[19]中华人民共和国专利局

[11] 公开号 CN 1133662A

[12] 发明专利申请公开说明书

[21]申请号 94193863.8

[51]Int.Cl[6]

[43]公开日 1996 年 10 月 16 日

H04M 3/00

[22]申请日 94.1.6

[30]优先权

[32]93.10.26[33]US[31]08/108,877

[86]国际申请 PCT/US94/00140 94.1.6

[87]国际公布 WO95/12267 英 95.5.4

[85]进入国家阶段日期 96.4.19

[71]申请人 塔里根特公司

地址 美国加利福尼亚州

[72]发明人 迈可尔·罗素·丢辞

史蒂文·H·米而恩

大卫·B·哥德斯密

[74]专利代理机构 永新专利商标代理有限公司

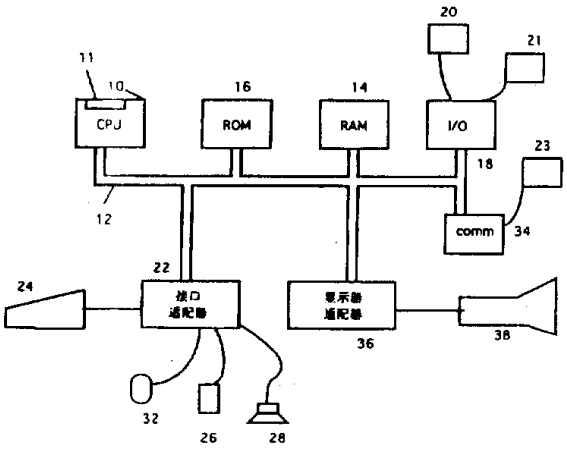
代理人 程 伟

权利要求书 4 页 说明书 80 页 附图页数 21 页

[54]发明名称 面向对象的电话系统

[57]摘要

一种用于启动一组对象接口应用部件和电话系统部件的方法和系统。根据将接口的电话系统的部件，可选择特定的对象。一个特定的对象可与电话系统的一个或多个部件接口。电话系统的部件可以是该电话系统的任何可标识的方面。例如，对象可表示一个手机或一条线路，也可表示无形的部件，例如，信号或过程，包含通话进行音调，通话建立，通话暂时中断处于占线等候，多方交谈，或其他通话特征。



权 利 要 求 书

1. 一种电话装置，包括：

(a) 一个处理器；

(b) 一个连接在所述处理器上并受其控制的存储器；

(c) 一个常驻在所述存储器中并控制所述处理器的操作的面向对象的操作系统；

(d) 一个与所述面向对象的操作系统控制下的处理器相连的显示器；

(e) 一个连接在所述处理器上的电话部件；

(f) 一个存储在所述存储器中并显示在所述显示器上的电话对象，所述电话对象表示在所述面向对象的操作系统控制下的电话部件；以及

(g) 用于由面向对象的操作系统利用所述电话对象控制所述电话部件的装置。

2. 根据权利要求 1 所述的电话装置，还包括用于将从所述电话部件接收的信息翻译成所述面向对象的操作系统能够利用的信息的装置。

3. 根据权利要求 1 所述的电话装置，还包括用于将从所述电话对象接收的信息翻译成所述电话部件能够利用的信息的装置。

4. 根据权利要求 1 所述的电话装置，其中，所述电话对象包括与电话对象相关联的一种方法及数据。

5. 根据权利要求 1 所述的电话装置，还包括用于将所述电话部件连接在所述处理器上的装置。

6. 根据权利要求 5 所述的电话装置，还包括用于将一条电话线路连接在所述处理器上的装置。

7. 根据权利要求 5 所述的电话装置，还包括用于将一个手机连接在所述处理器上的装置。

8. 根据权利要求 5 所述的电话装置，还包括用于建立对所述处理器的一次通话的装置。

9. 根据权利要求 1 所述的电话装置，还包括通过所述电话对象启动所述电话部件的特征的装置。

10. 根据权利要求 5 所述的电话装置，还包括用于在所述电话部件与所述处理器之间传递信息的装置。

11. 根据权利要求 10 所述的电话装置，还包括用于在所述电话部件与所述处理器之间交换 DTMF 音调的装置。

12. 根据权利要求 1 所述的电话装置，还包括用于在一个电话部件与所述面向对象的操作系统之间为查询服务的装置。

13. 根据权利要求 1 所述的电话装置，还包括用于在一个电话部件与所述面向对象的操作系统之间交换通知信息的装置。

14. 一种用于在一个计算机系统上启动电话部件的方法，包括带有一个连接的存储器，显示器，与电话部件的一个处理器，包括：

(a) 用常驻在所述存储器中的一个面向对象的操作系统控制所述处理器的操作；

(b) 建立一个存储在所述存储器中并显示在所述显示器上的电话对象，所述电话对象代表在所述面向对象的操作系统的控制下的电话部件；以及

(c) 由所述面向对象的操作系统利用所述电话对象控制所述电话部件。

15. 根据权利要求 14 所述的方法，还包括将从电话部件接收的信息翻译成所述面向对象的操作系统能够利用的信息的步骤。

16. 根据权利要求 14 所述的方法，还包括将从所述电话对象接收的信息翻译成所述电话部件能够利用的信息的步骤。

17. 根据权利要求 14 所述的方法，其中，所述电话对象包括与所述电话对象相关联的一种方法与数据。

18. 根据权利要求 14 所述的方法，还包括将所述电话部件连接在所述处理器上的步骤。

19. 根据权利要求 18 所述的方法，还包括将一条电话线路连接在所述处理器上的步骤。

20. 根据权利要求 18 所述的方法，还包括将一个手机连接在所述处理器上的步骤。

21. 根据权利要求 18 所述的方法，还包括建立一个对所述处理器的通话的步骤。

22. 根据权利要求14所述的方法，还包括通过所述电话对象启动所述电话部件的特征的步骤。

23. 根据权利要求18所述的方法，包括在所述电话部件与所述处理器之间传递信息的步骤。

24. 根据权利要求23所述的方法，包括在所述电话部件与所述处理器之间交换DTMF音调的步骤。

25. 根据权利要求14所述的方法，包括在一个电话部件与所述面向对象的操作系统之间交换状态信息的步骤。

26. 根据权利要求14所述的方法，包括在一个电话部件与所述面向对象的操作系统之间交换通知信息的步骤。

说　明　书

面向对象的电话系统

本发明所属技术领域

本发明涉及电话系统，尤其涉及用于将应用部件与电话部件接口的系统。

版权通告

本专利申请的某些部分受到版权保护。当出现在专利与商标局专利文件或记录中时，版权所有人不反对任何人对专利文档或专利公开的传真复制，除此以外，保留所有一切版权。

与本发明相关的背景技术

当前的电话系统是相当原始的，相对早期电话的基本原理而言其变化甚少。早期的系统通常用交换台话务员来连通呼叫方与被呼叫方。然而，借助于计算机技术，只在特殊情况下才需要话务员提供帮助。

其它技术进步已将电话系统变成了信息传递网络或数据高速公路。传真便是利用电话系统作为信息传递网络的许多实例之一。恐怕电话系统的最频繁的使用来自呼叫方与被呼叫方之间的简单语音电话通话。然而这种

交互通话只利用了电话系统优越功能性的很少一部分。

再者，典型的用户只拥有为只进行实时语音传输而设计的最基本的设备。然而除了语音以外，电话系统也能传送其它信息，并且还能提供可以更充分地利用电话系统的可用带宽与网络能力的各种各样的信息传递。

典型的 PBX （专用小交换机）便是其功能超过可利用的发送手段的技术的典型实例。普通用户只利用了大多数现代电话系统的全部功能的一小部分。标准的电话手机是人机接口的瓶颈。充其量它是以按钮行为特征的，由于在位置与功能之间没有约定的映射，这些按钮行必须加以标记。不幸的是，这些标记对于为了完成给定的操作必须按压这些按钮的序列并未给出线索，多功能按钮（转接/三路会谈）的存在更增加了情况的迷惑性。

虽然电话公司已利用计算机技术来得益与增进电话系统的潜力，单个用户尚不能利用其潜力到相同的程度。单个用户对电话系统的潜力的最直接的利用便是远程访问与使用电话应答机。

实际上电话只是一个原始的终端。与另一终端的连接是通过拿起接收机来发信号通知希望网络服务并然后输入一序列标识终端连接的号码而建立的。通过挂机来结束连接。在建立简单连接中这些操作提供方便，但电话系统的其它方面要求用户留意各种细节。

例如，电话号码通常是凭人脑记忆或保存在手写记事本中的，这一记事本必须永远随身携带。还经常使用地址本、索引卡片以及印在便宜的纸

上并装订成笨重本子的大量表格中，但存储还只是问题的一部分而已。在能使用它们之前，必须从记录中访问检索号码，查找卡片文件或目录。并且由于号码随时间改变，必须周期性地更新。将号码电子地存储在个人或联机数据库中能在一定程度上减轻检索与维护问题，但在定位了号码之后仍需要人工将其传递给电话机。

在比建立简单的一对一连接更复杂的操作的情况中，电话机也提供不理想的用户接口。例如在多方交谈的情况中，普通用户需要检验参考手册来确定使用哪些键的序列：用“*7”还是“*8”？一种替代方案是提供一种装有一行或两个附加按钮的特殊电话机，即计算机键盘上的功能键的电话等价物，但电话按钮上的标签通常稍长：“TRNSF”用于通话转接而不是“F6”。

因此，存在着挖掘当前的电话系统的潜力的要求，尤其是系统的信息传输与网络功能。

本发明的目的

因此，本发明的一个目的是提供对电话部件的访问进行的改进。

本发明的另一目的是将计算机技术应用在电话系统上。

本发明的又另一目的是提供电话系统部件与应用部件之间的一种接口。

本发明的又一目的是使电话世界与通用声音世界之间的边界平滑无隙。

本发明的技术方案

上述目的是通过提供与各种电话系统部件灵活接口的部件而实现的。这些接口部件通常并不取代电话系统部件，而是提供与特定电话系统部件的透明的接口方法。将接口部件设计成与电话系统的特定部件接口，而与电话系统的电话系统部件是硬件部件或者只是协议无关。例如，一种接口部件可能提供接收一次通话，并通知一次通话正在等待的一个应用部件的简单接口。

根据本发明的电话装置包括一个处理器；连接在该处理器上并受其控制的一个存储器；位于该存储器中并控制该处理器的操作的一个面向对象的操作系统；连接在该面向对象的操作系统控制下的处理器上的一个显示器；连接在该处理器上的一个电话部件；存储在该存储器中并显示在该显示器上并用于表示在该面向对象的操作系统的控制下的电话部件的一个电话对象；以及用于由面向对象的操作系统利用该电话对象控制该电话部件的装置。

一种根据本发明用于在一个计算机系统上启动电话部件的方法，该计算机系统包括一个处理器，该处理器具有与之连接的一个存储器，一个显示器，以及一个电话部件，该方法包括用位于该存储器中的一个面向对象的操作系统控制该处理器的操作；建立一个电话对象，该电话对象存储在该存储器中并显示在该显示器上，并用于代表在该面向对象的操作系统的

控制下的电话部件；以及出该面向对象的操作系统利用该电话对象控制该电话部件。

本发明的有益效果

接口部件提供一种灵活与方便的方法，其中，开发涉及电话的应用程序员能利用电话系统特征而无须考虑如何对一种特定的电话系统访问该特征的细节。接口部件对于电话设备的硬件设计人员也是有用的。基于计算机的电话具有消除电话机的限制的潜力，同时提供基于电话的通信与在桌面计算机上运行的应用程序的紧密合作。系统基础的主要部分要求能够实现远程和实时协作。

本发明允许应用使用目标机上可以利用的任何内部电话硬件并提供经出调制解调器或其它外部硬件设备的通用拨号功能。

在一个最佳实施例中，一个面向对象的操作系统是基于计算机的电话应用的主要部件。

从下面用附图考察的详细描述中，本发明的其它目的与优点将是显而易见的。

附图的简要说明

图 1 示出根据一个最佳实施例的代表性系统；

图 2 示出根据一个最佳实施例的手机与墙壁插座的典型连接；

图 3 示出根据一个最佳实施例的连接在多条线路与多个插座上的手

机;

图 4 示出根据一个最佳实施例的利用调制解调器的基于计算机的电话系统;

图 5 示出根据一个最佳实施例的在手机与插座之间插入一台计算机的基于计算机的电话系统;

图 6 示出根据一个最佳实施例的硬件;

图 7 示出根据一个最佳实施例的一个 TTelephoneLineHandle 对象的示例;

图 8 示出根据一个最佳实施例的 TTelephoneLineHandle 的功能特征;

图 9 示出根据一个最佳实施例的 TTelephoneCallHandle （电话通话管理）的示例;

图 10 示出根据一个最佳实施例的 TTelephoneCallHandle 的功能特征;

图 11 示出根据一个最佳实施例建立一种关心事物(interest)的应用;

图 12 示出根据一个最佳实施例的发送信息的电话系统;

图 13 示出根据一个最佳实施例的声音文件的播放;

图 14 示出根据一个最佳实施例的与 TTelephoneLineHandle （电话线路管理）交互作用来执行各种电话业务的应用;

图 15 示出根据一个最佳实施例的配置数据;

图 16 示出根据一个最佳实施例的 TTelephoneLineHandle 的功能;

图 17 示出根据一个最佳实施例的 TTelephoneLineFeatureControl（电话线路特征控制）的功能;

图 18 示出根据一个最佳实施例的 TTelephoneHandsetHandle （电话手机管理）的功能;

图 19 示出根据一个最佳实施例的 TInterest （关心事物）子类;

图 20 示出根据一个最佳实施例的 TNotification （通知）子类;

图 21 示出根据一个最佳实施例的 TTelephoneLine （电话线路）的功能特征;

图 22 示出根据一个最佳实施例的电话设备的设计人员的开发周期;

图 23 示出根据一个最佳实施例的 TTelephoneLine 的功能特征;

图 24 示出根据一个最佳实施例的 TTelephoneLineHandset （电话线路手机）的功能特征;

图 25 示出根据一个最佳实施例的 TTelephoneLineFeature 的功能特征。

本发明的最佳实施例

这里公开了本发明的详细实施例，但应理解，所公开的实施例只是本发明的示范，它们可以以各种形式实现。因此这里公开的细节不应理解为一种限制，而只是作为权利要求书的基础，及作为向熟悉本技术的人员教导如何作出与/或使用本发明的基础。

这里所使用的“电话系统”包括电话通信的所有方面。

电话标准

网络

历史上，已在逐个国家的基础上制订了网络标准。在许多情况下，它们是有相当大的变化的，从而导致局域化的恶梦，尤其在欧洲。由于诸如ISDN（综合业务数字网络）等数字标准的出现以及 1992 年欧洲共同体的成立，这一情况在不久的将来有望改善。在美国，快速地走向 ISDN 的动力不够大，应此，在未来一段时间内，很可能模拟网络和数字网络互相竞争。

在模拟领域内，电话功能的计算机控制通常是经由调制解调器完成的。（注意调制解调器不一定是一种外部硬件。 RISC （精简指令集计算机）及基于芯片的数字信号处理技术使得全功能内部软件调制解调器的实现成为可能。）一种计算机到调制解调器通信的广为流传的事实上的标准为原先由 Hayes 公司定义的所谓“ AT ”命令集。

在数字领域内，已由 CCITT （国际电报与电话咨询委员会）的一个多国工作组开发出 ISDN 标准。在 CCITT 建议 Q.931 中定义了建立与拆除连接的 ISDN 网络层协议。（ ISDN 还包括物理与数据链路层以及高层管理实体的规程）。

开关

在电话网络与电话手机之间存在着某种类型的一个开关，它将一个特定的分机连接在网络上。在商用市场上，这一开关很可能是一台 PBX （用户交换机）。除提供通过网络的简单连接之外，开关通常提供各式各样的所谓补充服务，诸如暂时中断并占线等候，转接，与多方交谈。使用这些服务的协议不但变化多端，而且在历史上已为各开关制造商所专有。虽然 Q.931 中的 ISDN 网络层规程并不建议补充特征，但在特征选择及用于实现一种给定的特征的报文的实际序列中仍留有相当大的活动余地。

音频数据格式

取决于网络的类型及计算机与电话网络之间的接口，有可能从电话线上录出音频数据以供计算机存储与播放。在模拟数据的情况中，必须设置一条模/数转换服务的通路，然后用对一个给定的设计中心有效的任何内部标准表示该数据。在数字数据的情况中，出现了外部标准问题。 CCITT 已经定义了与每秒 64,000 位传输率的 ISDNB 信道兼容的若干数据格式标准。存在着利用全部可用带宽的未压缩语音信号的两种标准。它们都基于对数压扩的 PCM （脉冲码调制）。在第一种情况中，它是北美的主要格式，对数压扩是由 μ 律函数提供的，而第二种格式主要为欧洲采用，所使用的是 A 律函数。每一格式假定 8KHZ 的取样率并为每一取样分配 8 位，从而提供 3.5KHZ 的有效信号带宽。

一方面 CCITT 采用了若干种音频压缩标准，而另一方面数字蜂窝工业则采用若干标准。

面向对象的编程

在一个最佳实施例中，本发明是用面向对象的编程技术在 C++编程语言中实现的。正如熟悉本技术的人员所理解的，面向对象的编程（ OOP ）的对象为包括数据结构及在这些数据上运算的软件实体。同时，这些元素使对象能根据由其数据元素所特征化及由其数据操作函数所表示的状态来塑造任何现实世界的实体。以这一方式，对象能够塑造象人与计算机等具体事物的模型，也能塑造象数或几何概念等抽样概念的模型。对象技术的好处来自三种基本原理：封装，多形性，及继承性。

对象隐蔽或封装它们的数据的内部结构以及其功能据其用于工作的算法。不暴露这些实现的细节，对象提供简洁地表示它们的抽象的接口而无额外的信息。多形又将封装向前推进了一步。这一观念是多种状态，一个接口。一个软件部件能请求另一部件而不必确切知道该部件是什么。接收请求的部件中断它并根据其变量与数据计算出如何执行该请求。第三种原理为继承，它容许开发人员重复使用先前存在的设计与代码。这一能力使开发人员能够避免从草案开始建立软件。而通过继承，开发人员导出继承行为的子类，然后开发人员将其定制以符合他们的特殊要求。

先有技术方法为在过程环境中分层对象与类库。上市的许多应用框架(framework)采取这一设计方法。在这一设计中，在一个整体操作系统的上面有一个或多个对象层。虽然这种方法在对象层中利用了全部封装，多形性，和继承性原理并且是对过程编程技术的实质改进，但这一方法存在相当限制。这些困难起源于这一事实，即开发人员容易重新使用他们自己的对象，但难于使用来自其它系统的对象，而开发人员仍需要用过程操作

系统（ OS ）调用来进入较低的非对象层。

面向对象的编程的另一方面为对应用开发的一种框架方法。最合理的框架定义之一来自 Illinois 大学的 Ralph E. Johnson 及 Purdue 的 Vincent F. Russo 。在 1991 年的论文，“重复使用面向对象的设计”，Illinois 大学技术报告 UTUCDCS91-1696 中，他们提出下述定义：“一个抽象类是合作完成一组任务的组对象的设计。从而，一个框架是合作执行定义的计算任务组的一组对象类。”从编程观点上，框架主要是提供预制的工作应用的结构的互相联系的对象类的群。例如，用户接口框架可能提供绘图窗口，卷动条，菜单等的支持与“系统预置”特性。由于框架是基于对象技术的，这一表现能被继承与替换，以便开发人员能扩展该框架并在特定的专门技术领域内建立定制的解决方法。由于程序员无须改变原来的代码，而只是扩展软件，所以这与传统的编程相比较是一个主要的优点。此外，开发人员并不盲目地通过代码层工作，因为框架提供体系结构指导与模型塑造，而同时又允许它们自由地提供对问题域独有的特定作用。

从商业观点上看，可将框架看成是封装或体现一个特定的知识领域内的专门知识的一种方法。合作开发组织，独立软件经销商（ ISV ），及系统综合人员已在诸如制造，会计，或货币交易等特定领域中得到专门知识，如上述实例中。这种专门知识是用它们的代码体现的。框架通过在组织的代码中体现它而使组织能捕捉与包装该专门知识的共同特征。首先，这允许开发人员建立或扩展利用该专门知识的应用，从而问题一旦得到解决，便能经常性地执行与使用这些商业规则与设计。同时，框架及框架背后所体现的专门知识对于已在诸如制造，会计，生物技术等垂直市场中的

专门知识的组织具有一种战略性资产的意义，并且提供了一种包装，转卖，及开发它们的专门知识的分配机制并促进技术的提高与普及。

历史上，框架是最近才作为个人计算平台上的主流概念出现的。这一出现已受到诸如 C++ 等面向对象的语言的可利用性的协助。传统上， C++ 绝大部分出现在 UNIX 系统及研究工作者的工作站上，而不是在商业安装的个人计算机上。这便是诸如 C++ 和诸如 Smalltalk 等其它面向对象的语言使得一些大学及科研项目产生今天的商业框架及类库的前身。其中的一些实例为来自 Stanford 大学的 InterViews，来自 Carnegie-Mellon 大学的 Andrew 工具箱及 Zurich 大学的 ET++框架。

取决于所关心的是系统的哪一级及试图解决的是哪一类问题而存在许多种框架。框架类型的范围从协助开发用户接口的应用框架到提供诸如通信，印刷，文件系统支持，图形等基本系统软件服务的低级框架。应用框架的商业实例可包括 MacApp(苹果)， Bedrock(Symantec)，OWL(Borland)， NeXTStep App Kit(NeXT)及 Smalltalk-80MVC(ParcPlace)几种。

对于习惯于其它种类的系统的开发人员而言，用框架编程需要一种新的思维方式。事实上，它完全不像传统意义上的“编程”。在诸如 DOS 或 UNIX 等老式操作系统中，开发人员本人的程序提供所有的结构。操作系统通过系统调用提供服务，开发人员的程序在它需要服务时作出调用而在提供完服务时返回控制。程序结构基于体现在开发人员编写的代码中的控制流程。

