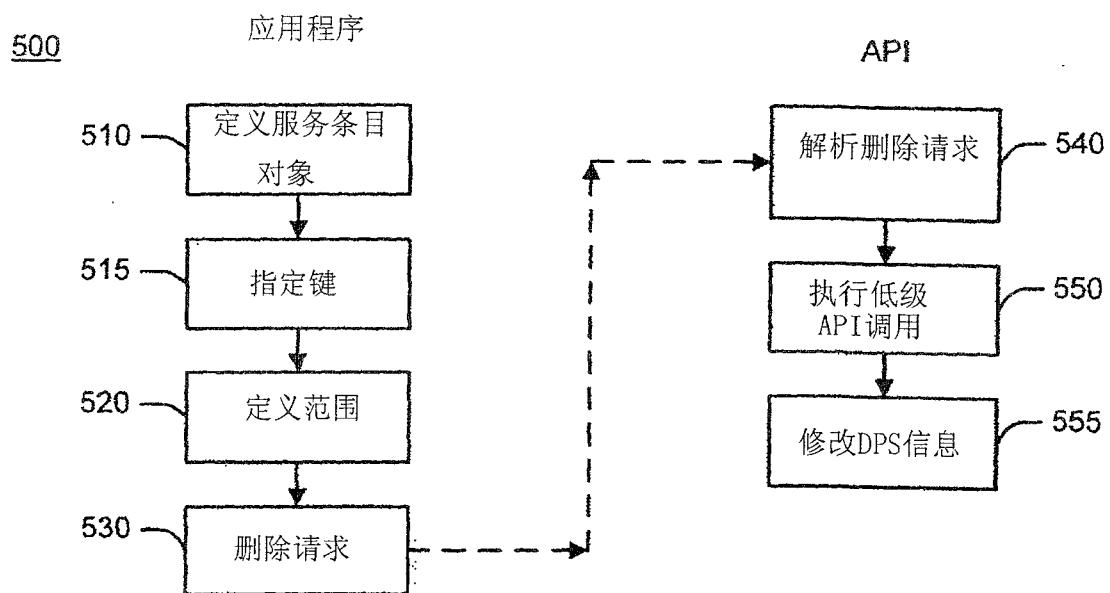


图 5

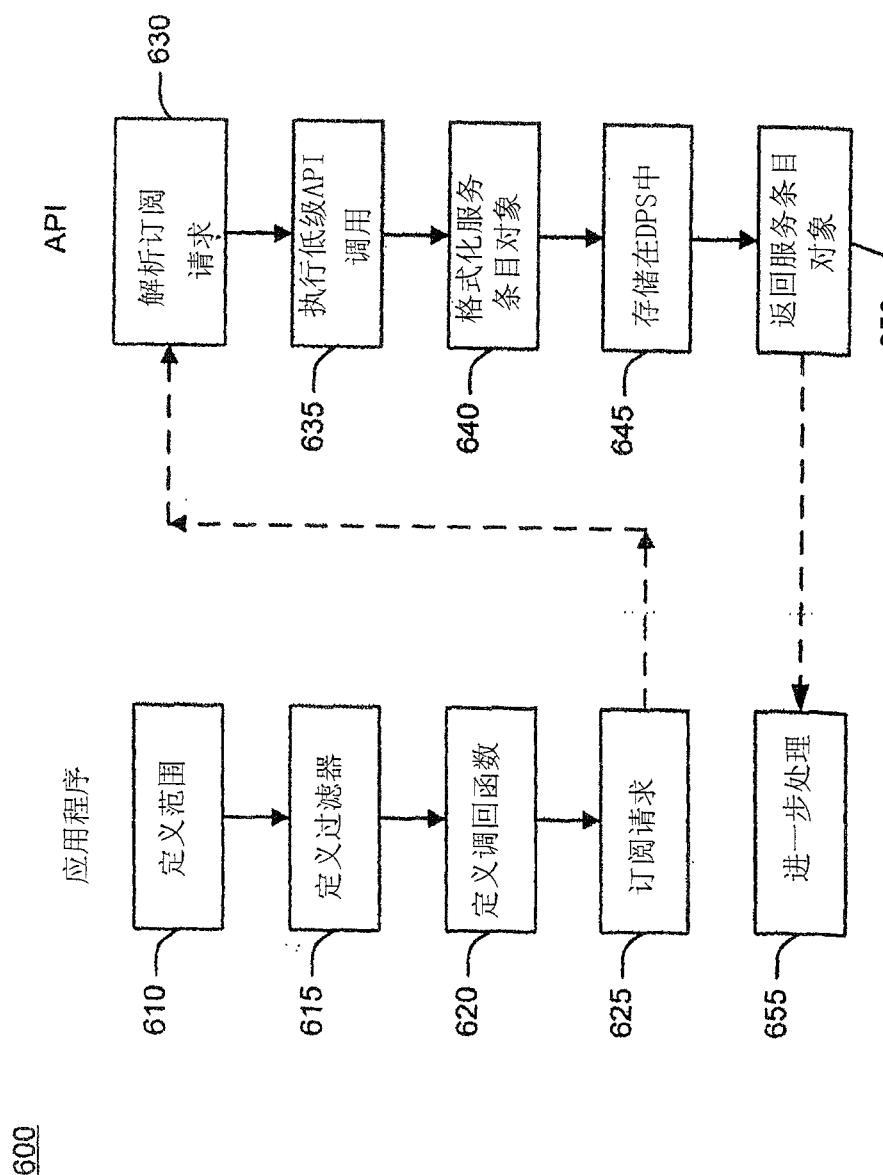


图 6

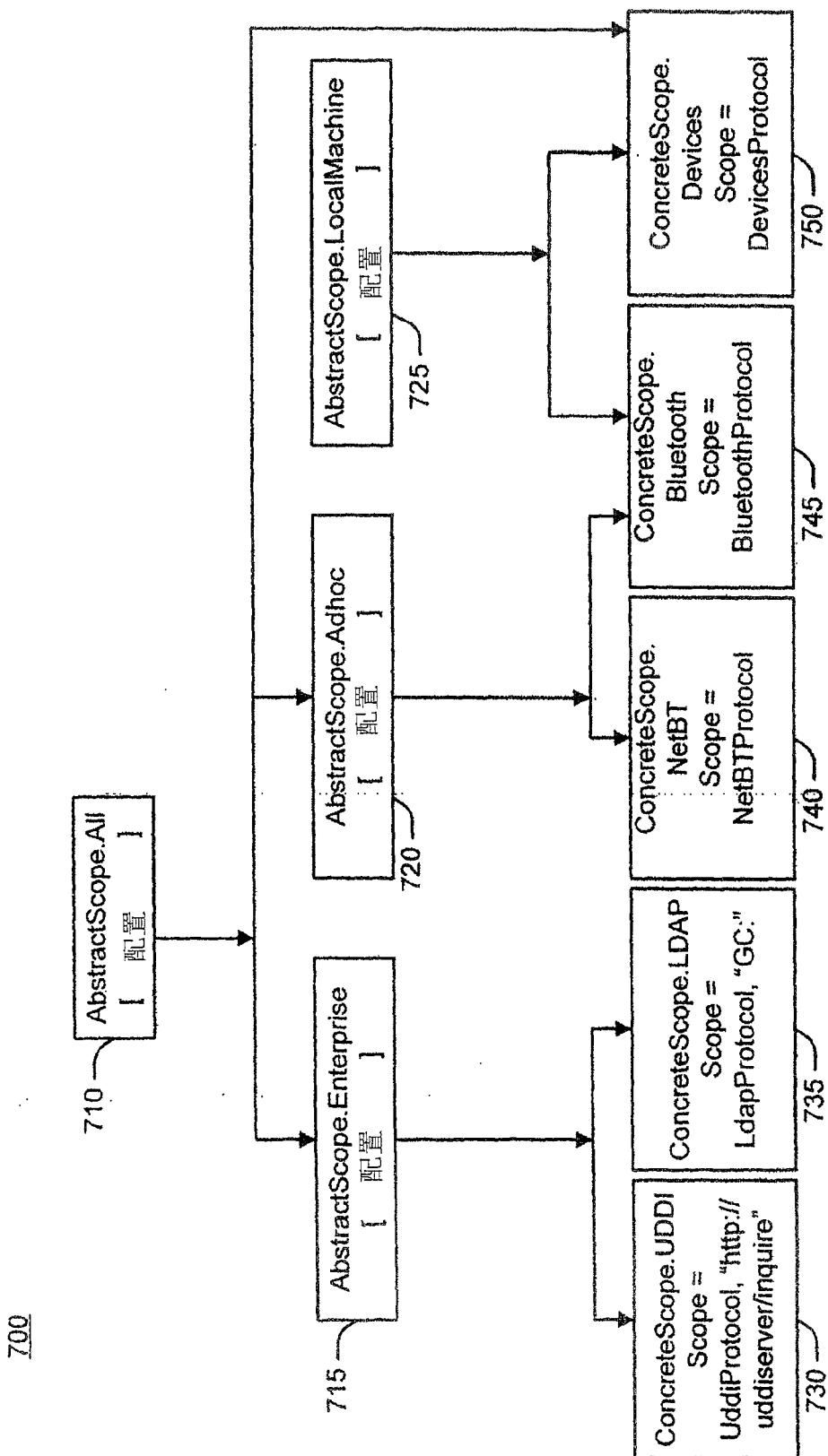


图 7

```
static void SimpleFilterAD()
{
    // 声明一个ServiceFinder
    System.Discovery.ServiceFinder servFind = new ServiceFinder();

    // 指定在全局目录下使用现用目录搜索
    servFind.Scopes.Add (new ConcreteScope (ConcreteScope.ADProtocol, "GC:",
        null));

    // 只用于打印机的过滤器
    SimpleFilter filter = new SimpleFilter ();
    filter.ServiceType = CommonServiceTypes.Printer;
    servFind.Filter = filter;

    // 声明用于匹配的属性
    filter.Properties.Add ("printcolor", "TRUE");
    filter.Properties.Add ("pagesperminute", "50");

    // 在范围内找到所有匹配过滤器的实体
    ServiceEntryCollection seColl = servFind.FindAll ();

    // 报告结果
    Console.WriteLine ("Results Found: " + seColl.Count);

    foreach (ServiceEntry se in seColl)
    {
        Console.WriteLine ("Name: " + se.Name);
        Console.WriteLine ("Description: " + se.Description);
        if ((se.Endpoints!=null) && (se.Endpoints.Count>0))
        {
            Console.WriteLine ("Address: " + se.Endpoints[0].Address);