在使用框架时，这是颠倒的。开发人员不再负责控制流程。开发人员必须摒弃用执行流程表示的理解编程任务的趋势。而代之以思维必须以对象的任务来表示，它必须依靠框架来确定何时应执行这些任务。开发人员编写的例程是由不是开发人员编写的并且开发人员从未见过的代码激活的。对于只习惯于过程型编程的开发人员，控制流程中的这一转变可能成为明显的心理障碍。然而，一旦理解了这一点，框架编程比其它种类的编程需要的工作量少得多。

本发明最好在诸如 IBM® PS/2®，或 Apple® Macintosh® 计算机等个人计算机上常驻的操作系统的环境中实现。

图 1 中描绘了代表性的硬件环境，它例示了根据本发明的具有一个诸如传统的微处理器的中央处理单元 100 及若干经由一条系统总线 132 连至其它单元的一台计算机的典型硬件配置。图 1 中所示的计算机包括一个只读存储器(ROM)104 ，一个随机存取存储器(RAM)106 ，一个用于将诸如软盘单元 108 及用 110 表示的其它 I/O 外围设备连接在系统总线 132 上的 I/O 适配器 112，一个用于将键盘 130，鼠标器 126，扬声器 122,麦克风 124 ，和/或诸如触屏设备（未示出）等其它用户接口设备连接在总线上的用户接口适配器 128，以及一个用于将工作站连接在用 114 表示的一个数据处理网络上的通信适配器 116 。一个显示器适配器 120 将总线连接在显示设备 118 上。工作站上常驻有诸如 Apple® 系统/7® 操作系统之类的操作系统。

基于计算机的电话应用

取决于它们的基本要求，基于计算机的电话的面向语音的应用分成三类:

1)依靠电话特征的软件控制而无捕捉或使用语音信号的设备的应用:

2)需要录制与播放语音数据的应用; 以及

3)同时承担语音与非语音数据传输的应用。

近年来，已设计出各种各样的设备来使电话更有用。其中许多涉及计算机控制的电话。

虚拟电话机

这类应用直接访问高级电话系统的版本，这种高级电话系统如此复杂，以致只有不到 10 %的用户可以利用它的全部潜能。这些系统的基本问题是必须处理普通电话手机是具有非常低的接口带宽的设备这一事实。加在电话系统上的每一种新的特征导致手机上的一个新的按钮（或者，在 MS-DOS 的阴影下，要记住另一个特殊的号码序列: “*3 ”对转接，“*5 ”对转交）。

来自桌面上的这些功能的软件控制结合设计完善的图形用户接口将帮助终端用户绕过他们的本地 PBX 的神秘特性而使他们开始从自 Alexander Graham Bell 的日子以来电话设备制造商设法达到的技术进步中得益。这一概念的一个很好的实例便是北方电讯的 Meridian TeleCenter,它运行

在当前的 Macintosh®上，但当然只与北方电讯自己的 PBX 一起工作。

电话销售

在其最简单的形式中，计算机控制的电话销售工具只允许从一个电话号码的数据库中自动拨号。较完善的改型可在多条线路上拨号并检测何时有人在连接的另一端上应答，或者在屏幕上提供各被拨号方的详细客户信息。

语音 I/O

桌面应答机

这里的概念是为了将其提高到已能从文字邮件系统所期望的功能水平而将语音邮件放置在桌面计算机。今天的典型语音邮件系统是与其它形式的电子通信平行并完全隔离的集中服务。

即使它们是能够得到的，诸如成组寻址与联机目录等特征也很难通过标准按钮电话机使用，并且即使它们包含重复的信息也必须与电子邮件数据库分开建立与维护。呼叫进来检验自己的语音信箱只是通过办公室与家里的信息流中又一种不必要的弯路。本年代的语音邮件系统与七十年代的集中计算机系统相类似。

把语音通讯联系与桌面计算机结合将极大地改进当前刚成为广泛可利用的现有电话服务。例如，接受方检验进入的呼叫方的电话号码的能力（所谓呼叫方识别）将允许用户在其他事件中确定针对个人的问候词。例

如，如果我正在等待来自我的兄弟 John 的一个重要电话并且我知道他呼叫我时我可能不在，我便能录制一则报文，当我的计算机应答来自列在我的电子地址本中的他的电话号码之一的任何呼叫时而予播放。

在一个通话的接收端上连有一台计算机可以在一旦录制了报文时提高处理报文的潜力。如果仅有的终端只是一台按钮手机，则显然毫无可能对任何音频数据进行编辑。在语音数据传输中增加了数字格式的使用（如 ISDN），不在音频信号流过时设置捕捉它的设备便是极大的浪费。

远程访问

为了从家里访问办公室的计算机系统，所需要的只是一条电话线，一台调制解调器，及另一台计算机。如果你到了机场但是你想起忘记带了你要访问的客户的地址，你将怎么办？如果你到达了目的地发现航空公司丢失了你的调制解调器连同你其它的行李时，你将怎么办？如果你买不起第二台计算机，你将怎么办？这时你需要能够从一台标准电话机拨通你的办公室系统并通过要求计算机以读给你听的方式而访问存储在你的硬盘上的信息。你还需要在你的办公室机器上有文本到语音的合成软件以及用 DTMF （双音调多频率）键或语音命令（即自动语言识别）来控制桌面极算机的方法。

交互式语音应答

这一术语复盖允许远程呼叫者听取从事先录制的报文的集合中选择的报文的大范围应用。这些应用可以是单纯专用于信息发布（诸如 IRS 的

Teletax 服务），或者它们可综合信息交互作用业务处理（诸如，邮购目录的定购单输入服务，或者允许存款人检查帐户余额，资金转帐，以及进行借款支付的电话银行服务）。

传统上，即使将一个直观的信息发布系统组合在一起也需要程序员及系统综合人员的专业化队伍。至少一个提供这种服务的第三方已在Macintosh®上开发了一种面向图形的工具用于生成一组控制文件，然后将其上装到一个独立的语音应答系统。

语音布告板服务

这是熟悉的电子 BBS 的一种语音中介版本。它实际上是语音邮件与交互式语音应答的混合物。呼叫者浏览其它呼叫者留下的各种语音报文，随后录制他们自己的应答报文。

同时的语音与数据

合作

电话服务在合作中扮演的角色当然取决于我们所讨论的合作类型。平淡无奇的语音邮件可包含在顺序的异步合作中（诸如书评人接收文件的拷贝，私下阅读该拷贝，并用语音报文应答）。同时的异步合作包含对文件的作者发出一次呼叫（从附在文件上的“名片”对象上自动抽取电话号码），从而当作者正在显示该文件的一个独立拷贝时书评作者能发布个人评论。第三种合作是同时的实时合作方式 CHER ，其中，一份文件同时出作者与书评者更新。在最后一种合作中，组员之间交互作用的增进程度无

疑是与他们之间物理上互相靠近程度成反比变化的。这便是电话在地理上分散的工作组之间的远程合作中起作用的情况。基本概念是在一条数据连接线上传输与接收文件更新的同时，保持标准的语音联系。这种交互作用可在两条标准电话线上或一条单一的 ISDN 连接线上进行。

廉价可视电话

同时的语音与数据传输的另一种有趣的应用涉及采用一台廉价的摄象机连同一个视频数字化器卡将一条标准的 ISDN 连接线变换成低收入人员的可视电话。换言之，通过把显示通话人的面部和/或抓摄到的举到摄象机前面的物体的视频帧进行周期性的双向传输，可以增强语音连接。即使帧速率极低时，最后的结果也是相当有趣的。

本发明是设计成使应用编写人员有能力在全部三个领域内创建新的里程碑。

电话对象

本发明设计用于实现基于计算机的电话应用的开发，并允许电话设计人员利用这些新开发的应用。通过拿起受话器来发信号通知要求网络服务，电话本质上是用于建立对另一终端的联系的一个原始终端。然后输入标识你要连接的终端的一序列号码。通过挂机来切断连接。本发明通过提供一个面向语音的电话功能的高级接口来支持基于计算机的电话。这些功能包含通常的电话配置中的性能的“最小公分母”。本发明旨在将应用级代码与通过它们对这些功能进行访问的设备之间的差别分隔开。同时，本

发明提供一个灵活的框架，能容易地将它扩展成包含新的特征或提供电话机，本地硬件与软件平台，网络类型，以及开关信令协议的各种组合的特征。

当提供了这样一种灵活的接口框架时，许多应用程序编写人员将利用这一接口的可用性来开发软件。依次，电话系统设计人员要求利用新的应用部件，因此也需要接口部件进行工作。因此，讨论两种对象是有用的：一方面是应用编写人员的对象，另一方面是对基本接口的扩展的开发人员的对象。通常，应用开发人员感兴趣的对象也是电话系统开发人员所感兴趣的。各种类型将在下面讨论。但是应记住下面的标题仅用于讨论的目的，无论如何不应认为是限制性的。

I. 应用开发人员的对象

电话系统出许多对接口有用的部件构成。通常存在着两类这些部件。第一类包含用于实现电话系统的物理设备，而第二类则包含由电话系统处理的业务。下面将讨论各类的对象，并将提供概述接口部件的可能类型的实例。

A. 硬件类型部件的对象

从应用编程人员的观点，本发明方便了基于计算机的电话应用的简单与直观的用户接口的实现。对象的硬件独立性允许一个应用程序运行，而与连接在计算机上的电话系统或当前最终用户所能利用的电话特征无关。标准协议使应用一开始就能判定它所知道的有关特征中哪些是有效的而哪

些是无效的。

图 2 示出最终用户最熟悉的现实世界电话对象：插入一个墙上插座 204 以连接在一条电话线 202 上的单线电话手机 200 。事实上，在墙上插座背后有并未由普通用户所意识到的线路，光纤，开关，和总局等的庞大“网络景色(netscape)”。重要的是现在可通过一个特定的手机访问一条特定的线路 202 。然而手机与线路之间的对应并不一定是一对一的。

单一的电话线路可分成若干分机，各带有其本身的手机（未示出）或者如图 3 所示，一个单一的手机 300 可提供对若干条线路 302 的顺序访问，各条线路连接在独立的插座 304 上。前一情况是典型的住宅装机。反之，多线路手机可由小型商家或为三个总裁工作的行政助理所用。通常，用户通过按下手机 300 上某处的一个带标记的按钮而选择一条线路。

图 2 与图 3 没有标识为先有技术，因为手机 200 可以实现成包含本发明的特征。

图 4 表示实现根据本发明的基于计算机的电话的一种可能的系统。在基于计算机的电话所需的实际电话设备配置中，电话线 404 不连接在手机 400 上，而是通过诸如调制解调器 408 等外部设备的媒介而连接到计算机 412 上。线路 410 将计算机 412 连接在调制解调器 408 上，而线路 404 则将调制解调器 408 连接在插座 406 上。甚至对许多应用并无必要的手机 400 也连接在外部设备 408 或电话接口上，并且通常能独立于计算机 412 启动或终止连接。计算机能够或不能控制和/或查询手机均可。

计算机 412 可与图 1 中所示的相同，包括一台个人计算机或甚至一台工作站。计算机 412 可包括一个键盘 414，一个显示器 416，以及一个软盘驱动器 418。

图 5 展示图 4 中的系统的一种变型，示出具有用于经由线路 502 与 504 将手机 500 连接在插座 506 上的电话接口的计算机 508。计算机 508 和图 4 中所示的一样，包括键盘 512，显示器 512，及软盘驱动器 514。计算机 508 能够响应手机 500 及经由线路 502 从电话线进来的信号两者。计算机 508 也能不加修改地在手机 500 与插座 506 之间传递信号。

以上的讨论概述了可用来实现本发明的某些特定硬件。然而应记住，这些硬件只是示例性的，而本发明能在连接到电话网络上的各式各样的系统与设备上实现。图 6 展示了实施本发明的至少某些原理的一种设备。设备 600 利用对象与经由线路 602 与其连接的电话系统 604 的各种部件接口。设备 600 可以或不可以连接在一个或多个设备 606 上。

还应指出，设备 600 可很好地连接在插座以外的系统中的点上。即，面向对象的设备可从网络中接出而与电话系统的部件接口。

在其上实现本发明的系统应能执行多重任务或多重处理。这可提供应用和通过对象的系统资源的利用的最佳性能。

体系结构概况

在连接到电话系统上的设备上运行的应用利用专门设计的接口部件与电话系统的部件进行交互。应指出，术语“应用运行”意味着不仅包含在

计算机上运行的软件程序，并且还包含可执行特定应用的硬件。这些接口部件使得在各种电话系统部件之间互相传递信息所用的方法标准化。在一个最佳实施例中，接口部件是用对象有利地实现的。

电话线路管理

图 7 为始于步骤 700 的展示多个 TTelephoneLineHandle 的例子的流程图。如步骤 702 与 704 所示，只要电话线路事先未曾为数据通信保留，本发明便将其看成为可在面向语音的应用之间共享的系统资源。这些应用实际上通过代理类 TTelephoneLineHandle 访问电话服务（步骤 706 ）。例如为了允许用户在前台发出重复的自动拨号应用事例的同时而在后台保持应答机应用的操作，可将多个 TTelephoneLineHandle 例示在同一条线路上并且跨越组的边界分布（步骤 706 与 708 ）。当不再需要 TTelephoneLineHandle 时，示例过程便在步骤 710 处结束，但每当需要 TTelephoneLineHandle 事例时便可重复。

图 8 示出了 TTelephoneLineHandle 所执行的功能的操作特征的可能流程。除了允许修改（步骤 802 与 804 ）与查询（步骤 806 与 808 ）电话线的状态之外， TTelephoneLineHandle 提供建立表示线路的音频输入或输出能力（如果有的话）的 TMicrophone （麦克风）（步骤 810 与 812 ）或 TSpeaker （扬声器）（步骤 814 与 816 ）对象的部件功能。TTelephoneLineHandle 的功能流程由步骤 800 与 818 描述。

注意从语音应用的观点，“实际”电话线是通过图 4 中所示的物理调制解调器设备还是通过图 4 中所示的某种内部接口连接在计算机上是无关

紧要的。在任何一种配置中，所有应用所“见到”的只是一个TTelephoneLineHandle。

电话手机管理

TTelephoneHandsetHandle 表示与电话线路对立的物理电话设备。标准的 TTelephoneHandsetHandle 类具有用于报告其挂钩开关状态及用于建立表示手机的音频输入或输出能力（如果有的话）的标准 TMicrophone 或 TSpeaker 对象的部件功能。如果存在这种能力，手机通常在两种模式中工作，一种模式将音频直接引导到并来自物理电话线，它完全绕过计算机；而在第二种模式中，则将音频引导到并来自计算机，从而将物理线路留在回路之外。然而，在手机能用作一个本地扬声器或麦克风之前，必须明确地将其从物理线路上“断开”。注意 TTelephoneHandsetHandle 所建立的扬声器和麦克风完全与 TTelephoneLineHandle 所建立的无关。播放音频数据到手机扬声器导致手机发出声音。播放数据到线路扬声器导致在电话网络上发送声音。

子类中可包括带有按钮的物理手机，显示器，或带在相关增益控制的外部扬声器与麦克风。当然有一个“较低”的公分母：一台不能监视其开关状态的电话设备。在这一情况中，TTelephoneHandsetHandle 对象不会被过度使用。即使开关状态可用，TTelephoneHandsetHandle 的标准版本对于典型的基于计算机的电话应用也完全是供选用的。

B. 电话服务对象

除了与电话网络的使用相关联的硬件类型实体之外（上面描述的），还有比较无形的对象。例如，连接本身，或者如最终用户所想象的，电话通话可认为是无形的。一次通话具有与电话线路无关的存在和状态。为了进行通话,用户必须首先“找到”一条电话线。（“什么地方我能找到一台电话? 我要打一个电话。”）为了接受一次通话，用户需要知道是从哪一条线路上进入的。（“史密司先生，线路 1 上有你一个电话。”）一条线路是摘机或挂机的，而一次通话在其生命周期中要经过各种状态。在许多情况中，当用户为了打出第二个电话而仍然保持当前通话不挂机时，要求一条单一的线路能支持多个同时通话。替换之，用户可在一个已存在的连接加入一个通话而形成三方的电话会谈。

在现实世界中，有关一次通话的进行的信息是通过音频信道反馈给用户的：一种连续的声调表示能拨号，一种断续的声调表示目的地线路忙音，“振铃”声调表示目的地电话正在振铃等。在基于计算机的电话接口中可以得到这些相同的信号，以及以数字字母状态说明形式的关于通话状态的更有辨别力的细节。

电话通话

如图 9 中所示，在步骤 900 与 910 之间，应用通过请求（步骤 902·）TTelephoneLineHandle 的一个实例来建立一个 TTelephoneCallHandle（步骤 904 ）而启动与结束电话连接。

图 10 示出 TTelephoneCallHandle 的功能流程。如步骤 1000 与 1010 之间所示， TTelephoneCallHandle 提供确定连接的状态的访问方法（步骤 1002 与 1004 ）以及返回远程端点的电话号码的方法（步骤 1006 与 1008 ）。

电话特征控制

由于 TTelephoneLineHandle 只允许放置，答复，及挂断通话，已提供了一族的类来方便对更普遍的补充语音特征的访问。除了建立通话，线路管理能在遇到可利用高级特征的电话机时建立单个特征的控制对象。这些特征的实例中可包含下述特征集：暂时中断并占线等候，转接，多方交谈，放弃，及转交。注意特征的可利用性不仅取决于电话硬件平台潜在能够提供的特征，还取决于用户实际订购的特征。

电话通知

图 11 示出应用 1100 如何建立接收某些关于电话系统的通知的流程图。想要接收任何下面讨论的通知的客户 1100 必须用标准通知框架登记对它们关心。必须通过调用 TTelephoneLineHandle 与 TTelephoneHandsetHandle 1102 所提供的“建立关心事物”部件功能之一来建立用于电话通知的 TInterest 1104 。

图 12 示出电话设备 1200 通过能用于管理电话服务的对象发送出信息。利用 TNotification 的三个面向电话的子类从下层的电话硬件向外传播关于线路状态，通话，以及特征的信息。

TTelephoneStatusNotification 在 1202 处传送 TInterest 的三个电话专用子类之一的实例: TTelephoneInterest , TTelephoneCallInterest , 或 TTelephoneFeatureInterest 。状态更新本身由 TToken 常数表示。TTelephoneRingNotification 在 1204 处播送关于进入呼叫的信息。TTelephoneDigitsNotification 在 1206 处允许客户接收远程端点生成的 DTMF 数字。

电话号码

TPhoneNumber , TPhoneEndpoint , 以及其它的类可用于表示电话号码。

音频对象

由 TTelephoneLineHandle 与 TTelephonehandset 返回的 TSpeaker 与 TMicrophone 对象为扬声器与麦克风类的子类。一个警告: 数字电话的数据格式标准可以因地区不同而改变。例如, 从一条 ISDN 线路进入的语音数据可以是对数压扩的而不是线性的。并且该语音数据在 8KHz 上而不是在 22KHz 上数字化。为了指引这种潜在的不兼容性, 本发明设想了格式转换类。例如, 格式转换类可包括: T8KMuLawTo22KConverter , T8KALawTo22KConverter , T22KTo8KMuLawConverter , 以及 T22KTo8KALawConverter 。

图 13 提供了用格式转换对象的实例播放一个声音文件的例子。例如, 为了播放原先直接从一条北美 ISDN 线路录制的一个声音文件, 可将

文件 1300 的输出端口连接在 T8KMuLawTo22KConverter 1302 的一个实例的输入端口上。然后将该转换器的输出端口连接到 TSpeaker 1304 的一个实例的输入端口，并调用文件的播放成员功能。音频端口利用类型协商来选择一种公共数据格式。

类与成员功能细节

本节详细描述以上所述的本发明的类与成员功能。

如图 14 所示，为了利用可以得到的电话资源，在 1400 处，一个应用例示一个 TTelephoneLineHandle，如上所述。然后可以通过调用由 1402 表示的线路的 CreateAndPlaceCall 方法或由 1404 表示的 CreateAndAnswerCall 方法而建立一个电话连接，两者都返回一个调用管理。

为了得到正在进行的连接的控制，可利用由 1406 表示的线路的 CreateCallList 方法。由这三种方法中任何一种启动的连接是通过调用通话管理上的由 1408 表示的 HandUp 而终止的。

TTelephoneLineConfigurationData（电话线路配置数据）

本类表示一条特定电话线的配置。如图 15 所示，TTelephoneLineConfigurationData 1500 标识与该线路的本地端点（1502）相关联的号码簿号码并通过类型协商指定当前在线路（1504）上可得到哪些电话特征。

```
class  TTelephoneLineConfigurationData:  public MCollectible,
                                         public MTypeNegotiator
//公用构造程序
    //用电话号码集合及特征序列类型，只用电话号码或用单个电话号码初始化
    TTelephoneLineConfigurationData(const TCollection & localEndpoint,
                                    const TSequence& featureTypes);
    TTelephoneLineConfigurationData(const TCollection & localEndpoint);
    TTelephoneLineConfigurationData(const TPhoneNumber&);
    //拷贝构造程序
    TTelephoneLineConfigurationData(const
                                TTelephoneLineConfigurationData&);
    //只用于流送与分配的系统预置构造程序
    TTelephoneLineConfigurationData();
//解除程序
    virtual          ~ TTelephoneLineConfigurationData();
//赋值运算符
    TTelephoneLineConfigurationData& operator = (const
      TTelephoneLineConfigurationData&);
//访问功能
    //返回与线路关联的端点（一个或多个电话号码）
          void          GetLocalEndpoint(TPhoneEndpoint&)const;
//MTypeNegotiator 替换
    //返回有效的特征类型表
    virtual          void   CreatePreferredTypeList(TSequence&)const;
    // 确认由这一配置表示的线路存在一个特定的特征类型
    virtual TTypeDescription* ChoosesPreferredType(const
                                                   TSequence& )const;
//MCollectible 替换
```

```
MCollectibleDeclarationsMacro(TTelephoneLineConfigurationData);
virtual    long          Hash()const;
virtual    Boolean       IsEqual(const MCollectible*)const;
virtual    TStream&      operator < < = (TStream& fromWhere);
virtual    TStream&      operator > > = (TStream& toWhere)const;
```

TTelephoneLineHandle

前面已针对图 7 与图 8 讨论过 TTelephoneLineHandle。图 16 示出与作为一条电话线的标准代理者的 TTelephoneLineHandle 相关联的功能。它允许通过一个电话线配置数据对象选择一条特定的线路（ 1600 ）并提供用于启动电话通话连接的成员功能（ 1602 ），从而判定线路的挂钩状态（ 1604 ），通过类型协商匹配电话特征类型（ 1606 ），以及为线路建立音频对象（ 1608 ）。它还生成供通知框架采用的用于登记与该线路相关联的受关注的电话事件的 TInterest 子类（ 1610 ）。TTelephoneLineHandle 是多线索安全的，但并未设计成供跨越组边界的共享。

```
// 公用构造程序
    //用配置数据初始化
    TTelephoneLineHandle(const TTelephoneLineConfigurationData&);
    // 拷贝构造程序
    TTelephoneLineHandle(const TTelephoneLineHandle&);
    //只用于流送及分配的系统预置建造器
    TTelephoneLineHandle();
//公用解除程序
    virtual                ~ TTelephoneLineHandle();
```

```
//赋值运算符
    virtual TTelephoneLineHandle& operator = (const
                                        TTelephoneLineHandle&);
//应用接口
    //返回这一线路表示的端点
    virtual      void      GetLocalEndpoint(TCollection&)const;
    //检验挂钩状态
    virtual      Boolean     IsOnHook()const;
    //建立一次通话，拨指定的号码并返回通话
    virtual TTelephoneCallHandle* CreateAndPlaceCall(const
                                        TPhoneNumber&);
    //建立一次通话，应答进入的通话并返回通话
    virtual TTelephoneCallHandle* CreateAndAnswerCall();
    //返回对应于所请求的类型的特征控制对象
    virtual TTelephoneLineFeatureControl* CreateFeatureControl(const
                                        TTypeDescription& featureType);
    //返回电话线上当前正在进行的所有连接的 TTelephoneCallHandle 对象表
    virtual void CreateCallList(TCollection&)
    //送出原始 DTMF 数字
    virtual void SendDigits(const TPhoneCharacters&);
//音频 I/O
    //建立适当类型的扬声器与麦克风对象
    virtual TSpeaker* CreateSpeaker();
    virtual TMicrophone* CreateMicrophone();
//通知框架支持
    //返回这一线路所关心的挂钩状态
    virtual TInterest* CreateHookInterest()const;
    //返回这一线路所关心的振铃通知
```

```
    virtual TInterest* CreateRingInterest()const;
    //返回指定的通知管理所关心的一个通话通知
    virtual TInterest* CreateCallInterest(const
                                TTelephoneCallHandle& )const;
    //返回指定的特征控制所关心的一个特征通知
    virtual TInterest* CreateFeatureInterest(const
                                TTelephoneLineFeatureControl&)const;
//低级控制功能将线路摘机
    virtual void TakeOffHook(); \
    //将线路挂机
    virtual void PutOnHook();
//拨全部指定的电话号码
    virtual void DialNumber(const TPhoneNumber&);
    //送出原始 DTMF 数字
    virtual void SendDigits(const TPhoneCharacters&);
//MTypeNegotiator 替换
    //协商用于特征类型控制的可利用性
    virtual void CreatePreferredTypeList(TSequence&)const;
    virtual TTypeDescription* ChoosePreferredType (const
                                                TSequence&)const;
//MCollectible 替换
    MCollectibleDeclarationsMacro(TTelephoneLineHandle);
    virtual   long        Hash()const;
    virtual   Boolean     IsEqual(const MCollectible*)const;
    virtual   TStream&    operator < < = (TStream& fromWhere);
    virtual   TStream&    operator > > = (TStream& toWhere)const;
```

TTelephoneCallHandle

这是电话连接的轻加权代理者。它允许客户确定参加连接的远程号码及通话的当前状态。前面已经相对于图 9 与图 10 讨论过 TTelephoneCallHandle 。 TTelephoneCallHandle 是多线索安全的，但并未设计成跨越组边界共享的。

```
class TTelephoneCallHandle:public MCollectible
//注：本类必须通过线路或线路管理建立
//解除程序
        virtual ~TTelephoneCallHandle();
//客户接口
        //抽取用于连接的远程电话号码
        void GetRemoteNumber(TPhoneNumber& theNum)const;
        //返回当前通话状态
        virtual TToken GetStatus()const;
        //挂断通话
        virtual void HangUp();
//MCollectible 替换
        MCollectibleDeclarationsMacro(TTelephoneCallHandle);
        virtual        long          Hash()const;
        virtual        Boolean       IsEqual(const MCollectible*)const;
//注：客户不得分配或流送本类
```

TTelephoneLineFeatureControl

图 17 示出了在 1700 处响应一个客户而用 TTelephoneLineHandle 在 1700 建立 TTelephoneLineFeatureControl 。 TTelephoneLineHandle 可以或不能支持超出执行和断开电话连接的能力的任意组特征。例如，线路管理可能能够在新的通话到来时采用占线等候的方式暂时中断当时的通话，转送一次通话，进行多方交谈，或将原有的通话换到另一条线路。客户通过要求线路管理建立 TTelephoneLineFeatureControl 的一个子类而使用诸如此类的特征。特征控制子类是用基于所要求的特征的类名的 TTypeDescription 指定的。 TTelephoneLineFeature-Control 是多线索安全的，但并未设计成跨越边界的共享。

```
class TTelephoneLineFeatureControl:public MCollectible
//用于访问定义的特征类型的静态成员定义如下:
static const void GetHoldFeatureType(TTypeDescription&);
static const void GetDropFeatureType(TTypeDescription&);
static const void GetTransferFeatureType(TTypeDescription&);
static const void GetConferenceFeature Type(TTypeDescription&);
static const viod GetForwardFeatureType(TTypeDescription&);
        //注: 本类必须通过线路或线路管理建立
        //公用解除程序
        virtual ~ TTelephoneFeatureControl();
        //客户接口
            //返回特征类型描述
            virtual void GetFeatureType(TTypeDescription&)const = 0;
            //返回当前特征状态
```

```
        virtual TToken GetStatus()const;
//MCollectible 替换
        MCollectibleDeclarationsMacro(TTelephoneLineFeatureControl);
        virtual        long          Hash()const;
        virtual        Boolean       IsEqual(const MCollectible*)const;
//注：客户不得分配或流送本类
```

标准 TTelephoneLineFeatureControl 子类

提供了五种标准特征控制子类，它们对应于“大四”补充特征并加上通话转交。

1）暂时中断并占线等候特征控制：

```
class TTelephoneHoldFeatureControl:public TTelephoneLineFeatureControl
//公用解除程序
virtual ~ TTelephoneHoldFeatureControl();
//TTelephoneLineFeatureControl 替换
        //返回特征类型描述
        virtual void GetFeatureType(TTypeDescription&)const;
//客户接口
        //暂时中断并占线等候当前有效的通话
        virtual void PutOnHold();
        //恢复指定的通话
        virtual void Reconnect(const TTelephoneCallHandle&);
//MCollectible 替换
        MCollectibleDeclarationsMacro(TTelephoneHoldFeatureControl);
        virtual        long          Hash()const;
```

```
        virtual        Boolean        IsEqual(const MCollectible*)const;
```

2）放弃特征控制:

```
class TTelephoneDropFeatureControl:public TTelephoneLineFeatureControl
//公用解除程序
virtual ~TTelephoneDropFeatureControl();
//TTelephoneLineFeatureControl 替换
        //返回特征类型描述
        virtual void GetFeatureType(TTypeDescription&)const;
//客户接口
        //放弃指定的通话
        virtual void Drop(const TTelephoneCallHandle&);
//MCollectible 替换
        MCollectibleDeclarationsMacro(TTelephoneDropFeatureControl);
        virtual        long        Hash()const;
        virtual        Boolean        IsEqual(const MCollectible*)const;
```

3）转接特征控制:

```
class TTelephoneTransferFeatureControl:publicTTelephoneLineFeatureControl
//公用解除程序
virtual ~TTelephoneTransferFeatureControl();
//TTelephoneLineFeatureControl 替换
        //返回特征类型描述
        virtual void GetFeatureType(TTypeDescription&)const;
//客户接口
        //起动转接并将通话放置到指定的号码
```

```
        virtual TTelephoneCallHandle* CreateCallAndStartTransfer(const
                                                TPhoneNumber& newNumber);
        //结束进行中的转接操作
        virtual void CompleteTransfer();
        //中止进行中的转接操作
        virtual void CancelTransfer();
//MCollectible 替换
        MCollectibleDeclarationsMacro(TTelephoneTransferFeatureControl);
        virtual       long          Hash()const;
        virtual       Boolean       IsEqual(const MCollectible*)const;
```

4）多方交谈特征控制:

```
class TTelephoneConferenceFeatureControl:public
        TTelephoneLineFeatureControl
//公用解除程序
virtual ~TTelephoneConferenceFeatureControl();
//TTelephoneLineFeatureControl 替换
        //返回特征类型描述
        virtual void GetFeatureType(TTypeDescription&)const;
//客户接口
        //起动多方交谈操作并将通话放置到指定号码
        virtual TTelephoneCallHandle*
        CreateCallAndStartAddingToConference(const TPhoneNumber&
        newNumber);
        //完成正在进行的多方交谈操作
        virtual void FinishAddingToConference();
        //中止正在进行的多方交谈操作
```

```
        virtual void CancelAddingToConference();
//MCollectible 替换
        MCollectibleDeclarationsMacro(TTelephoneConferenceFeatureControl);
        virtual        long          Hash()const;
        virtual        Boolean       IsEqual(const MCollectible*)const;
```

5）转交特征控制:

```
class TTelephoneForwardFeatureControl:public TTelephoneLineFeatureControl
//公用解除程序
virtual ~TTelephoneForwardFeatureControl();
//TTelephoneLineFeatureControl 替换
        //返回特征类型描述
        virtual void GetFeatureType(TTypeDescription&)const;
//客户接口
        //转交线路给指定的号码
        virtual void Forward(const TPhoneNumber&phoneNumber);
        //撤消通话转交
        virtual void CancelForward();
//MCollectible 替换
        MCollectibleDeclarationsMacro(TTelephoneForwardFeatureControl);
        virtual        long          Hash()const;
        virtual        Boolean       IsEqual(const MCollectible*)const;
```

TTelephoneHandsetConfigurationData（电话手机配置数据）

本类表示一个特定的电话手机的配置。它标识该电话手机可连接的一条或多条电话线路。这已参照图 15 讨论过。

```
class TTelephoneHandsetConfigurationData:public MCollectible
//构造程序

//用手机能够连接的电话线的配置初始化
TTelephoneHandsetConfigurationData(const TCollection&);
TTelephoneHandsetConfigurationData(const
                                    TTelephoneLineConfigurationData&);

//拷贝构造程序
TTelephoneHandsetConfigurationData(const
TTelephoneHandsetConfigurationData&);

//只用于流送与分配的系统预置构造程序
TTelephoneHandsetConfigurationData();

//解除程序
virtual ~TTelephoneHandsetConfigurationData();

//赋值运算符
TTelephoneHandsetConfiguationData& operator=(const
      TTellephoneHandsetConfigurationData&);
```

```
//访问功能
    //为手机能够连接的所有电话线建立配置数据对象
    virtual void CreateConnectibleLineConfigurations(TCollection&)const;
//MCollectible 替换
    MCollectibleDeclarationsMacro(TTelephoneHandsetConfigurationData);
    virtual     long          Hash()const;
    virtual     Boolean       IsEqual(const MCollectible*)const;
    virtual     TStream&      operator < < = (TStream& fromWhere);
    virtual     TStream&      operator > > = (TStream& toWhere)const;
```

TTelephoneHandsetHandle

图 18 提供 TTelephoneHandsetHandle 的功能表示，它是物理电话终端设备的标准代理者。它提供监视挂钩状态的机制（ 1800 ）并允许建立对应于手机的内部扬声器与麦克风的音频对象（ 1802, 1804 ）。它还生成供通知框架使用的用于登记与手机相关的受关注的电话事件（ 1806 ）。作为一种物理设备，手机通常提供对一条特定的电话线路的手动控制，从而拿起手机就要导致某处的电话线路成为摘机。在用于音频 I/O 之前，手机必须从其线路上“断开”。也有可能将一个手机连接到一组电话线中的一条上。 TTelephoneHandsetHandle 是多线索安全的，但并未设计成跨越组边界的共享。

```
class TTelephoneHandsetHandle:public MCollectible
//公用构造程序

    //用配置数据初始化
    TTelephoneHandsetHandle(const TTelephoneHandsetConfigurationData&);
```

```
        //拷贝建造器
        TTelephoneHandsetHandle(const TTelephoneHandsetHandle&);

        //只用于流送与分配的系统预置构造程序
        TTelephoneHandsetHandle();

        //公用解除程序
        virtual  ~ TTelephoneHandsetHandle();

//赋值运算符
virtual TTelephoneHandsetHandle& operator=(constTTelephoneHandsetHandle&);
//应用接口
        //检验挂钩状态
        virtual Boolean IsOnHook()const;
        //检验连接状态
        virtual Boolean IsConnected()const;
        //连接到一条特定的电话线
        //注: 手机一次只能连接到一条线路
        virtual void ConnectToTelephoneLine(const TTelephoneLineHandle&);
        //从一条特定的电话线上断开
        virtual void DisconnectFromTelephoneLine();
        //为当前连接到该手机的线路建立配置
        virtual TTelephoneLineConfigurationData*
        CreateLineConfigurationData() const;
        //为可用来连接该手机的所有线路建立配置数据对象
        virtual void
        CreateConnectibleLineConfigurations(TCollection&)const;
//音频 I/O
```

```
        //建立适当类型的扬声器与麦克风音频对象
        注：这些对象表示手机本身而不是它所连接的线路的音频 I/O 能力
        virtual         TSpeaker*       CreateSpeaker();
        virtual         TMicrophone*    CreateMicropone();
//通知框架支持
        //返回这一手机的挂钩状态关心事物
        virtual         TInterest*      CreateHookInterest()const;

        //返回这一手机的振铃通知关心事物
        virtual         TInterest*      CreateRingInterest()const;
MCollectible 替换
        MCollectibleDeclarationsMacro(TTelepnoneHandsetHandle);
        virtual     long        Hash()const;
        virtual     Boolean     IsEqual(const MCollectible*)const;
        virtual     TStream&    operator < < = (TStream& fromWhere);
        virtual     TStream&    operator > > = (TStream& toWhere)const;
```

TInterest 子类

图 19 示出 TInterest 子类。本发明利用三种特定的 TInterest 子类来通知状态更新：TTelephoneInterest 1900，TTelephoneCallInterest 1908，以及 TTelephoneFeatureInterest 1912。TTelephoneInterest 1900 允许客户请求通知影响一条特定电话线或手机的某些电话事件。这些事件包括挂钩状态中的改变 1902，来自进入呼叫的振铃指示 1904，及到达的远程生成的数字串 1906。TTelephoneCallInterest 1908 允许客户请求通知一个特定的电话连接过程中的变化 1910。TTelephoneFeatureInterest 1912 同样允许客户请求

通知一种电话特征的特定事例的过程中的变化 1914 。客户应该通过调用 TTelephoneLineHandle 与 TTelephoneHandsetHandle 所提供的适当成员功能来建立上述类的事例 1916 。它们都不是多线索安全的。

• TTelephoneInterest

```
class TTelephoneInterest:public TInterest
//公用构造程序
TTelephoneInterest(const TTelephoneInterest&);
TTelephoneInterest();
//公用解除程序
virtual ~TTelephoneInterest();

//赋值运算符
virtual TTelephoneInterest& operator=(const TTelephoneInterest&);
//MCollectible 替换
      MCollectibleDeclarationsMacro(TTelephoneInterest);
      virtual     long       Hash()const;
      virtual     Boolean    IsEqual(const MCollectible*)const;
      virtual     TStream&   operator < < = (TStream& fromWhere);
      virtual     TStream&   operator > > = (TStream& toWhere)const;
```

• TTelephoneCallInterest

```
class TTelephoneCallInterest:publlic TInterest
//公用构造程序
TTelephoneCallInterest(const TTelephoneCallInterest&);
TTelephoneCallInterest();
```

```
//公用解除程序
virtual ~ TTelephoneCallInterest();

//赋值运算符
virtual TTelephoneCallInterest& operator=(const TTelephoneCallInterest&);
//MCollectible 替换
        MCollectibleDeclarationsMacro(TTelephoneCallInterest);
        virtual     long        Hash()const;
        virtual     Boolean     IsEqual(const MCollectible*)const;
        virtual     TStream&    operator < < = (TStream& fromWhere);
        virtual     TStream&    operator > > = (TStream& toWhere)const;
```

- TTelephoneFeatureInterest

```
class TTelephoneFeatureInterest:public TInterst

//公用构造程序
TTelephoneFeatureInterest(const TTelephoneFeatureInterest&);
TTelephoneFeatureInterest();
//公用解除程序
virtual ~ TTelephoneFeatureInterest();
//赋值运算符
virtual TTelephoneFeatureInterest& operator=(const
                                        TTelephoneFeatureInterest);
//MCollectible 替换
        MColloectibleDeclarationsMacro(TTelephoneFeatureInterest);
        virtual     long        Hash()const;
        virtual     Boolean     IsEqual(const MCollectible*)const;
```

```
virtual    TStream&   operator < < = (TStream& fromWhere);
virtual    TStream&   operator > > = (TStream& toWhere)const;
```

关心事物名常量

下述公用静态常量是为与通知框架一起使用而定义的:

- 定义在 TTelephoneInterest 范围内的关心事物名:

```
//挂钩状态更新
static const TToken& kHookStatus;
//电话振铃关心事物名
static const TToken& kRingNotification;
//TTelephone 数字关心事物名
static const TToken& kDigitsReceived;
```

- 定义在 TTelephoneCallInterest 范围内的关心事物名:

```
//通话状态
static const TToken& kCallStatus;
```

- 定义在 TTelephoneFeatureInterest 范围内的关心事物名:

```
//电话特征状态
static const TToken& kFeatureStatus;
```

TNotification 子类

图 20 示出从 TNotification 子类得出的功能关系。有三种 TNotification 的面向电话的子类。 TTelephoneStatusNotification 2000 输送一个状态更新 2002 并包含一个 TTelephoneInterest, 一个 TTelephoneCallInterest, 或一个 TTelephoneFeatureInterest 2004 来替代一个简单的 TInterest 。取决于与它们相关的操作的进行，可将状态更新分配给四种类型之一: 成功, 失败, 进行中, 或未知。这允许应用执行一个进入通知的初步分类而无须明显地检验其状态字段。 TTelephoneRingNotification 2006 在它已被应答之前输送一个进入呼叫的电话号码 2008 。最后， TTelephoneDigitsNotification 2010 输送来自一个远程连接的电话键盘的原始 DTMF 数字 2012 。客户通常通过通知框架 2016 而不是通过建立它们自己的事例来得到这些类的事例 2014 。这些类不是多线索安全的。

• 状态通知

```
class TTelephoneStatusNotification:public TNotification
//公用构造程序
TTelephoneStatusNotification(const TTelephoneStatusNotification&);
TTelephoneStatusNotification(const TInterest&);
//公用解除程序
virtual ~TTelephoneStatusNotification();
//赋值运算符
virtual TTelephoneStatusNotification& operator=(const
        TTelephoneStatusNotification&);
```

```
//客户接口
//取得/设置状态信息
        TToken GetStatus()const;
//取得通知分类
        long GetClassification()const;
//MCollectible 替换
        MCollectibleDeclarationsMacro(TTelephoneStatusNotification);
        virtual     long        Hash()const;
        virtual     Boolean     IsEqual(const MCollectible*)const;
        virtual     TStream&    operator < < = (TStream& fromWhere);
        virtual     TStream&    operator > > = (TStream& toWhere)const;
```

- 振铃通知:

```
class TTelephoneRingNotification:public TNotification
//公用构造程序
TTelephoneRingNotification(const TPhoneNumber&);
TTelephoneRingNotification();
TTelephoneRingNotification(const TTelephoneRingNotification&);
//公用解除程序
virtual ~ TTelephoneRingNotification();
//赋值运算符
virtual TTelephoneRingNotification& operator=(const
                                        TTelephoneRingNotification&);
//客户接口
        //抽取呼叫方电话号码
        void GetIncomingNumber(TPhoneNumber&)const;
//Mcollectible 替换
```

```
MCollectibleDeclarationsMacro(TTelephoneRingNotification);
virtual      long        Hash()const;
virtual      Boolean     IsEqual(const MCollectible*)const;
virtual      TStream&    operator < < = (TStream& fromWhere);
virtual      TStream&    operator > > = (TStream& toWhere)const;
```

- 进入数字通知:

```
class TTelephoneDigitsNotification:public TNotification
//公用构造程序
      TTelephoneDigitsNotification(const TTelephoneDigitsNotification&);
      TTelephoneDigitsNotification();
//公用解除程序
      virtual ~TTelephoneDigitsNotification();
//赋值运算符
virtual TTelephoneDigitsNotification& operator=(const
                                        TTelephoneDigitsNotification&);
//客户接口
      viod GetDigits(TPhoneCharacters&)const;
//MCollectible 替换
      MCollectibleDeclarationsMacro(TTelephoneDigitsNotification);
      virtual     long        Hash()const;
      virtual     Boolean     IsEqual(const MCollectible*)const;
      virtual     TStream&    operator < < = (TStream& fromWhere);
      virtual     TStream&    operator > > = (TStream& toWhere)const;
```