        }
        Console.WriteLine ("Key: " + se.Key);
        Console.WriteLine ();
    }
}
```

图 8

```

static void RichFilterUDDI()
{
    // 声明一个ServiceFinder
    ServiceFinder servFind = new ServiceFinder();

    // 指定在Microsoft UDDI商业注册试验版中使用UDDI协议搜索
    servFind.Scopes.Add (new ConcreteScope (ConcreteScope.UddiProtocol,
        "http://test.uddi.microsoft.com/inquire", null));

    // 只用于名为Fabrikam且实现特定接口（即tModel “uddi-org: inquiry_v2”）的web服务的过滤器
    RichFilter filter = new RichFilter
    ("WebService[ name = 'Fabrikam' and ServiceInterface = 'uuid:ac104dco-d623-452f-88a7-f8acd94d9b2b'
    ]");

    servFind.Filter = filter;

    // 找到所有在范围内匹配过滤器的实体
    ServiceEntryCollection seColl = servFind.FindAll ();

    // 报告结果 -----
    Console.WriteLine ("Results Found: " + seColl.Count);
    foreach (ServiceEntry se in seColl)
    {
        Console.WriteLine ("Name: " + se.Name);
        if((se.Endpoints!=null) && (se.Endpoints.Count>0))
        {
            Console.WriteLine ("Address: " + se.Endpoints[0].Address);
        }
        Console.WriteLine ("Key: " + se.Key);