状态常量

下述公用静态常量 TToken 标记是为 TTelephoneStatusNotification::的范围内的电话状态更新而定义的

- 线路状态常量

KOnHook — 线路或手机挂机

KOffHook — 线路或手机摘机

- 通话状态常量

kCallIdle — 通话无效。系统预置状态。

kIncompleteCallInfo — 通话安排正在进行，但需要更多信息。

kCallReorder — 通话安排不能前进。号码中有错误。

kCallCannotComplete — 通话安排不能完成。

kReadyToDial — 存在拨号音。

kFastBusy — 拨号音暂停。

kDialingComplete — 成功地完成拨号。

kRingbackDetected — 收到返回振铃指示。目的地电话正在振铃。

kBusyDetected — 收到忙音指示。目的地电话号码正忙。

kNoAnswer — 目的地电话号码无应答。

kCallActive — 通话已成功地连接。

kCallOnHold — 通话已被暂时中断处于占线等候状态。

kCallTerminationDetected — 网络或连接方已结束通话。

- 特征控制常量

kActivateInProgress — 已确认特征激活请求。

kActivateComplete — 特征现被激活。

kCannotActivate — 不能激活特征。

kDeactivateInProgress — 已确认特征释放请求。

kDeactivateComplete — 特征现被释放。

kCannotDeactivate — 不能释放特征。

kErrorState — 出现一个不明错误。

ETelephoneStatusClassification

ETelephoneStatusClassification 是一种在 TTelephoneStatusNotification 范围内定义的危害，用以表示一种状态通知的通用分类，从而客户能适当地响应而无须检验状态值。定义的值为:

kSuccess — 成功地完成操作

kFailure — 没有完成操作

kInProgress — 正在正常进行操作

kUnknown — 对于这一状态操作的进行不明

转换器类

数据格式转换是通过诸如 T8KMuLawTo22KConverter, T8KALawTo22K-Converter, T22KTo8KMuLawConverter, 和 T22KTo8KALawConverter 等转换器进行的。这些类继承 MAudio 并且不增加它们自己的新的接口协议。

II. 电话系统开发人员的对象

除了上面讨论的对象以外，电话系统开发人员在开发新的电话设备时需要进一步的对象用来工作。例如，诸如 AT&T 等第三方开发人员可能希望增加对专有电话设备的支持和/或扩展本发明来提供高级通讯能力。本发明将尽可能快与好地满足第三方开发人员的这些努力。电话体系结构在使得与标准对象相关的现存代码的潜在重复使用最大化的同时，还应预见这些扩展的需要。

对象的体系结构概述

电话开发人员须考虑三类种在上面应用部分中未描述的类。抽象的电话线路类定义对电话网络功能的一个标准接口。它具有与 TTelephoneLineHandle 面对面的主代理者关系，并包括用于与可能存在多个组中的处理进行通信的内装式机构。由 TTelephoneFeature 及其子类提供用于访问超出简单的挂/摘机与拨号功能的通用补充特征的接口。希望在专用电话硬件上提供对标准功能的访问的开发人员将需要产生线路类的子类及预先定义的特征子类。

电话线

图 21 表示抽象的主类 TTelephoneLine,它提供对电话网络的物理连接的一个软件模型。每一条电话线路具有相关的(1)一种挂钩状态(挂或摘)2100 ，(2)包含一组唯一的号码簿号码 2102 的一个 TPhoneEndpoint ， 以及(3)一组本身可用的电话特征 2104 。此外，一条

线路管理向其引导状态更新的一张有效的处理表 2116 。

TTelephoneLine 包含纯虚拟的成员功能，用于进行摘机，挂机，以及拨出数字。这些功能中的每一种的实现必须由开发人员提供。在大多数情况下，这种实现将需要与专用于给定的硬件/软件环境的一个 I/O 访问管理程序和/或一个听音程序任务通信。

例如，如果物理电话接口位于运行一个特定任务 A 的 γ 总线卡上，设备驱动器的角色便可出作为 TTelephoneLine 子类的一个成员字段的示例的一个 TaskACard 对象来担任。对该子类的 OffHook 方法的调用会导致线路向 TaskACard 发送一则包含适当的命令与数据的报文。通过一个包含 TaskAListener 的听音程序任务向该线路送回应答。然后，线路子类建立适当的 TNotification 子类并通过从 TTelephoneLine 继承的 ReceiveUpdate 方法将它们向下传送到相关的 TTelephoneLineHandle 。

开发人员可能感兴趣的两个其它的成员功能为 CreateSpeaker 与 CreateMicrophone 。这两个功能具有简单地返回一个 NIL （零）指针的系统预置实现。然而，建立请求调度程序来平整非零指针供另一方的 TTelephoneLineHandle 再现。一个最佳实施例包含与实际电话硬件对应的音频对象以及对声音服务程序的适当调用来例示这些对象。注意在许多情况中（诸如连接在串行端口上的一个标准的外部调制解调器），不可能在主计算机上传进或传出实际的语音数据而必须保留系统预置实现。

保证按正常序列来生成出线路处理与通话处理类所期望的通知事件是 TTelephoneLine 子类的责任。

电话特征

为了提供超出建立与断开连接所需的高级特征，设想基本TTelephoneLine 类支持诸如最普遍使用的五种“暂时中断并占线等候”，“放弃”，“转接”，“多方交谈”，与“转交”高级电话特征的请求的调度与处理。在一个给定的电话开关上可以得到的特定特征组既取决于该开关的内在能力，还取决于该开关的客户所选订的服务。由于这一点以及由于每年开关制造商推出更新与更精巧的特征， TTelephoneLine 工作在一组复杂的特征上，从而多形地激活与释放它们。

继续参见图 21 ，其中示出了 TTelephoneLineFeature 的子类。TTelephoneLineFeature 为所有特征从其中导出的基本类，它定义激活/释放协议。从 TTelephoneLineFeature 2104 可导出一组标准特征子类: TTelephoneHoldFeature 2106 ， TTelephoneDropFeature 2108 ， TTelephoneTransferFeature 2110 ， TTelephoneConferenceFeature 2112 ，以及 TTelephoneForwardFeature 2114 。其中每一个与保持该特征的控制协议的任何子类所使用的一种类型描述相关联。上列五种特征的标准类型描述是由 TTelephoneLineFeatureControl 通过静态成员功能提供的。

希望提供实现标准特征组的开发人员必须建立 TTelephoneHoldFeature 2106 ， TTelephoneDropFeature 2108 ， TTelephoneTransferFeature 2110 ， TTelephoneConferenceFeature 2112 ，以及 TTelephoneForwardFeature 2114 的子类，替换继承的纯虚拟功能激活与释放，以便执行有用的工作。

图 22 示出在 2202 处，如果具有着没有标准 TTelephoneLineFeature 子类存在的高级特征情况下的过程。在 2204 处，如果该特征是标准的，便使用它。如果该特征不是标准的，则依赖于开发人员通过置换纯虚拟的成员功能 GetTypeDescription 在 2206 提供一个类型描述，并在 2208 处提供激活与释放的实现。开发人员还必须在 2210 处定义一个特征控制子类以提交给线路处理，并且必须在 2212 处置换 CreateFeatureControl，以使其返回控制的一个实例。

电话配置

本发明采用两种配置信息。例如，从一个可显示号码到一个可拨号码的转换要求知道本地区的代码，以及用于访问长途服务的前缀。这一类信息可随地区变化，但对于装设在一个给定地点的所在电话线路则是通用的。另一方面，本地电话号码及与该号码相关联的分机所能利用的特征则是在同一地点内随不同线路而不同的信息。

在企业安装中，可由系统管理人员提供地点专用的配置信息并包含有分配给单个用户的装机组件。用户仍必须指定线路专用的配置。非企业用户要负责提供这两组信息。配置数据的实际显示与采集可能需要开发人员生成一个能够采集输入的专用应用程序或一个数据模型。

配置信息是由 TTelephoneLineConfigurationData 表示的，它是包含一个由给定的 TTelephoneLine 所需要的所有地点与线路专用的配置数据的一个轻加权配置类。一种类似的关系可在 TTelephoneHandsetConfigurationData 与 TTelephoneHandset 之间得

出。这两个类已在前一部分中描述过。

音频对象

如果开发人员的 TTelephoneLine 子类能够捕捉与再现实际的语音信号，则其依赖于电话开发人员通过声音服务程序提供支持这一能力所需的音频类。例如，对从 ISDN 卡提供音频 I/O 感兴趣的开发人员要编写 MAudio/TAudioProcessor 对以表示该硬件的输入与输出能力。

电话开发人员的编程接口

本节详细描述前一节中所引入的开发人员工具的类与成员功能。想要实现标准功能和特定电话环境的特征的开发人员将对 TTelephoneLine 以及标准的 TTelephoneLineFeature 和 TTelephoneLineFeatureControl 子类再行分类。想要增加新特征与功能的开发人员将需要 TTelephoneLineFeature 与 TTelephonelineFetureControl 的其他子类。必须支持专用的终端设备的开发人员将编写 TTeleponeHandset 与 TTelephoneHandsetHandle 的子类。

TTelephoneLine

图 23 示出 TTelephoneLine 2300 的功能动作。这是表示一条电话线路的基本对象。它提供对简单的电话网络功能的低级接口。功能包括: TakeOffHook 2302 ， PutOnHook 2304 ， DialNumber 2306 ，以及 SendDigits 2308 。该线路建立 TelephoneCallHandle 2310 来表示连接，从其 TTelephoneLineFeature 组中得到 TTelephoneFeatureControl

2312，并且也能建立麦克风与扬声器音频对象 2314 。一个 TTelephoneLine 与一个或多个称作 TTelephoneLineHandle 的代理对象相关联。处理中继对它们的主线路的请求，后者又依次将电话挂钩、振铃、通话、特征与数字通知分配给线路处理。诸如暂时中断并占线等候，放弃，转接，多方交谈等高级特征是通过 TTelephoneLineFeature 的子类支持的。 TTelephoneLine 已设计成由属于不同的组的管理安全地共享。

```
class TTelephoneLine:protected MRemoteDispatcher,
                     public MDelegatingNotifier,
                     private MReferenceCounted
//公用构造程序
virtual ~TTelephoneLine();
//公用访问功能
//指定该线路的配置
//注: 客户可能想要改变一条现有线路的配置
virtual void SetConfigurationData(const
                                  TTelephoneLineConfigurationData&);
//建立与返回该线路的配置
 virtual TTelephoneLineConfigurationData* CreateConfigurationData() const;
//受保护的构造程序
//指定本地配置信息
TTelephoneLine(const TTelephoneLineConfigurationData&);
//拷贝构造程序
TTelephoneLine(const TTelephoneLine&);
//系统预置构造程序，只用于流送
TTelephoneLine();
//受保护的赋值运算符
```

```
        TTelephoneLine& operator=(const TTelephoneLine&);
//子类接口
        //在特征表上增加一个新特征对象
        //线路子类应在它们的构造程序中建立与采用特征
        virtual void AdoptFeature(TTelephoneLineFeature*);
        //指定该线路实例为之配置的特征类型
        //采用特征后必须在构造程序中调用的
        //如果要配置的特征不存在便生成异常
        virtual void ConfigureFeatures(const TCollection&);
        //响应外部事件更新线路与传播通知
        virtual void ReceiveUpdate(TNotification&);
        //挂钩状态设定程序
        virtual void SetHookStatus(const TToken&);
//来自管理的进程请求，这些成员功能是受保护的
//注：以下 4 种纯虚拟功能的特定实现必须为使用的实际电话硬件提供
        //将线路摘机
        virtual void TakeOffHook()=0;
        //将线路挂机
        virtual void PutOnHook()=0;
        //拨全部指定的电话号码
        virtual void DialNumber(const TPhoneNumber&)=0;
        //送出原始 DTMF 数字
        virtual void SendDigits(const TPhoneCharacters&)=0;
//其余的受保护的请求管理具有系统预置的实现：
        //作为原子功能放置通话 - 提供系统预置实现
        virtual TTelephoneCallHandle* CreateAndPlaceCall(const
                                        TPhoneNumber&);
        //作为原子功能应答通话 - 提供系统预置实现
```

```
        virtual TTelephoneCallHandle* CreateAndAnswerCall();
        //返回该线路上正在进行的所有通话的管理
        virtual void CreateCallList(TCollection&)const;
        //返回当前挂钩状态
        virtual TToken UpdateHookStatus() const;
        //返回指定的通话的当前状态
        virtual TToken UpdateCallStatus(const TTelephoneCallHandle&);
        ///返回对应于所请求的特征类型的特征控制对象
        virtual TTelephoneLineFeatureControl* CreateFeatureControl(
const TTypeDescription&) const;
//受保护的音频 I/O 功能
        //建立适当类型的扬声器与麦克风音频对象
        virtual TSpeaker* CreateSpeaker()=0;
        virtual TMicrophone* CreateMicrophone()=0;
//受保护的通话管理功能
        //建立一个新连接
        virtual TTelephoneCallHandle* AddCallAndCreateHandle(
                                        const TPhoneNumber&);
        //定位与指定的状态匹配的通话管理
        virtual TTelephoneCallHandle* FindCallAndCreateHandle(
                                        const TToken& callState)const;
        //消除一个存在的连接
        virtual void DeleteCall(const TTelephoneCallHandle&);
//受保护的特征管理功能
        //定位指定类型的特征
        virtual TTelephoneLineFeature* GetFeature(
                                const TTypeDescription&  featureType)const;
```

```
//受保护的生成通知的功能
        virtual TNotification* CreateHookNotification(const TToken&
                                                      HookState,
TTelephoneStatusNotification :: ETelephoneStatusClassification
            classification=TTelephoneStatusNotification :: kSuccess)const;
        virtual TNotification* CreateRingNotification(const
TPhoneNumber&)const;
vritual TNotification*CreateCallNotification(const TToken& callState,
        const TTelephoneCallHandle&,
        TTelephoneStatusNotification :: ETelephoneStatusClassification
        classification=TTelephoneStatusNotification :: kSuccess)const;
virtual TNotification* CreateFeatureNotification(const TToken&
        featureState,const TTelephoneLineFeature&,
        TTelephoneStatusNotification :: ETelephoneStatusClassification
        classification=TTelephoneStatusNotification :: kSuccess)const;
virtual TNotification*CreateDigitsNotification(const TPhoneCharacters&)
                                                      const;

//公用MCollectible替换
        VersionDeclarationsMacro(TTelephoneLine);
        virtual     long        Hash()const;
        virtual     Boolean     IsEqual(const MCollectible*)const;
//受保护的流送运算符只能由子类调用
        virtual     TStream&    operator<<=(TStream& fromWhere);
        virtual     TStream&    operator>>=(TStream& toWhere)const;
```

TTelephoneHandset

图 24 示出 TTelephoneHandset 2400 执行的功能流程。这是代表一个物理电话手机的基本对象。它也能建立麦克风与扬声器音频对象 2402 。一个 TTelephoneHandset 2400 与一个或多个称作 TTelephoneHandsetHandle 2404 的代理对象相关联。管理中断对它们的手机的请求 2406 ，后者又依次将状态更新 2408 分配给管理。TTelephoneHandset 2400 已设计成由属于不同组的管理安全地共享。

```
class TTelephoneHandset:protected MRemoteDispatcher,
                        public MDelegatingNotifier,
                        public MReferenceCounted
//公用解除程序
virtual ~ TTelephoneHandset();
//公用访问功能
//指定该手机的配置
//注: 客户可能想要改变一条现有线路的配置
virtual void SetConfigurationData(const
                                  TTelephoneHandsetConfigurationData&);
//建立与返回该手机的配置
virtual TTelephoneHandsetConfigurationData* CreateConfigurationData()
                                                        const;
//受保护的构造程序
//指定本地配置信息
TTelephoneHandset(const TTelephoneHandsetConfigurationData&);
//拷贝构造程序
TTelephoneHandset(const TTelephoneHandset&);
```

```
//系统预置构造程序 - 只用于流送
TTelephoneHandset();
//赋值运算符
TTelephoneHandset& operator=(const TTelephoneHandset&);
//子类接口
        //响应外部事件更新线路及传播通知
        virtual void ReceiveUpdate(TNotification&);
        //受保护的挂钩状态设定程序
        virtual void SetHookStatus(const TToken& theStatus);
        //用指定的配置定位可连接的电话线路
        //为使于实现"ConnectToTelephoneLine"纯虚拟功能而提供本功能
        virtual TTelephoneLine* FindLine(
                                const TTelephoneLineConfigurationData&);
//来自管理的进程请求 - 这些成员功能是受保护的
//注: 对于使用的实际电话硬件，必须提供用于以下二种纯虚拟功能的特定实现
    //连接到配置数据指定的线路
    //自动进行可连接性检验，连接状态维护
    virtual void ConnectToTelephoneLine(const TTelephoneLineHandle& )=0;
    //从当前相关的线路上断开
    //自动维护连接状态
    virtual void DisconnectFromTelephoneLine()=0;
//其余的受保护的请求管理具有系统预置的实现:
    //挂钩状态获取程序
    virtual TToken UpdateHookStatus() const;
    //连接状态获取程序
    virtual Boolean UpdateConnectionStatus() const;
```

```
        //获取当前连接的电话线路的配置数据
        //任何时刻，手机可连接在这些线路中的零或一条上
        virtual TTelephoneLineConfigurationData*
                                        CreateLineConfigurationData()const;
        //为手机可以连接的所有电话线路建立配置数据对象
        virtual void CreateConnectibleLineConfigurations(Tcollection&)
                                                        const;
/受保护的音频 I/O 功能
        //建立适当类型的扬声器与麦克风音频对象
        //注：这些对象表示手机而不是它所连接的电话线路的本地音频 I/O 能力
        virtual TSpeaker* CreateSpeaker()=0;
        virtual TMicrophone* CreateMicrophone()=0;
//受保护的生成通知功能
        virtual TNotification* CreateHookNotification(const TToken&
                                                        hookState,
        TTelephoneStatusNotification::ETelephoneStatusClassification
           classificaiton=TTelephoneStatusNotification::kSuccess) const;
        virtual TNotification* CreateRingNotification(const TPhoneNumber&)
                                                        const;

//公用 MCollectible 替换
        VersionDeclarationsMacro(TTelephoneHandset);
        virtual     long        Hash()const;
        virtual     Boolean     IsEqual(const MCollectible*)const;
//受保护的流送运算符只能由子类调用
        virtual     TStream&    operator<<==(TStream& fromWhere);
        virtual     TStream&    operator>>==(TStream& toWhere);
```

TTelephoneLineFeature

图 25 示出 TTelephoneLineFeature 2500 的功能。TTelephoneLineFeature 为定义一组标准的高级电话线路特征的一个基本类。它定义多种形式的激活 2502 与释放 2504 高级电话特征 2510 的协议，并提供判定它们的类型 2506 及查询它们的状态字段 2508 的方法。TTelephoneLineFeature 不是多线索安全的，但是出线路以安全的方式访问。

```
class TTelephoneLineFeature:protected MRemoteDispatcher
//公用解除程序
        virtual         ~ TTelephoneLineFeature();
//公用访问功能
//返回特征类型描述 - 纯虚拟功能
        virtual void          GetFeatureType(TTypeDescription&)const=0;
        // 返回当前特征状态
        virtual    TToken     UpdateStatus() const;
        //建立匹配控制对象
        //纯虚拟部件功能 - 必须加以替换
        virtual TTelephoneLineFeatureControl*CreateFeatureControl()const=0;
//受保护的构造程序
        // 供子类使用 - 系统预置成释放状态
        TTelephoneLineFeature();
        //拷贝构造程序
        TTelephoneLineFeature(const TTelephoneLineFeature&);
//受保护的赋值运算符
        TTelephoneLineFeature&  operater=(const TTelephoneLineFeature&);
```

```
// 子类接口
        //多种形式激活/释放特征
        // 纯虚拟功能 - 必须加以替换
        virtual         void    Activate()=0;
        virtual         void    Deactivate()=0;
//公用 MCollectible 替换
        VersionDeclarationsMacro(TTelephoneLineFeature);
        virtual         long            Hash()const;
        virtual         Boolean         IsEqual(const MCollectible*)const;
// 受保护的流送运算符只能由子类调用
        virtual         TStream&        operator<<=(TStream& fromWhere);
        virtual         TStream&        operator>>=(TStream& toWhere)const;
```

标准 TTelephoneLineFeature 子类

- 暂时中断并占线等候特征:

```
class TTelephoneHoldFeature:public TTelephoneLineFeature

// 解除程序
        virtual         ~ TTelephoneHoldFeature();
// TTelephoneLineFeature 替换
        // 返回标准暂时中断并占线等候类型描述
        virtual     void     GetFeatureType(TTypeDescription&)const;
        //返回标准暂时中断并占线等候控制对象
        virtual     TTelephoneFeatureControl* CreateFeatureControl()const;
//受保护的构造程序
        TTelephoneHoldFeature();
```

```
        TTelephoneHoldFeature(const TTelephoneHoldFeature&);
```

- 放弃特征:

```
class TTelephoneDropFeature:public TTelephoneLineFeature

// 公用解除程序
        virtual      ~TTelephoneDropFeature();
// TTelephoneLineFeature 替换
        //返回标准放弃类型描述
        virtual   void GetFeatureType(TTypeDescription&) const;
        //返回标准放弃控制对象
        virtual   TTelephoneFeatureControl* CreateFeatureControl()const;
//受保护的构造程序
        TTelephoneDropFeature()
        TTelephoneDropFeature(const TTelephoneDropFeature&);
```

- 转接特征:

```
class TTelephoneTransferFeature:public TTelephoneLineFeature

// 解除程序
        virtual      ~TTelephoneTransferFeature();
// TTelephoneLineFeature 替换
        // 返回标准转接类型描述
        virtual    void    GetFeatureType(TTypeDescription&)const;
        // 返回标准转接控制对象
        virtual    TTelephoneFeatureControl*CreateFeatureControl()const;
```

```
// 受保护的构造程序
        TTelephoneTranferFeature();
        TTelephoneTransferFeature(const TTelephoneTransferFeature&);
```

• 多方交谈特征:

```
class TTelephoneConferenceFeature:public TTelephoneLineFeature
// 解除程序
        virtual     ~ TTelephoneConferenceFeature();
// TTelephoneLineFeature 替换
        //返回标准多方交谈类型描述
        virtual     void     GetFeatureType(TTypeDescription&)const;
        // 返回标准多方交谈控制对象
        virtual     TTelephoneFeatureControl* CreateFeatureControl()const;
// 受保护的构造程序
        TTelephoneConferenceFeature();
        TTelephoneConferenceFeature(const TTelephoneConferenceFeature&);
```

• 转交特征:

```
class TTelephoneForwardFeature: public TTelephoneLineFeature
//公用解除程序
        virtual     ~ TTelephoneForwardFeature();
//TTelephoneLineFeature 替换
        // 返回标准转交类型描述
        virtual     void     GetFeatureType(TTypeDescription&)const;
        // 返回标准转交控制对象
        virtual TTelephoneFeatureControl* CreateFeatureControl()const;
```

```
// 受保护的构造程序
    TTelephoneForwardFeature();
    TTelephoneForwardFeature(const TTelephoneForwardFeature&);
```

TTelephoneLineHandle

已在应用编写人员接口部分中描述了本类的构造程序，解除程序，以及一些其它的成员功能。在编写一个 TTelephoneLineFeatureControl 子类时，TTelephoneLineHandle 所提供的下述低级控制功能可能有用。

```
// 将线路摘机
virtual    void     TakeOffHook();
// 将线路挂机
virtual    void     PutOnHook();
// 拨全部指定的电话号码
virtual    void     DialNumber(const TPhoneNumber&);
// 送出原始 DTMF 数字
virtual    void     SendDigits(const TPhoneCharacters&);
```

TTelephoneFeatureControl

以下是用于 TTelephoneFeatureControl 的功能的类似的表。它们内部地用于实现控制的特征特定的协议。GetLineHandle 返回的指针给出对其关联的线路管理的公用成员功能的控制访问。

```
// 发送激活/释放请求到对应的特征对象
virtual    void     Activate();
```

```
virtual    void      Deactivate();
// 返回参照给通话的线路管理
TTelephoneLineHandle*    GetLineHandle()const;
```

音频考虑

与公用电话网络本身的交互作用通常是间歇的且相当慢，从而施加很少（如果有的话）严格的性能限制。反之，音频数据的出现带来实时要求组的标准，这些要求连同模/数与数/模转换硬件一起工作。跟不上这些硬件的要求会引入不良的数据不连续性，其结果是不但降低语音数据的清晰性程度，并且十分难听（甚至痛苦）。然而，这些实时要求不应损害诸如用户接口工具等非实时处理的性能。

具体而言，录制或播放一个声道的音频不会在诸如(a)选择一个菜单项，(b)键入一个字符，(c)选择一个图形对象，或(d)一个正文范围等标准用户操作上提供任何感觉得到的不利影响。在下列各种情况中这应当是可能的：

1)从电话手机上捕捉并录制到盘上的 8KHz 对数压扩的取样

2)从电话网络上捕捉并录制到盘上的 8KHz 对数压扩的取样

3)从盘上检索到并播放到手机上的 8KHz 对数压扩的取样

4)从盘上检索到并播放到网络上的 8KHz 对数压扩的取样

5)从盘上检索到并播放到系统扬声器上的 8KHz 对数压扩的取样

6)从盘上检索到并播放到手机上的 22KHz 线性取样

7)从盘上检索到并播放到网络上的 228KHz 线性取样

情况 5 至 7 要求具有同时压扩到线性或线性到压扩的转换的取样速率转换。

为了使本发明与音频对象一起操作，电话对象支持各式各样的标准。例如，设想成目标机器支持 μ 律与 A 律格式两者并提供转换成和来自标准音频格式的转换。通过提供与音频对象交互作用的能力，电话世界和普通声音世界之间的边界呈现为平滑与“无缝”的。有可能通过声音服务程序来示例封装电话线路和电话手机的音频输入或输出能力的一个标准麦克风或扬声器对象。

根据一个最佳实施例的实例

以上的讨论提供了可能用于电话设备接口的某些示范性对象的描述。以下提供使用以上讨论的某些对象的实例。

放置一次通话:

```
...
// 建立一个新的 TTelephoneHandle.
// 线路配置是一个 TTelephoneHandsetConfigurationData 对象
```

```
TTelephoneLineHandle myLineHandle(lineConfiguration);
//建立一个新的通话对象
TPhoneNumber myNumber=TPhoneNumber("974-0001")
TTelephoneCallHandle* myPhoneCall=
        myLineHandle.CreateAndPlaceCall(myNumber);
...
// OK, 离开电话的时间到了
myPhoneCall.HangUp();
delete myPhoneCall;
```

应答一次通话:

```
...
// 为了示例, 我们说明一个特殊的类来接收振铃通知
class TRingHandler
{
public:
        TRingHandler(const TTelephoneLineHandle*);
virtual         ~TRingHandler();
virtual       void   HandleRing(const TTelephoneRingNotification&);
private:
TTelephoneLineHandle*       fLineHandle;
TTelephoneCallHandle*       fCallHandle;
TMemberFunctionConnection   fConnection;
}
// TRingHandler 的定义如下
TRingHandler::TRingHandler(const TTelephoneLineHandle* lineHandle)
                              :fLineHandle(lineHandle)
```

```
{
        fConnection.SetReceiver(this, (NotificationMemberFunction)
                        &TRingHandler::ReceiveEvent);
        fConnection.AdoptInterest(fLineHandle->CreateRingInterest());
        fConnection.Connect();
}
TRingHandler::~TRingHandler()
{}
TRingHandler:: HandleRing(const TNotification& ringNotification)
{
        // 在连接通话前,可以问谁在呼叫
        TPhoneNumber callingParty;
        ((const TTelephoneRingNotification&)
                        theNotification).GetIncomingNumber(callingParty);
        // 继续及应答它
        fCallHandle = fLineHandle->CreateAndAnswerCall()
}
```

暂时中断通话处于占线等候:

```
...
//建立一个新的 TTelephoneHandle 如上
TTelephoneLineHandle myLineHandle(lineConfiguration);
// 协商暂时中断并占线等候特征
// 实际上,这将在例示线路以前用代理完成
TDeque featureList;
TTypeDescription holdFeatureType;
TTelephoneLineFeatureControl::GetHoldFeatureType(holdFeatureType);
```

```
featureList.AddLast(&holdFeatureType);
TTypeDescription* featureType=
                    myLineHandle.ChoosePreferredType(featureList);
Boolean canHold = (featureType !=NIL);
// 建立一个新的通话对象
TPhoneNumber myNumber = TPhoneNumber("974-0001")
TTelephoneCallHandle* myPhoneCall = myLineHandle.CreateAndPlaceCall
      (myNumber);
...
// OK,通话现在是有效的,因此暂时中断并处于占线等候.
if ((myPhoneCall->GetStatus() == TTelephoneStatusNotification: :
              kCallActive)&& (canHold)) {
      TTelephoneFeatureControl* holdControl;
      holdControl = myLineHandle.CreateFeatureControl(holdFeatureType);
      holdControl->PutOnHold();
...
      //他已等待太久，让我们再与他连接
      holdControl->Reconnect(myPhoneCall);
      }
...
```

增加一种新的电话特征:

```
//-------------------------------------------------------
//     TFancyFeature - 说明
//-------------------------------------------------------
class TFancyFeature : public TTelephoneLineFeature{
public:
```

```
// 构造程序
    TFancyFeature();
    TFancyFeature(const TFancyFeature &);
// 解除程序
      virtual       ~ TFancyFeature();
// 赋值运算符
      TFancyFeature&       operator=(const TFancyFeature&);
// TTelephoneLineFeature 替换
      // 返回新的特征类型描述
      virtual void GetFeatureType(TTypeDescription&)const;
      // 返回新的特征控制对象
      virtual  TTelephoneFeatureControl* CreateFeatureControl() const;
      // 激活特征
      virtual       void   Activate();
      // 释放特征
      virtual       void   Deactivate();
// MCollectible 替换
MCollectibleDeclarationsMacro(TFancyFeature);
      virtual       long         Hash() const;
      virtual       Boolean      IsEqual(const MCollectible*) const;
      virtual       TStream&     operator<<=(TStream& fromWhere);
      virtual       TStream&     operator>>=(TStream& toWhere)const;
}
// ---------------------------------------------------------------
//    MFancyFeature - 定义
// ---------------------------------------------------------------
// 构造程序实现
    TFancyFeature::TFancyFeature()
```

```
{}

// 解除程序实现
    TFancyFeature::~TFancyFeature()
{}
// 返回新的特征类型描述
void   GetFeatureType(TTypeDescription& featureType)const;
{
       TFancyFeatureControl::GetFancyFeatureType(featureType);
}
// 建立新的特征控制
TTelephoneFeatureControl* TFancyFeature::CreateFeatureControl() const
{
       TTelephoneFeatureControl* fancyControl=new TfancyFeatureControl();
       return fancyControl;
}
// 激活特征
void   TFancyFeature::Activate()
{
       // Do Actual Work to Activate Feature
}

// 释放特征
void   TFancyFeature::Deactivate()
{
       // 进行实际工作来释放特征
}
// MCollectible 替换
```

```
MCollectibleDefinitionsMacro(TFancyFeature,0);
...
// ------------------------------------------------------------------
//      MFancyFeatureControl - 说明
// ------------------------------------------------------------------
class TFancyFeatureControl : public TTelephoneFeatureControl{
public:
static const void GetFancyFeatureType(TTypeDescription&);
// 解除程序
      virtual              ~TFancyFeatureControl();

// TTelephoneLineFeatureControl 替换
      // 返回特征类型描述
      virtual      void   GetFeatureType(TTypeDescription&)const;
// 客户接口
      // 访问该特征 - 可能需要传递线路或通话管理参数
      virtual      void   TurnOnFancyFeature();
      virtual      void   TurnOffFancyFeature();
      // Mcollectible 替换
      MCollectibleDeclarationsMacro(TFancyFeatureControl);
      virtual      Long   Hash() const;
      virtual      Boolean      IsEqual(const MCollectible*)const;
protected:
      virtual      TStream&     operator<<=(TStream& fromWhere);
      virtual      TStream&     operator>>=(TStream& toWhere) const;
// 构造程序
      // 被来自指定的特征的值初始化
      TFancyFeatureControl(const TFancyFeatureControl&);
```

```
        // 拷贝构造程序
        TFancyFeatureControl(const TFancyFeatureControl&);

        // TFancyFeature 调用的系统预置构造程序
        TFancyFeatureControl();
        // 赋值运算符
        TFancyFeatureControl& operator=(const TFancyFeatureControl&);
// 只有一个 TFancyFeature 能构造该类
        friend class TFancyFeature;
};
// ------------------------------------------------------------------
//      TFancyFeatureControl - 定义
// ------------------------------------------------------------------

const void TTelephoneLineFeatureControl::GetFancyFeatureType
                                        (TTypeDescription& featureType)
{
    static const TTypeDescription fancyType("BrandXFancyFeature");
    featureType=fancyType;
}
// 解除程序
TFancyFeatureControl::~TFancyFeatureControl()
{}
//返回特征类型描述
void TFancyFeatureControl::GetFeatureType(TTypeDescription& featureType)
                                                        const
{
        GetFancyFeatureType(featureType);
```

```
}
/ /客户接口
void   TFancyFeatureControl::TurnOnFancyFeature()
{
       Activate();
}
void TFancyFeatureControl::TurnOffFancyFeature()
{
       Deactivate();
}
/ / MCollectible 替换
MCollectibleDefinitionsMacro(TFancyFeatureControl,0);
//构造程序
//用来自指定特征的值初始化
TFancyFeatureControl::TFancyFeatureControl(const TFancyFeature&
              theFeature)::TTelephoneLineFeatureControl(theFeature)
{
}
//拷贝构造程序
TFancyFeatureControl::TFancyFeatureControl(const TFancyFeatureControl&
                                                          source)
                     :TTelephoneLineFeatureControl(source)
{
}
/ / 只为流送与赋值调用系统预置构造程序
TFancyFeatureControl::TFancyFeatureControl():
TTelephoneLineFeatureControl()
{
```

```
}
// 赋值运算符
TFancyFeatureControl&       TFancyFeatureControl::operator=(
                                    const TFancyFeatureControl& source)
{
        // 无自我赋值
        if (&source!=this){
        TTelephoneFeatureControl: operator=(source);
        }
        return *this;}
```

预定义的电话常数

名称	意义
电话通知所关心的事物名	
TTelephoneInterest::kHookStatus	挂钩状态更新
TTelephoneCallInterest::kCallStatus	通话状态更新
TTelephoneFeatureInterest::kFeatureStatus	特征状态更新
TTelephoneInterest::kRingNotification	进入通话通知
TTelephoneInterest::kDigitsReceived	进入 DTMF 通知
电话线路状态	
TTelephoneStatusNotification::kOnHook	线路是挂机的
TTelephoneStatusNotification::kOffHook	线路是摘机的
电话通话状态	
TTelephoneStatusNotification::kCallIdle	系统预置状态
TTelephoneStatusNotification::kIncompleteCallInfo	放置通话需要更多信息

TTelephoneStatusNotification::kCallReorder	通话放置不能进行
TTelephoneStatusNotification::kCallCannotComplete	测到特殊服务信息音调
TTelephoneStatusNotification::kReadyToDial	存在拨号音
TTelephoneStatusNotification::kFastBusy	拨号音时间过完
TTelephoneStatusNotification::kDialingComplete	完成拨出
TTelephoneStatusNotification::kRingbackDetected	远程连接正在“振铃”
TTelephoneStatusNotification::kBusyDetected	收到忙音，远程电话忙
TTelephoneStatusNotification::kNoAnswer	远程电话未应答
TTelephoneStatusNotification::kCallActive	通话已成功连接
TTelephoneStatusNotification::kCallOnHold	通话已被暂时中断并处于占线等候
TTelephoneStatusNotification::kCallTerminationDetected	另一方已终止

电话特征状态

TTelephoneStatusNotification::kActivateInProgress	确认特征激活请求
TTelephoneStatusNotification::kActivateComplete	现在激活特征
TTelephoneStatusNotification::kCannotActivate	不能激活特征
TTelephoneStatusNotification::kDeactivate	正在进行特征释放请求确认
TTelephoneStatusNotification::kDeactivateComplete	现在释放特征
TTelephoneStatusNotification::kCannotDeactivate	不能释放特征

通用错误状态

TTelephoneStatusNotification::kErrorState	不好或未定义的状态

基本电话通话状态机

以下提供利用上面讨论的某些对象的一种通话状态机的实现。

起始状态: kCallIdle[Outbound Call]

动作: TTelephoneLineHandle::TakeOffHook 结束状态: kReadyToDial

起始状态: kCallIdle[Incoming Call]

动作: TTelephoneLineHandle::TakeOffHook 结束状态: kCallActive

起始状态: kReadyToDial

事件: kFastBusyDetected 结束状态: kFastBusyDetected

动作: TTelephoneLineHandle::SendDigits 结束状态: kDialingComplete

动作: TTelephoneLineHandle::SendDigits 结束状态: kIncompleteCallInfo

动作: TTelephoneLineHandle::SendDigits 结束状态: kCallReorder

动作: TTelephoneCallHandle::HangUp 结束状态: kCallIdle

起始状态: kFastBusyDetected

动作: TTelephoneCallHandle::HangUp 结束状态: kCallIdle

起始状态: kDialingComplete

事件: kRingbackDetected 结束状态: kRingbackDetec

事件: kBusyDetected 结束状态: kBusyDetected

事件: kCallCannotComplete 结束状态: kCallCannotComplete

动作: TTelephoneCallHandle::HangUp 结束状态: kCallIdle

起始状态: kIncompleteCallInfo

动作: TTelephoneCallHandle::HangUp 结束状态: kCallIdle

起始状态: kCallReorder

动作: TTelephoneCallHandle::HangUp 结束状态: kCallIdle

起始状态: kRingbackDectected

事件: kCallActive 结束状态: kCallActive

事件: kNoAnswer 结束状态: kNoAnswer

动作: TTelephoneCallHandle::HangUp 结束状态: kCallIdle

起始状态: kBusyDetected

动作: TTelephoneCallHandle::HangUp 结束状态: kCallIdle

起始状态: kCallCannotComplete

动作: TTelephoneCallHandle::HangUp 结束状态: kCallIdle

起始状态: TTelephoneLine::kCallActive

事件: kCallIdle 结束状态: kCallIdle

动作: TTelephoneCallHandle::HangUp 结束状态: kCallIdle

起始状态: TTelephoneLine::kNoAnswer

动作: TTelephoneCallHandle::HangUp 结束状态: kCallIdle

虽然已经以特定系统环境中的最佳实施例描述了本发明，熟悉本技术的人员应能认识到在所附的权利要求书的精神与范围内的其它与不同的硬件与软件环境中可通过修改上述实施例来实践本发明。

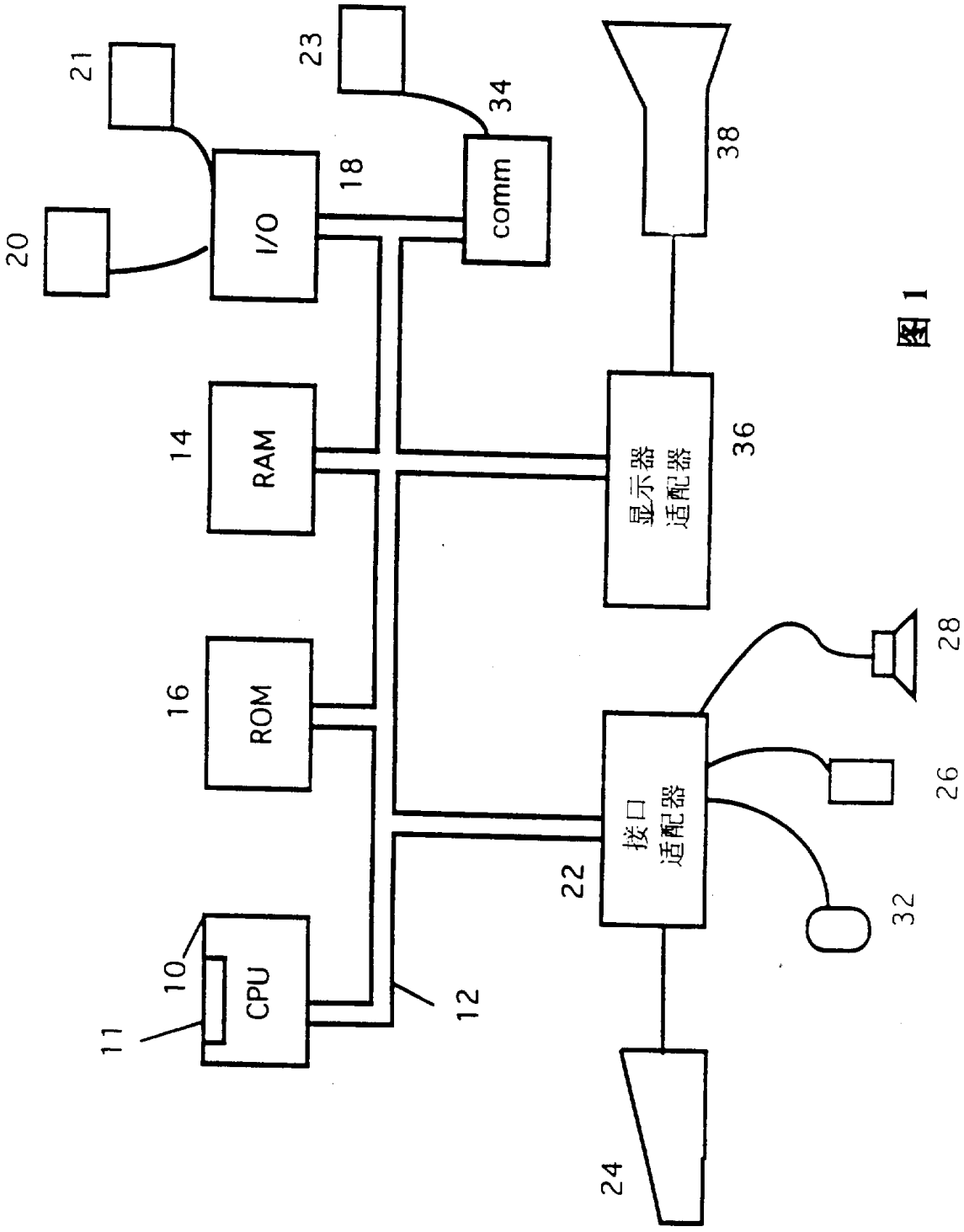

CPU
ROM
RAM
I/O
comm
显示器
适配器
接口
适配器
10
11
12
14
16
18
20
21
22
23
24
26
28
32
34
36
38