        Console.WriteLine ();
    }
}

```

```

static void SimpleFilterUDDI ()
{
    // 声明一个ServiceFinder
    System.Discovery.ServiceFinder servFind = new ServiceFinder();

    // 指定在Microsoft UDDI商业注册版节点中使用UDDI协议搜索
    servFind.Scopes.Add (new ConcreteScope
        (ConcreteScope.UddiProtocol,
        "http://uddi.microsoft.com/inquire", null));

    // 只用于实现指定接口（即tModel_“uddi-org: inquiry_v2”）的服务的过滤器
    SimpleFilter filter = new SimpleFilter ();
    filter.ServiceInterfaces.Add(
        "uuid:ac104dcc-d623-452f-88a7-f8acd94d9b2b");

    servFind.Filter = filter;

    // 找到所有在范围内匹配过滤器的实体
    ServiceEntryCollection seColl = servFind.FindAll ();

    // 报告结果
    Console.WriteLine ("Results Found: " + seColl.Count);

    foreach (ServiceEntry se in seColl)
    {
        Console.WriteLine ("Name: " + se.Name);
        Console.WriteLine ("Description: " + se.Description);
        if((se.Endpoints!=null) && (se.Endpoints.Count>0))
        {
            Console.WriteLine ("Address: " + se.Endpoints[0].Address);
        }
        Console.WriteLine ("Key: " + se.Key);
    }
}