图 1

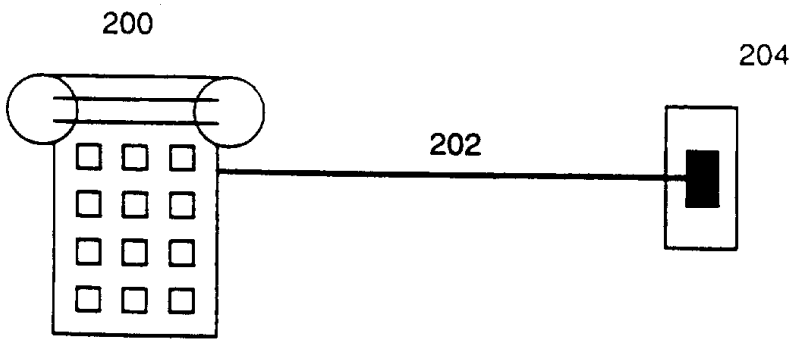


图 2

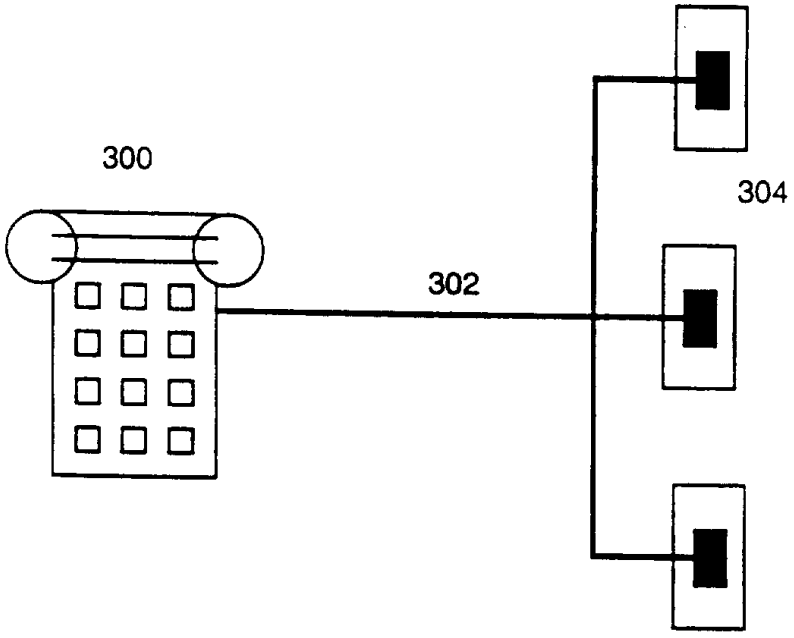


图 3

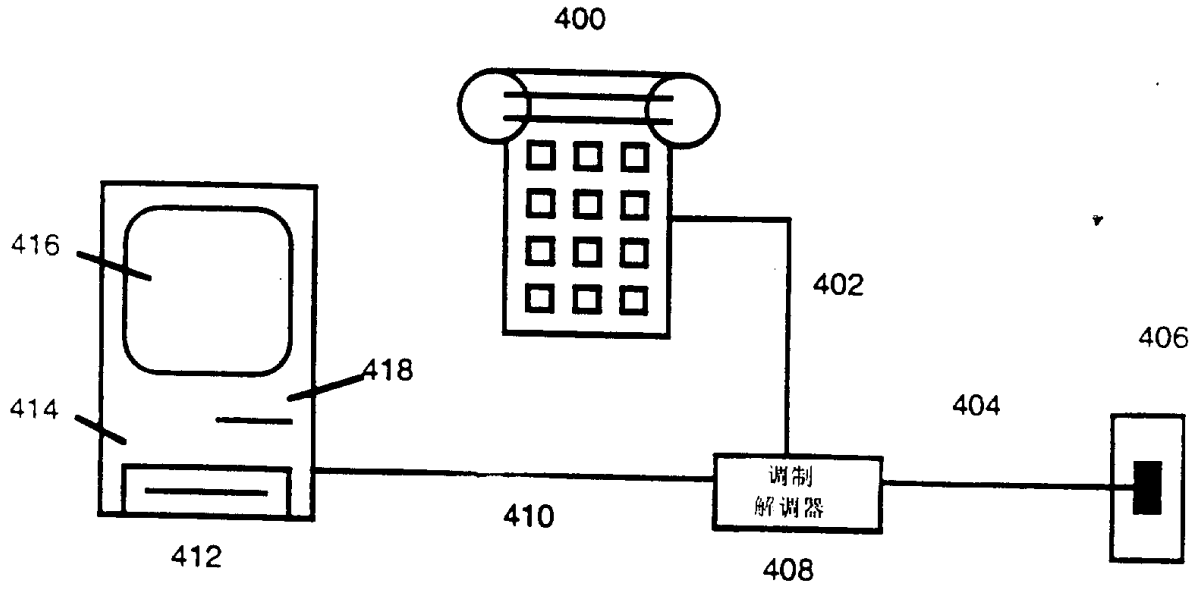


图 4

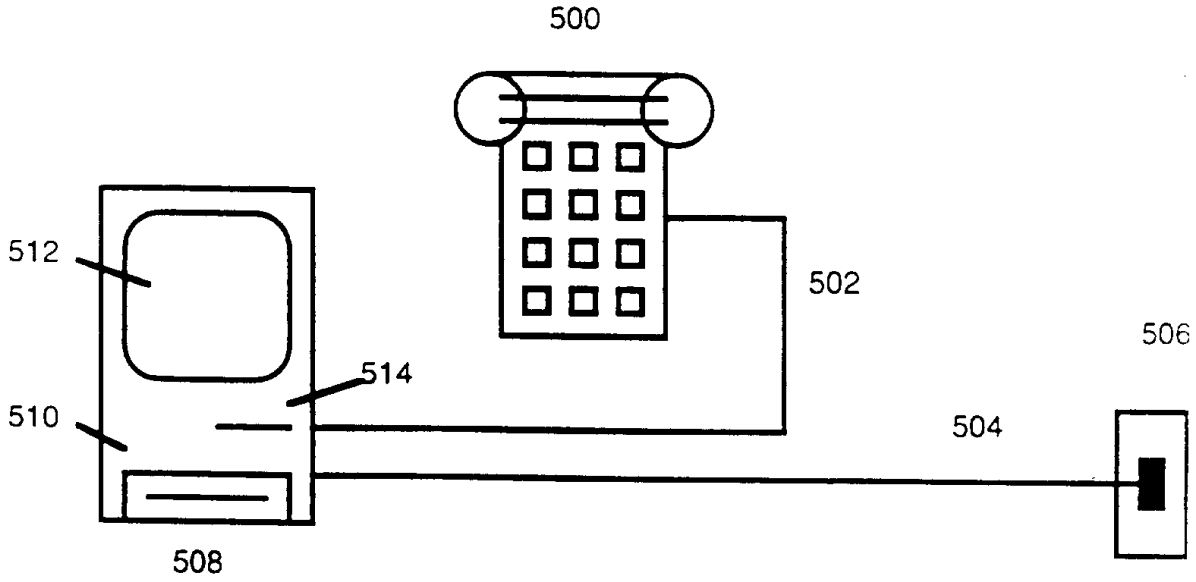


图 5

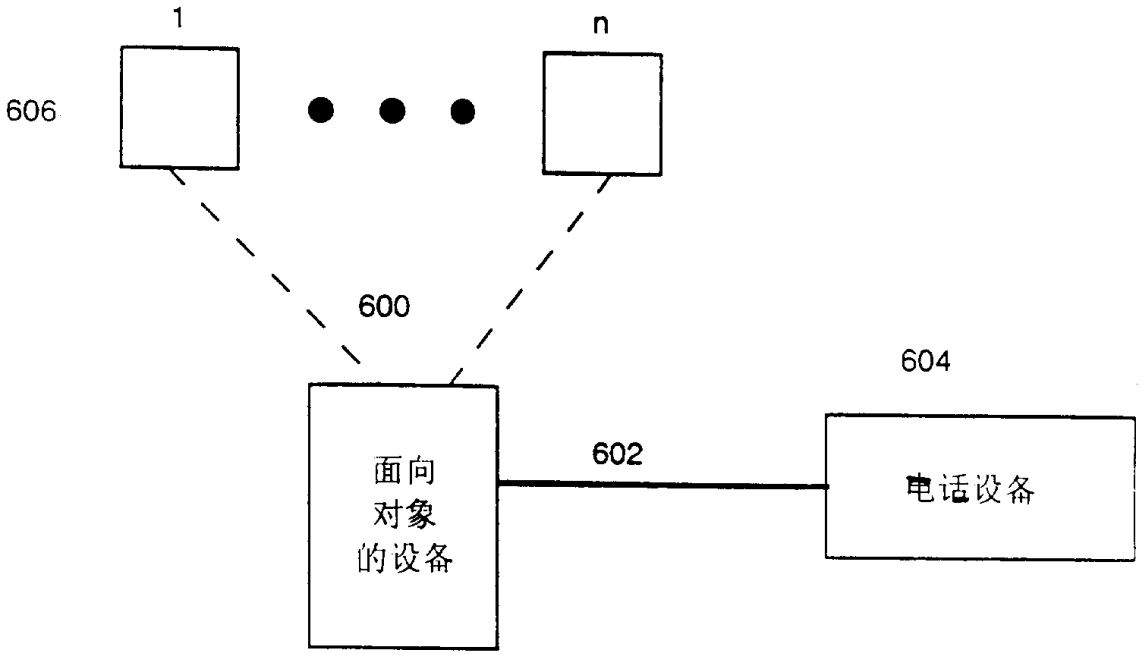


图 6

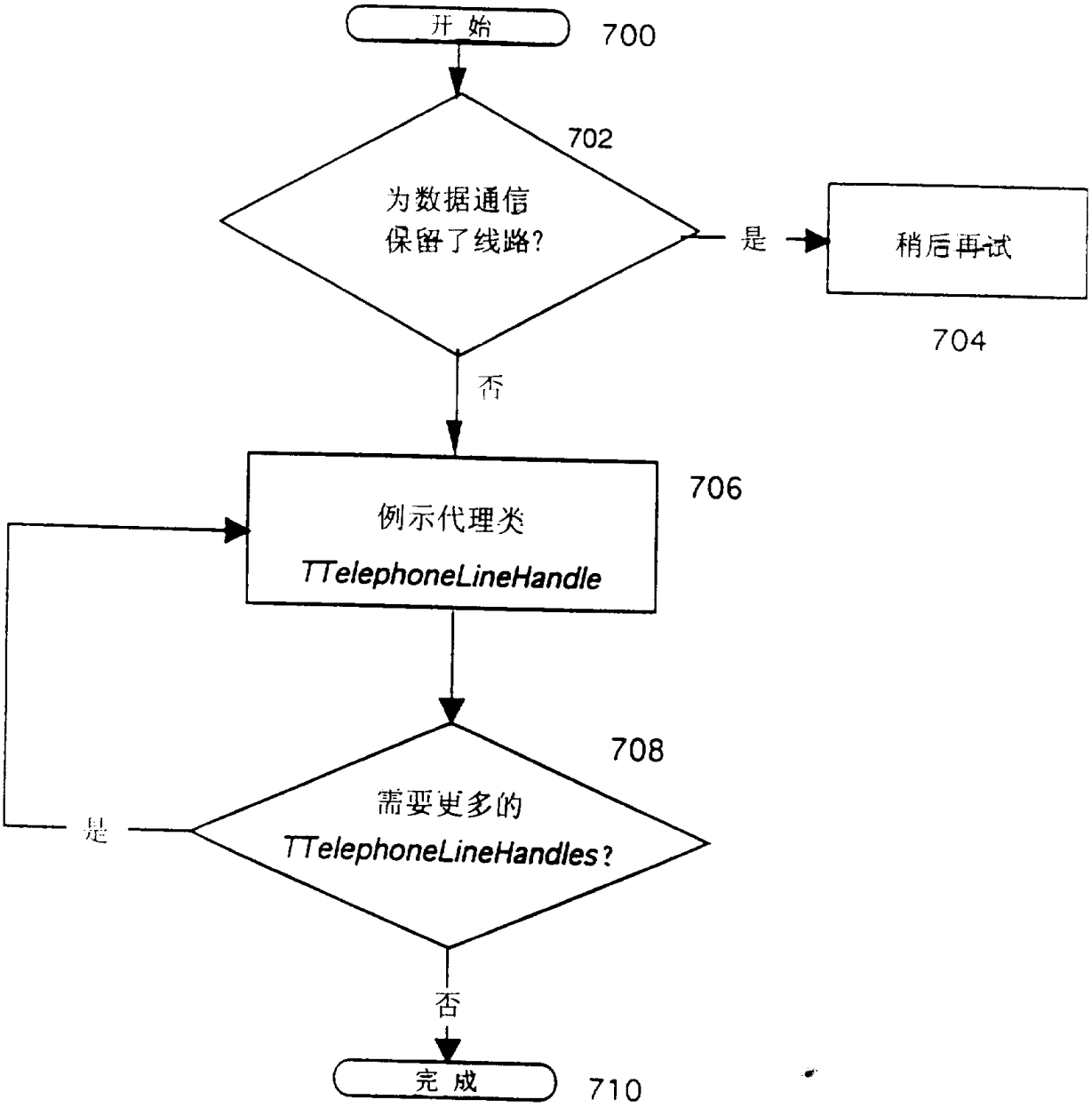

开 始
700
702
为数据通信
保留了线路?
是
稍后再试
704
否
例示代理类
TTelephoneLineHandle
706
708
需要更多的
TTelephoneLineHandles?
是
否
完 成
710

图 7

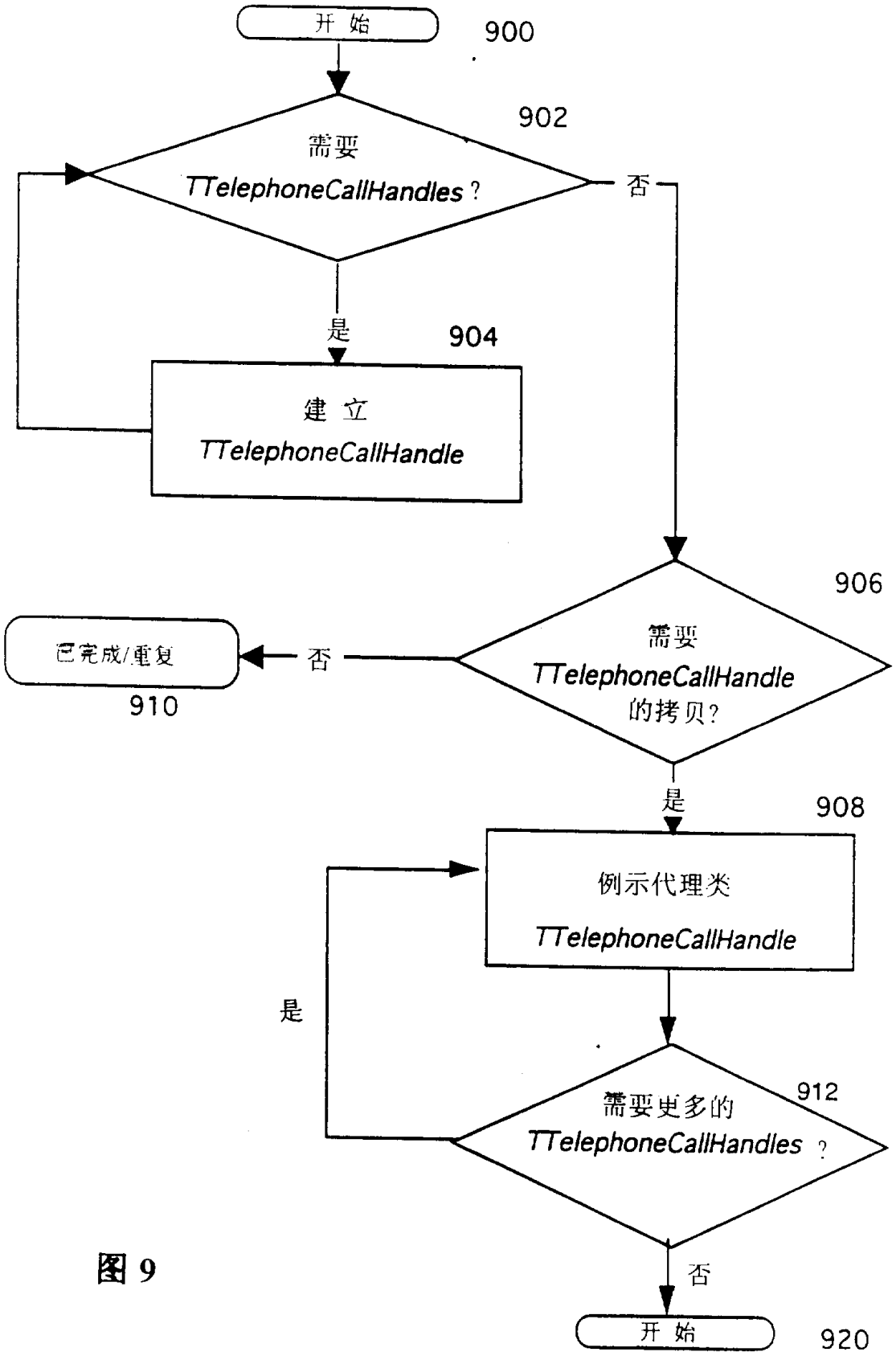

开 始
900
902
需要
TTelephoneCallHandles ?
否
是
904
建 立
TTelephoneCallHandle
906
需要
TTelephoneCallHandle
的拷贝?
已完成/重复
否
910
是
908
例示代理类
TTelephoneCallHandle
是
912
需要更多的
TTelephoneCallHandles ?
否
开 始
920

图 9

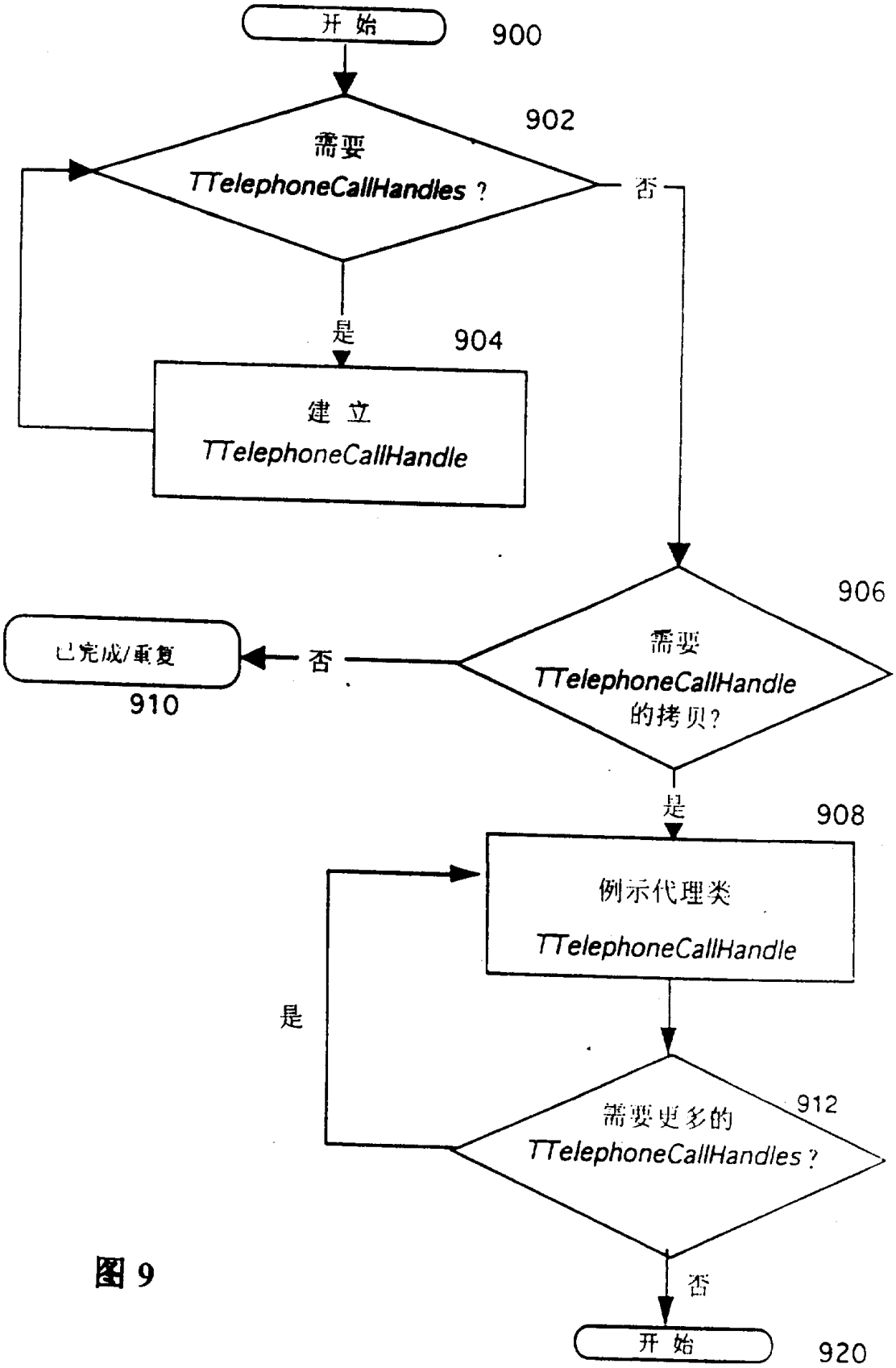

开 始
900
902
需要
TTelephoneCallHandles ?
否
是
904
建 立
TTelephoneCallHandle
906
需要
TTelephoneCallHandle
的拷贝?
否
已完成/重复
910
是
908
例示代理类
TTelephoneCallHandle
是
912
需要更多的
TTelephoneCallHandles ?
否
开 始
920

图 9

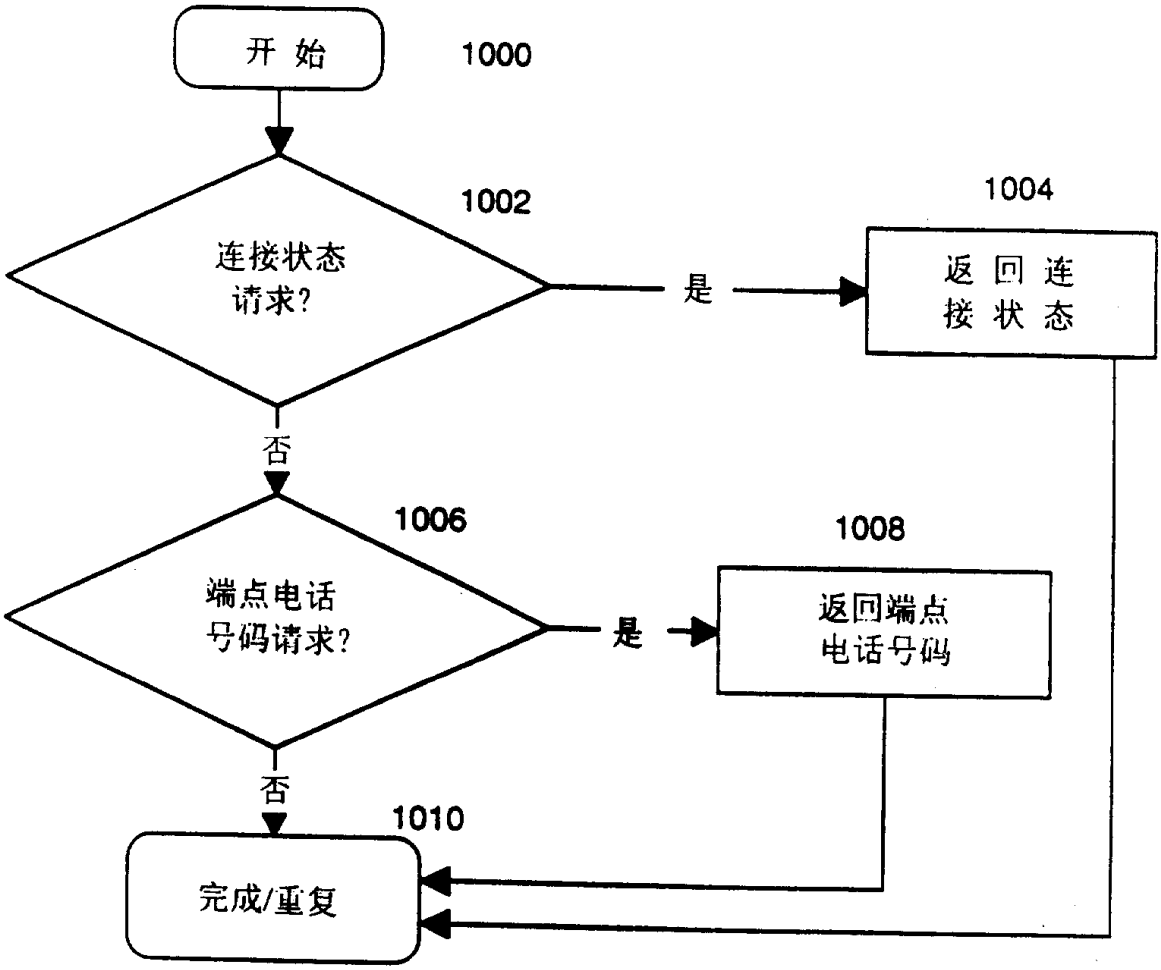

开 始
1000
1002
连接状态
请求?
是
1004
返 回 连
接 状 态
否
1006
端点电话
号码请求?
是
1008
返回端点
电话号码
否
1010
完成/重复

图 10

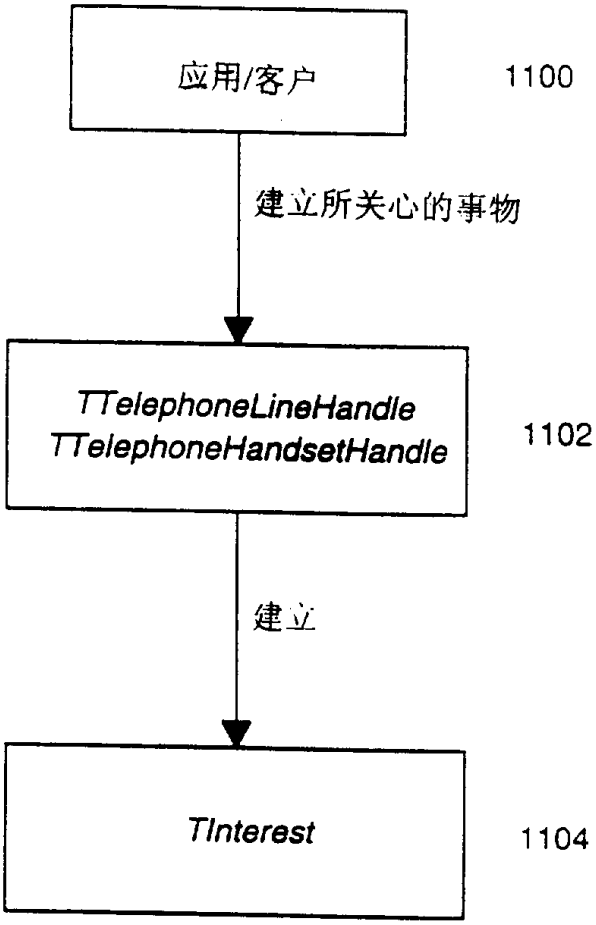

应用/客户
1100
建立所关心的事物
TTelephoneLineHandle
TTelephoneHandsetHandle
1102
建立
TInterest
1104

图 11

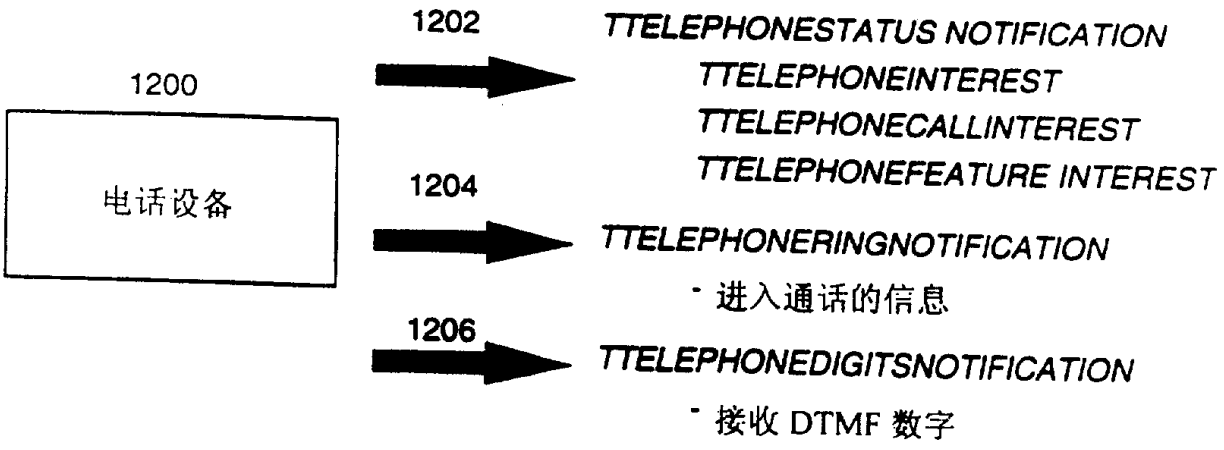

1200
电话设备
1202
TTELEPHONESTATUS NOTIFICATION
TTELEPHONEINTEREST
TTELEPHONECALLINTEREST
TTELEPHONEFEATURE INTEREST
1204
TTELEPHONERINGNOTIFICATION
- 进入通话的信息
1206
TTELEPHONEDIGITSNOTIFICATION
- 接收 DTMF 数字

图 12

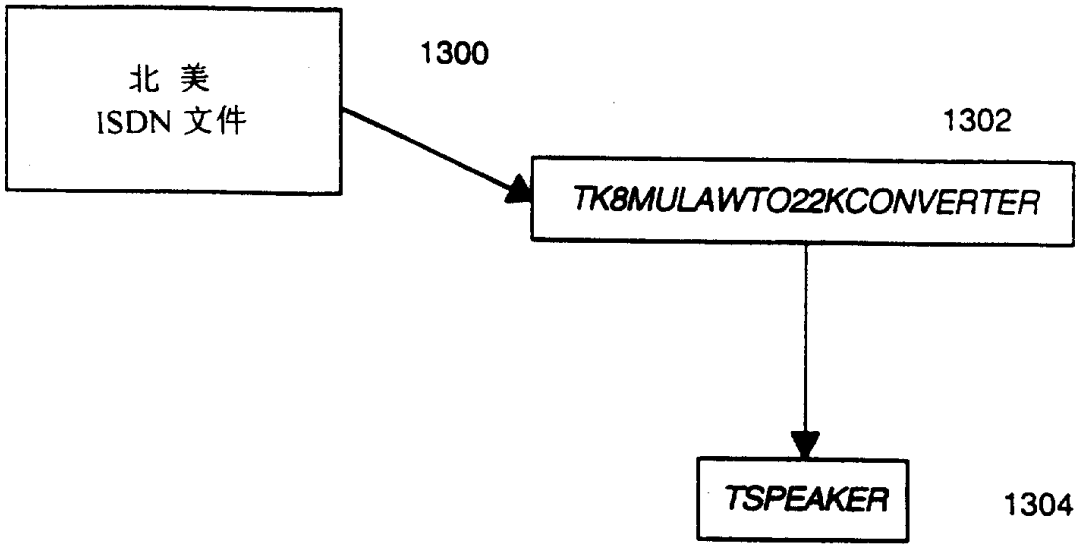

北 美
ISDN 文件
1300
1302
TK8MULAWTO22KCONVERTER
TSPEAKER
1304

图 13

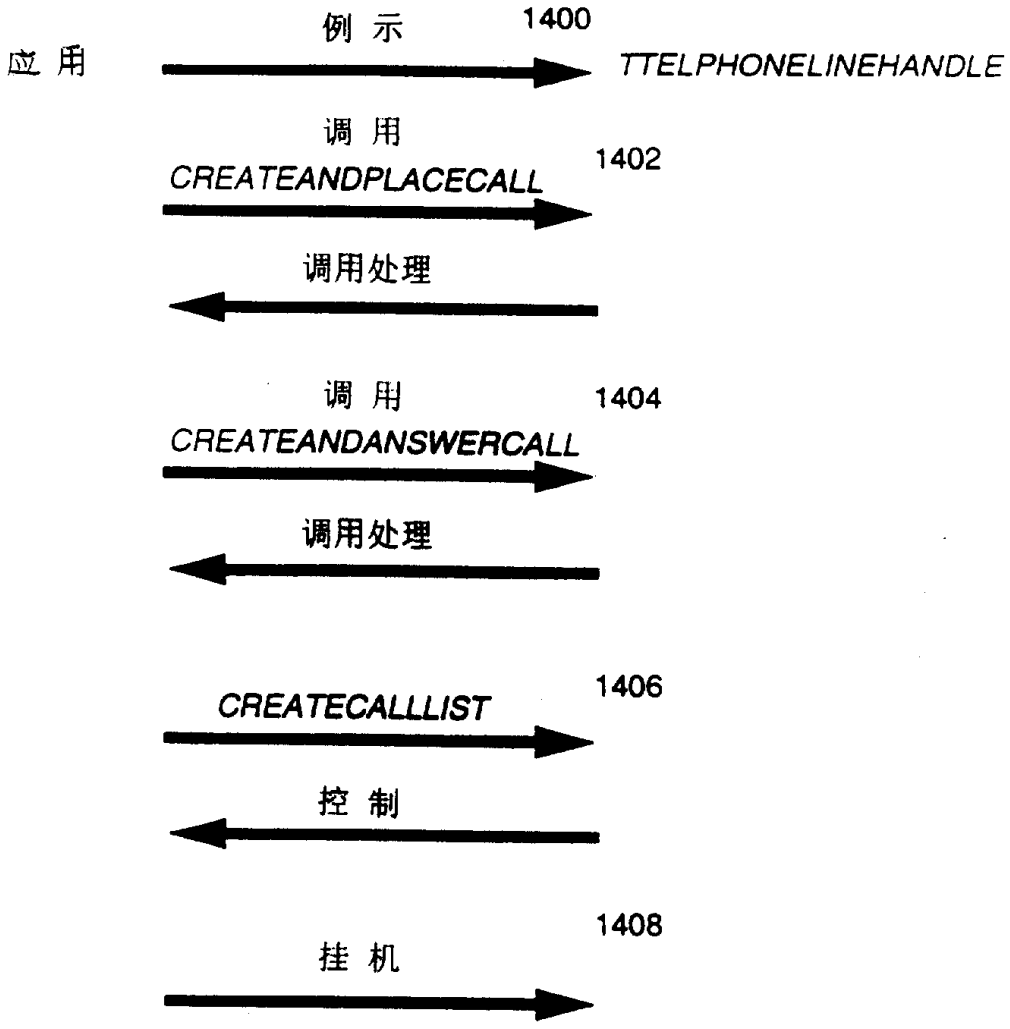

应用
例示
1400
TTELPHONELINEHANDLE
调用
CREATEANDPLACECALL
1402
调用处理
调用
CREATEANDANSWERCALL
1404
调用处理
CREATECALLLIST
1406
控制
1408
挂机

图 14

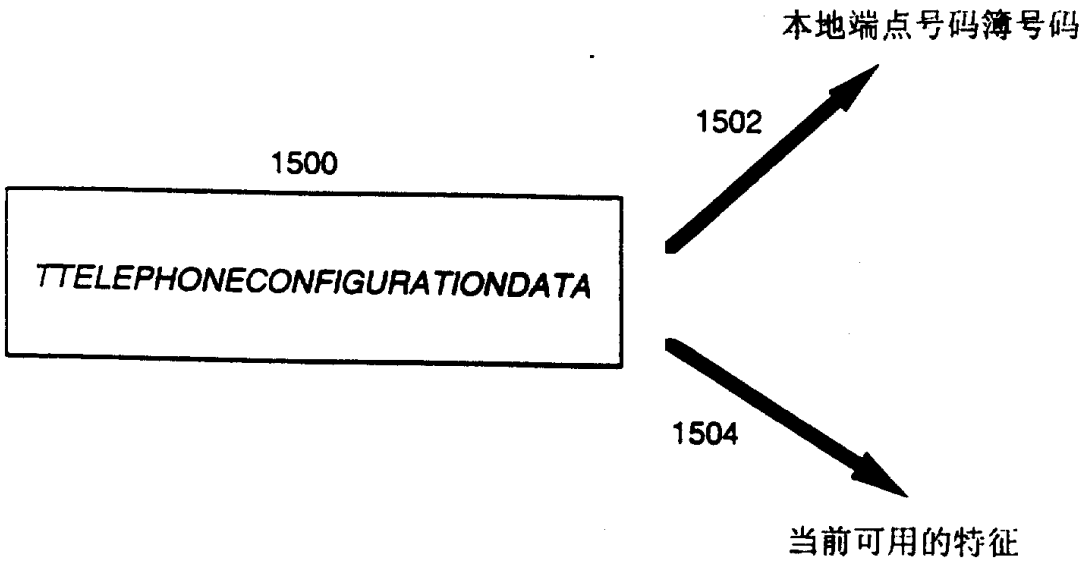

本地端点号码簿号码
1502
1500
TTELEPHONECONFIGURATIONDATA
1504
当前可用的特征

图 15

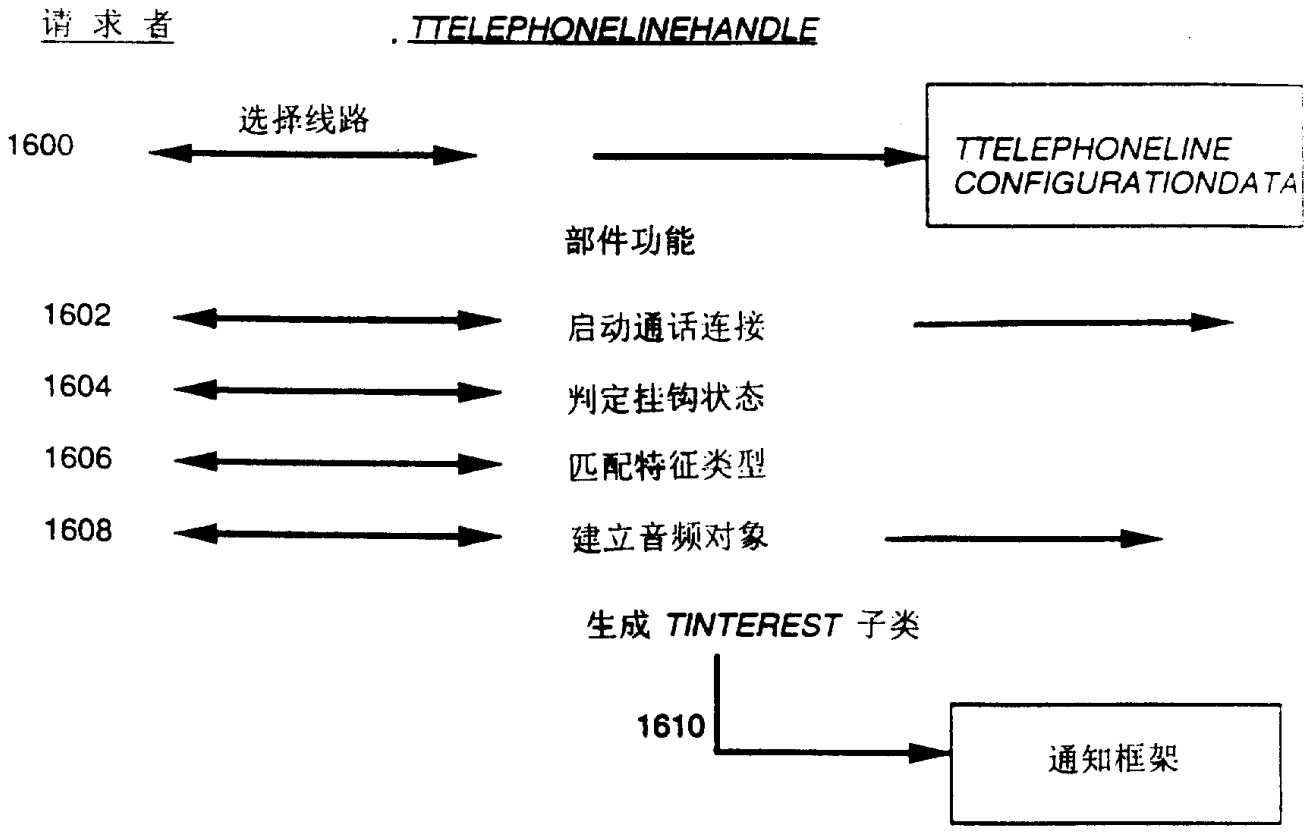

请求者
.TTELEPHONELINEHANDLE
选择线路
1600
TTELEPHONELINE CONFIGURATIONDATA
部件功能
1602
启动通话连接
1604
判定挂钩状态
1606
匹配特征类型
1608
建立音频对象
生成 TINTEREST 子类
1610
通知框架

图 16

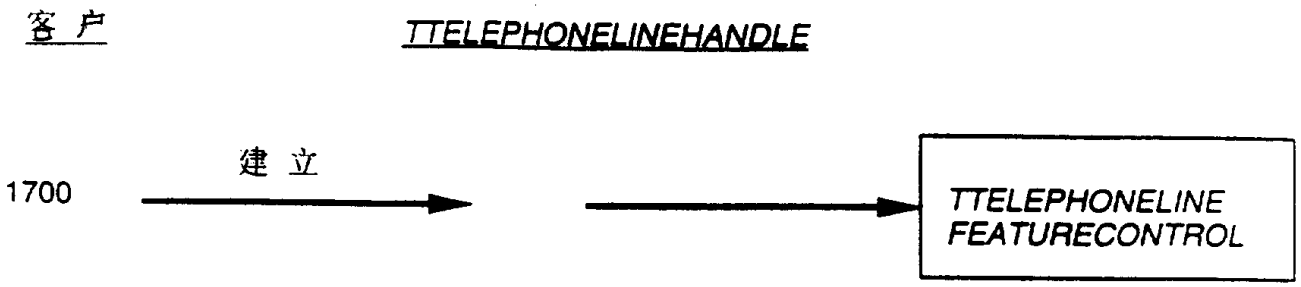

客户
TTELEPHONELINEHANDLE
1700
建立
TTELEPHONELINE
FEATURECONTROL

图 17

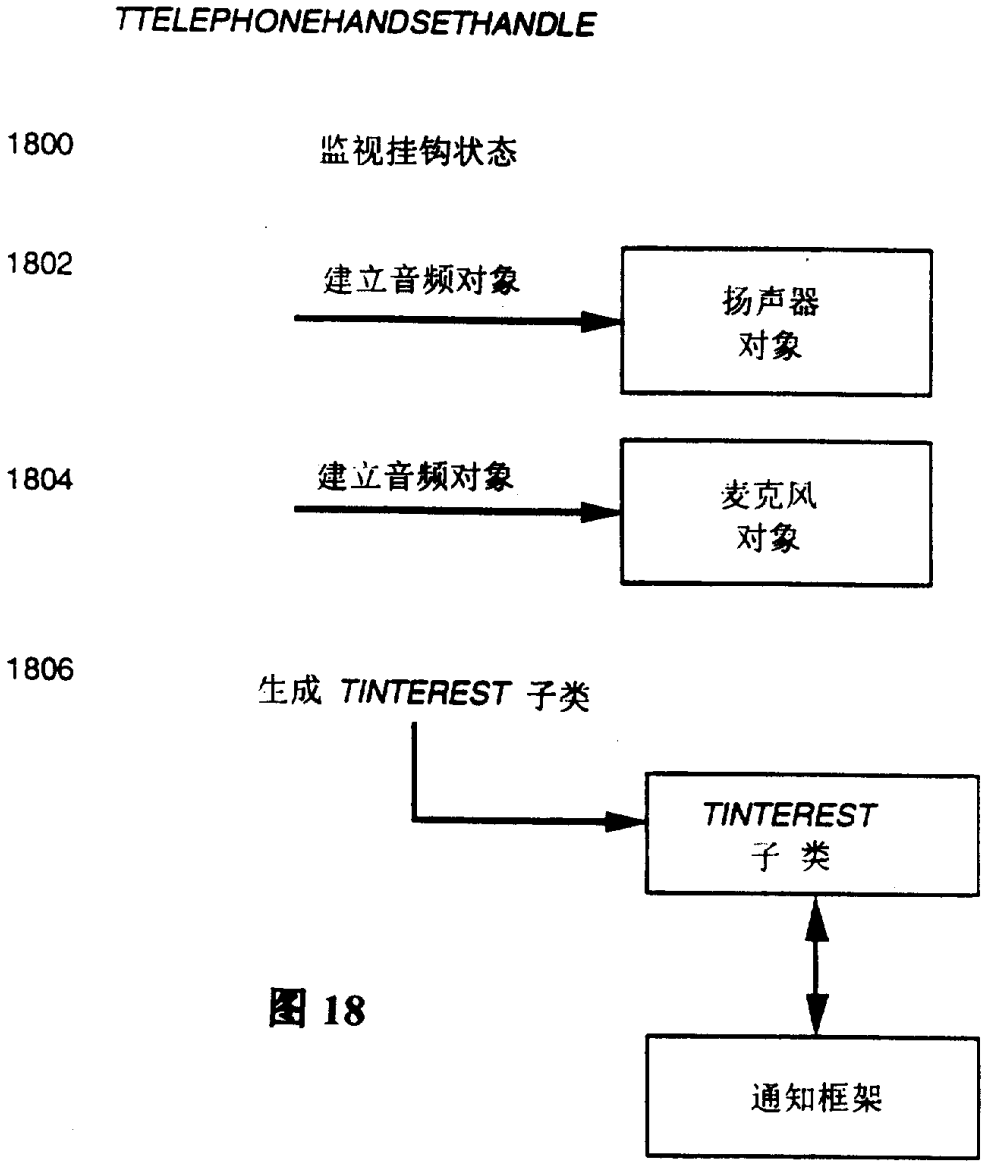

TTELEPHONEHANDSETHANDLE
1800
监视挂钩状态
1802
建立音频对象
扬声器
对象
1804
建立音频对象
麦克风
对象
1806
生成 TINTEREST 子类
TINTEREST
子 类
通知框架