```

```

Sub SimpleFilterUDDI()
    ' 声明一个ServiceFinder
    Dim servFind = New ServiceFinder

    ' 指定在Microsoft UDDI商业注册版节点中使用UDDI协议搜索
    servFind.Scopes.Add(New ConcreteScope(ConcreteScope.UddiProtocol, _
    "http://uddi.microsoft.com/inquire", Nothing))
    ' 只用于实现指定接口（即tModel "uddi-org: inquiry_v2"）的服务的过滤器
    Dim filter As New SimpleFilter
    filter.ServiceInterfaces.Add(
    "uuid:ac104dcc-d623-452f-88a7-f8acd94d9b2b")

    servFind.Filter = filter

    ' 找到所有在范围内匹配过滤器的实体
    Dim seColl As ServiceEntryCollection = servFind.FindAll()

    ' 报告结果
    Console.WriteLine("Results Found: " & seColl.Count)

    Dim se As ServiceEntry
    For Each se In seColl
        Console.WriteLine("Name: " & se.Name)
        Console.WriteLine("Description: " & se.Description)
        If Not (se.Endpoints Is Nothing) AndAlso se.Endpoints.Count > 0 Then
            Console.WriteLine("Address: " & se.Endpoints(0).Address)
        End If
        Console.WriteLine("Key: " & se.Key)
    Next se
End Sub 'SimpleFilterUDDI

```

11

图

```
static void RichFilterAD ()
{
    // 声明一个ServiceFinder
    ServiceFinder servFind = new ServiceFinder ();

    // 指定在全局目录下使用现用目录搜索
    servFind.Scopes.Add (new ConcreteScope (
ConcreteScope.ADProtocol, "GC:", null));

    // 用于名字以' Office Printer' 开头的打印机的过滤器
    RichFilter filter = new RichFilter (
"Printer[ name like 'Office Printer' ]");

    servFind.Filter = filter;

    //找到所有在范围内匹配过滤器的实体
    ServiceEntryCollection seColl = servFind.FindAll ();

    //报告结果
    Console.WriteLine ("Results Found: " + seColl.Count);
    foreach (ServiceEntry se in seColl)
    {
        Console.WriteLine ("Name: " + se.Name);
        if((se.Endpoints!=null) && (se.Endpoints.Count>0))
        {
            Console.WriteLine ("Address: " +
se.Endpoints[0].Address);
        }
        Console.WriteLine ("Key: " + se.Key);
    }
}
```

图 12

```

Sub RichFilterAD()
    ' 声明一个ServiceFinder
    Dim servFind As New ServiceFinder

    '指定在全局目录下使用现用目录搜索
    servFind.Scopes.Add(New ConcreteScope(
        ConcreteScope.ADProtocol, "GC:", Nothing))

    '用于名字以'Office Printer'开头的打印机的过滤器
    Dim filter As New RichFilter("Printer[ name like 'Office Printer' ]")

    servFind.Filter = filter

    '找到所有在范围内匹配过滤器的实体
    Dim seColl As ServiceEntryCollection = servFind.FindAll()

    '报告结果
    Console.WriteLine("Results Found: " & seColl.Count)
    Dim se As ServiceEntry
    For Each se In seColl
        Console.WriteLine("Name: " & se.Name)
        If Not (se.Endpoints Is Nothing) AndAlso se.Endpoints.Count > 0
    Then
        Console.WriteLine("Address: " & se.Endpoints(0).Address)
    End If
        Console.WriteLine("Key: " & se.Key)
    Next se

End Sub 'RichFilterAD

```

图 13

```
const string sampleServiceType = "75FF48AB-E771-4d44-9E8C-E09141112417";
static string sampleServiceKey = Guid.NewGuid().ToString();

static void Save()
{
    // 实例化一个新的ServiceEntry对象
    ServiceEntry service = new ServiceEntry();

    // 添加此范围以发布此服务条目
    service.Scopes.Add( new ConcreteScope( ConcreteScope.SsdpProtocol ) );

    // 分配一个唯一的键，有些协议会生成一个键
    service.Key = sampleServiceKey;

    // 通过设置ServiceType属性来指定服务的类型
    // the ServiceType property.
    service.ServiceType = new ServiceInterface( sampleServiceType );

    // 添加对用于访问服务的地址的端点引用
    Endpoint ep = new Endpoint( Environment.MachineName, null );
    service.Endpoints.Add( ep );

    // 将服务信息同步地发布至指定的范围
    service.Save();
}
```

图 14

```
Const sampleServiceType As String = "75FF48AB-E771-4d44-9E8C-
E09141112417"
Shared sampleServiceKey As String = Guid.NewGuid().ToString()

Shared Sub Save()
    '实例化一个新的ServiceEntry对象
    Dim service As New ServiceEntry

    '添加此范围以发布此服务条目
    service.Scopes.Add(New ConcreteScope(ConcreteScope.SsdpProtocol))

    '分配一个唯一的键，有些协议会生成一个键
    service.Key = sampleServiceKey

    '通过设置ServiceType属性来指定服务的类型
    service.ServiceType = New ServiceInterface(sampleServiceType)

    '添加对用于访问服务的地址的端点引用
    Dim ep As New Endpoint(Environment.MachineName, Nothing)
    service.Endpoints.Add(ep)

    '将服务信息同步地发布至指定的范围
    service.Save()

End Sub '保存
```

图 15

```
static void Delete()
{
    //实例化一个新的ServiceEntry对象
    ServiceEntry service = new ServiceEntry();

    // 指定将要删除的服务的唯一的键
    service.Key = sampleServiceKey;

    // 加入删除服务的范围
    service.Scopes.Add( new ConcreteScope( ConcreteScope.SsdpProtocol ) );
};

// 同步的从指定的范围删除服务信息
service.Delete();
}
```

图 16

```
Sub Delete()
    '实例化一个新的ServiceEntry对象
    Dim service As New ServiceEntry

    '指定将要删除的服务的唯一的键
    service.Key = sampleServiceKey

    '加入删除服务的范围
    service.Scopes.Add(New
ConcreteScope(ConcreteScope.SsdpProtocol))

    '同步的从指定的范围删除服务信息
    service.Delete()

End Sub '删除
```

图 17

```
public sealed class SubscriptionSample
{
    // 该委托在每次在SampleService上有更新通知时调用。
    private static ServiceFinder.ServiceUpdateEventHandler handler =
        new ServiceFinder.ServiceUpdateEventHandler( Handler );

    private static ServiceFinder finder = new ServiceFinder();

    public static void Subscribe()
    {
        // 初始化查找' castle' 家庭网络的发现程序
        finder.Scopes.Add( new ConcreteScope(
            ConcreteScope.SsdpProtocol ) );
        finder.Filter = new SimpleFilter(
            CommonServiceTypes.HomeNetwork );

        // 将我们的处理程序加入服务更新的委托
        finder.ServiceUpdate += handler;
    }

    public static void Unsubscribe()
    {
        // 将我们的处理程序从服务更新委托中删除
        finder.ServiceUpdate -= handler;
    }

    // 这里是与委托一起注册的回调函数
    private static void Handler( object sender, ServiceUpdateEventArgs e )
    {
        // 在这里处理服务更新
        Console.WriteLine( "UpdateType{0}\t(key={1})",
            e.UpdateType, e.Services[ 0 ].Key );
    }
}
```

**Public NotInheritable Class SubscriptionSample**

'该委托在每次在SampleService上有更新通知时调用。

**Private Shared finder As New ServiceFinder****Public Shared Sub Subscribe()**

'初始化查找' castle' 家庭网络的发现程序

finder.Scopes.Add(New

ConcreteScope(ConcreteScope.SsdpProtocol))

finder.Filter = New SimpleFilter(CommonServiceTypes.HomeNetwork)

'将我们的处理程序加入服务更新的委托

AddHandler finder.ServiceUpdate, AddressOf Handler

End Sub 'Subscribe

**Public Shared Sub Unsubscribe()**

'将我们的处理程序从服务更新委托中删除

RemoveHandler finder.ServiceUpdate, AddressOf Handler

End Sub 'Unsubscribe

'这里是与委托一起注册的回调函数

**Private Shared Sub Handler(ByVal sender As Object, \_**

**ByVal e As ServiceUpdateEventArgs)**

'在这里处理服务更新

Console.WriteLine("UpdateType{0}" & vbTab & "(key={1})", \_

    e.UpdateType, e.Services(0).Key)

End Sub '处理程序

End Class 'SubscriptionSample

图 19