图 18

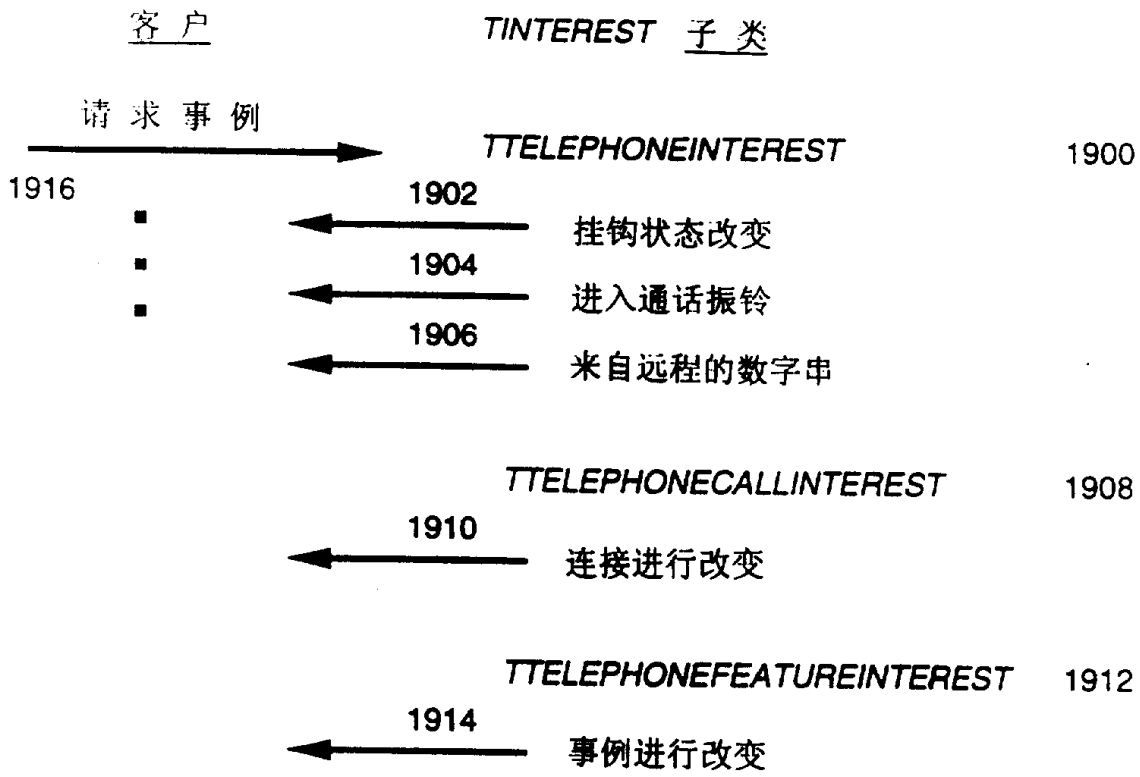

客户
TINTEREST 子类
请求事例
TTELEPHONEINTEREST
1900
1916
1902
挂钩状态改变
1904
进入通话振铃
1906
来自远程的数字串
TTELEPHONECALLINTEREST
1908
1910
连接进行改变
TTELEPHONEFEATUREINTEREST
1912
1914
事例进行改变

图 19

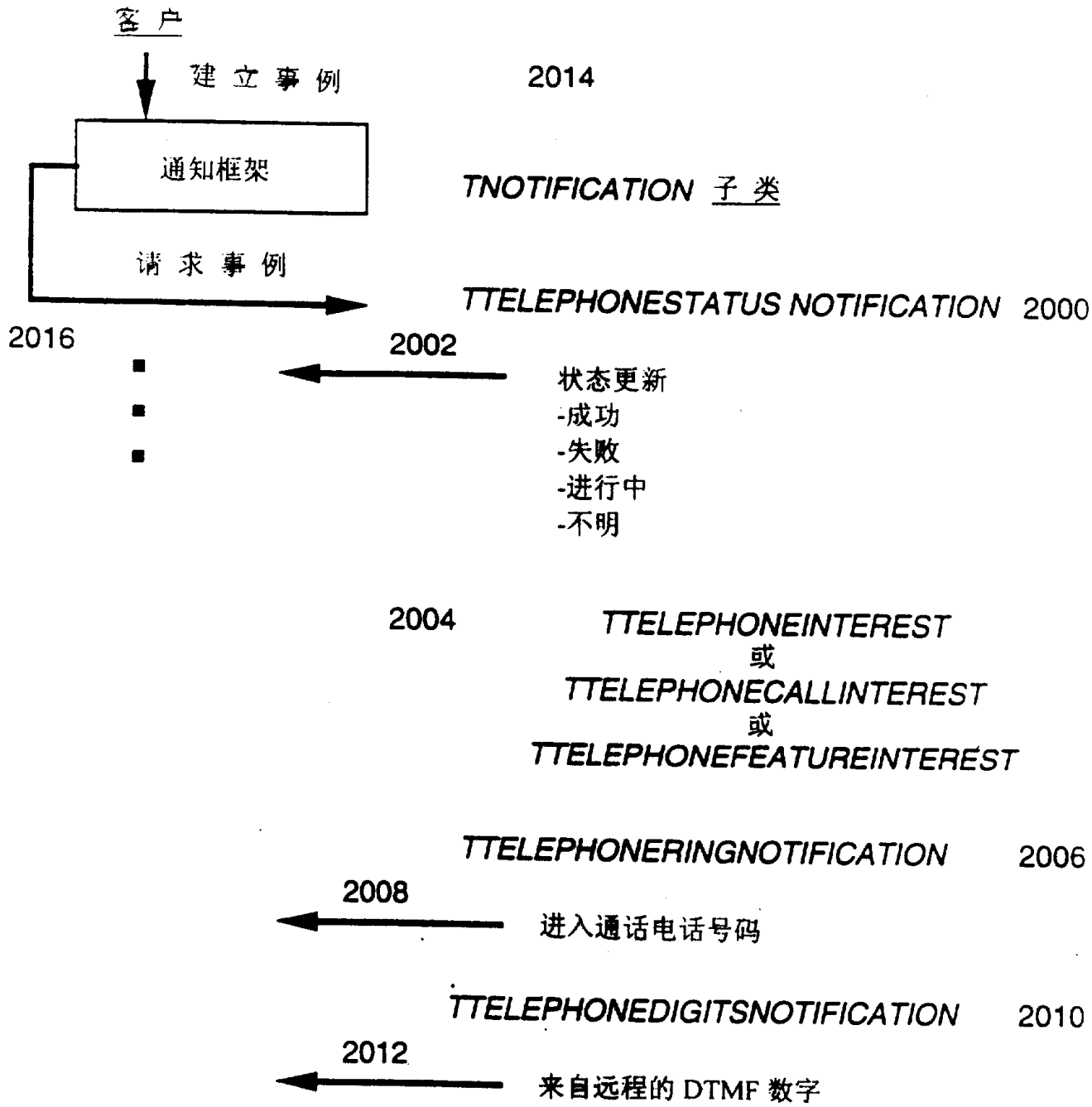

客户
建立事例
2014
通知框架
TNOTIFICATION 子类
请求事例
TTELEPHONESTATUS NOTIFICATION 2000
2016
2002
状态更新
-成功
-失败
-进行中
-不明
2004
TTELEPHONEINTEREST
或
TTELEPHONECALLINTEREST
或
TTELEPHONEFEATUREINTEREST
TTELEPHONERINGNOTIFICATION 2006
2008
进入通话电话号码
TTELEPHONEDIGITSNOTIFICATION 2010
2012
来自远程的 DTMF 数字

图 20

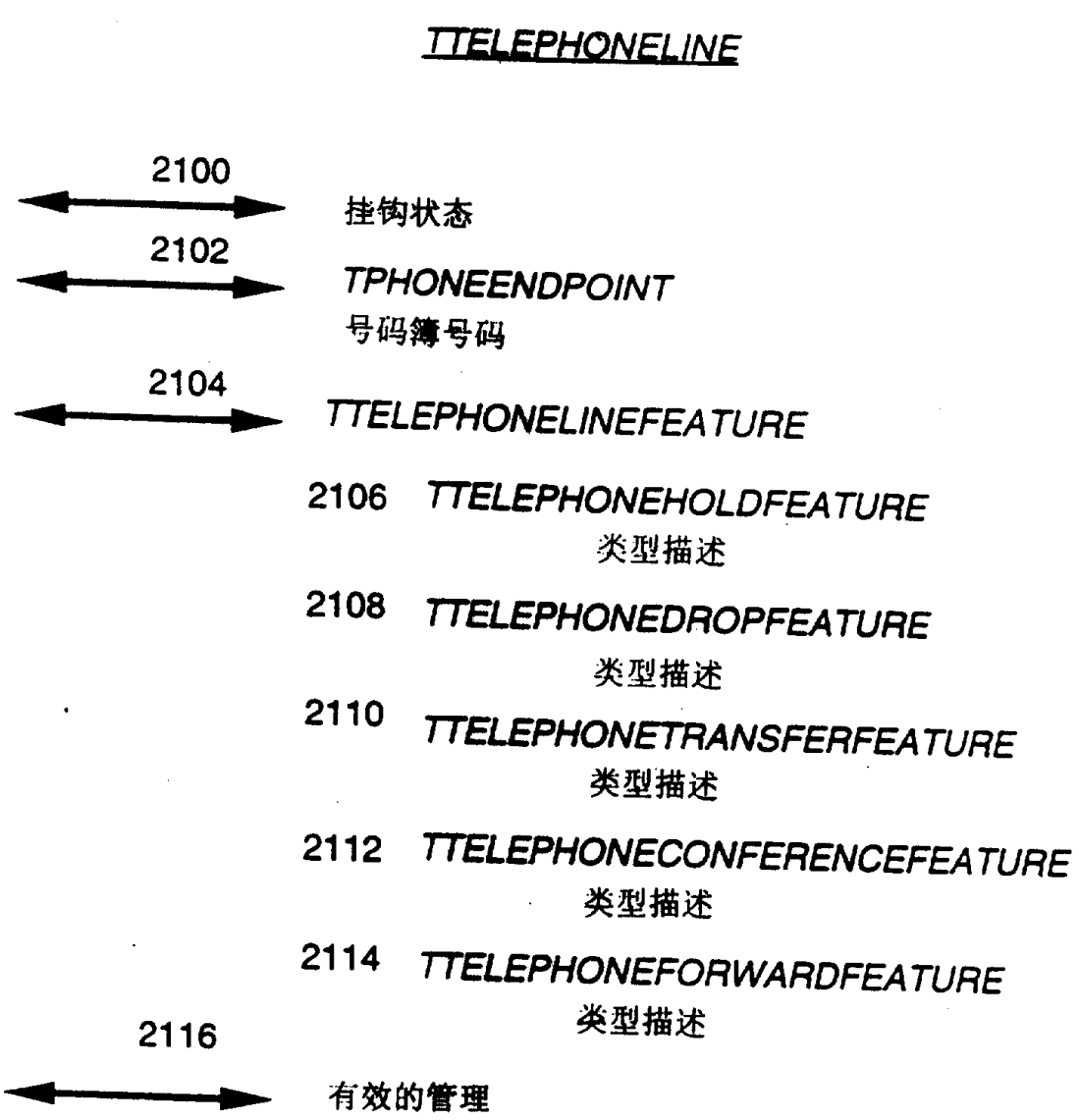

TTELEPHONELINE
2100
挂钩状态
2102
TPHONEENDPOINT
号码簿号码
2104
TTELEPHONELINEFEATURE
2106 TTELEPHONEHOLDFEATURE
类型描述
2108 TTELEPHONEDROPFEATURE
类型描述
2110 TTELEPHONETRANSFERFEATURE
类型描述
2112 TTELEPHONECONFERENCEFEATURE
类型描述
2114 TTELEPHONEFORWARDFEATURE
类型描述
2116
有效的管理

图 21

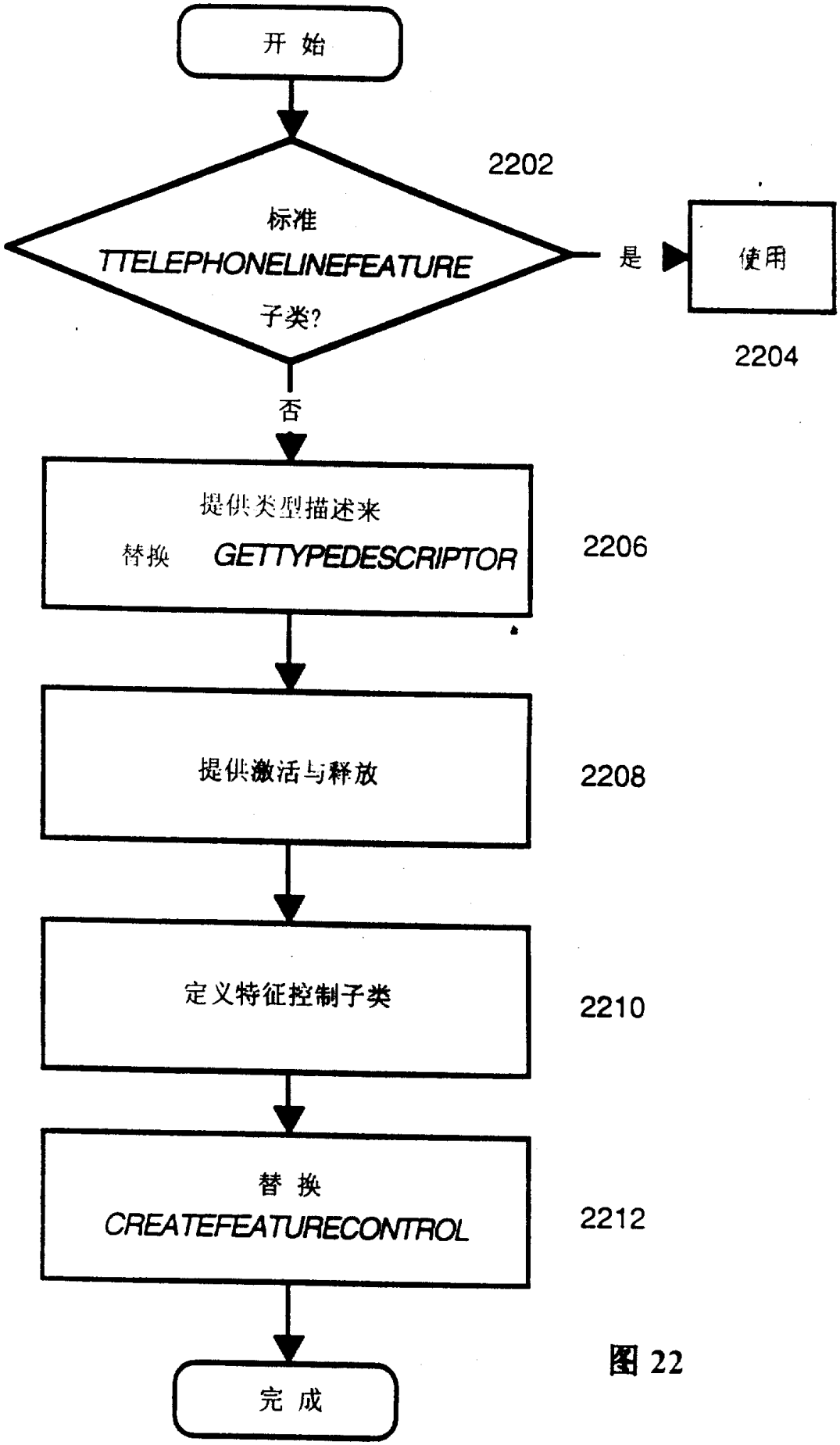

开 始
2202
标准
TTELEPHONELINEFEATURE
子类?
是
使用
2204
否
提供类型描述来
替换 GETTYPEDESCRIPTOR
2206
提供激活与释放
2208
定义特征控制子类
2210
替 换
CREATEFEATURECONTROL
2212
完 成

图 22

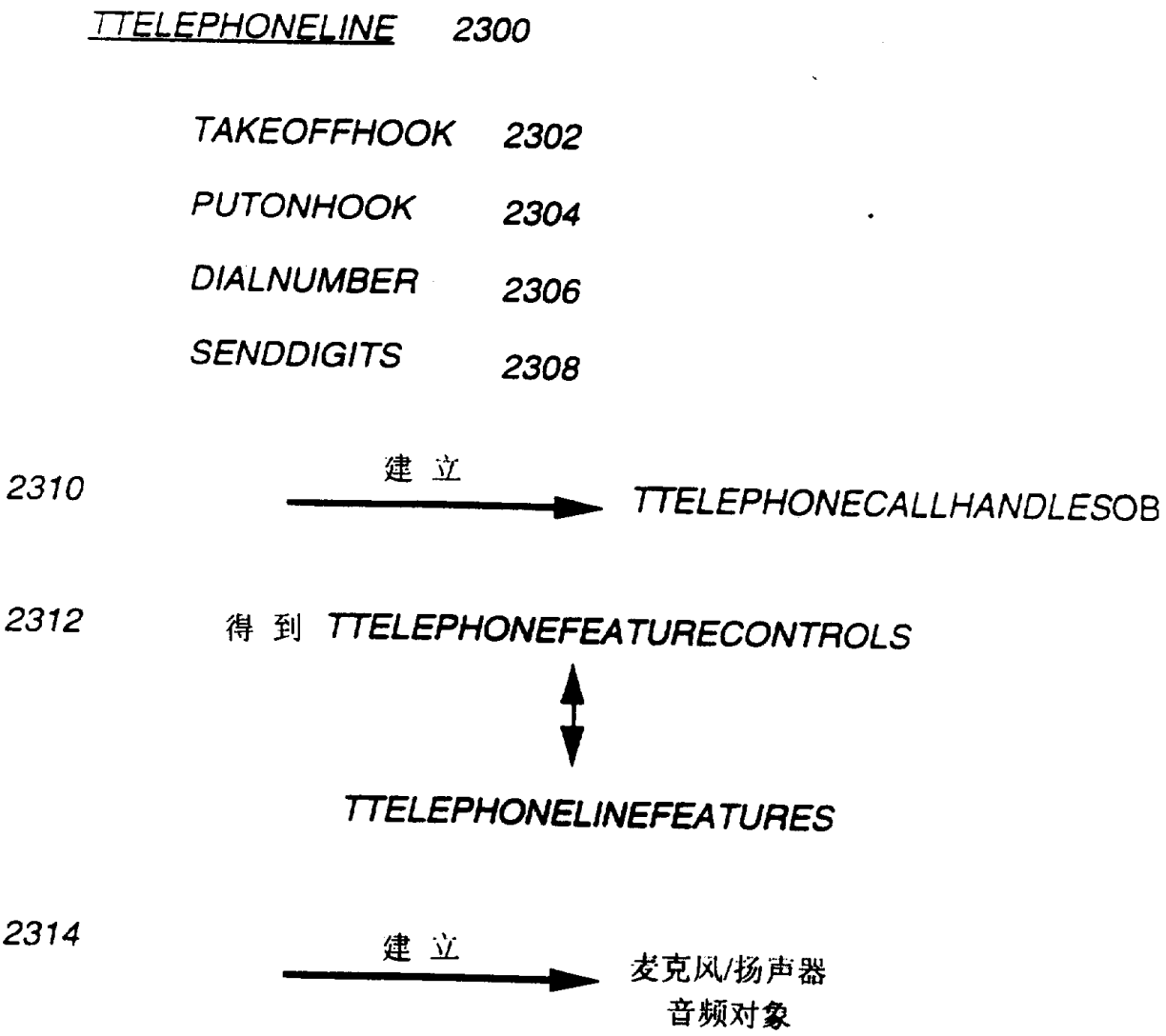

TTELEPHONELINE 2300
TAKEOFFHOOK 2302
PUTONHOOK 2304
DIALNUMBER 2306
SENDDIGITS 2308
2310
建 立
TTELEPHONECALLHANDLESOB
2312
得 到 TTELEPHONEFEATURECONTROLS
TTELEPHONELINEFEATURES
2314
建 立
麦克风/扬声器
音频对象

图 23

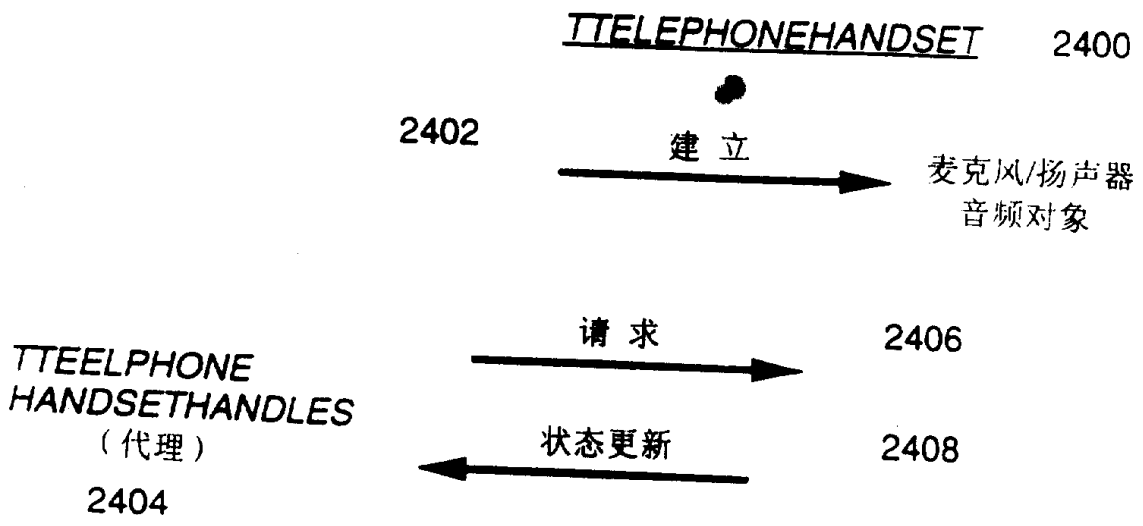

TTELEPHONEHANDSET
2400
2402
建 立
麦克风/扬声器
音频对象
TTEELPHONE
HANDSETHANDLES
（代理）
2404
请 求
2406
状态更新
2408

图 24

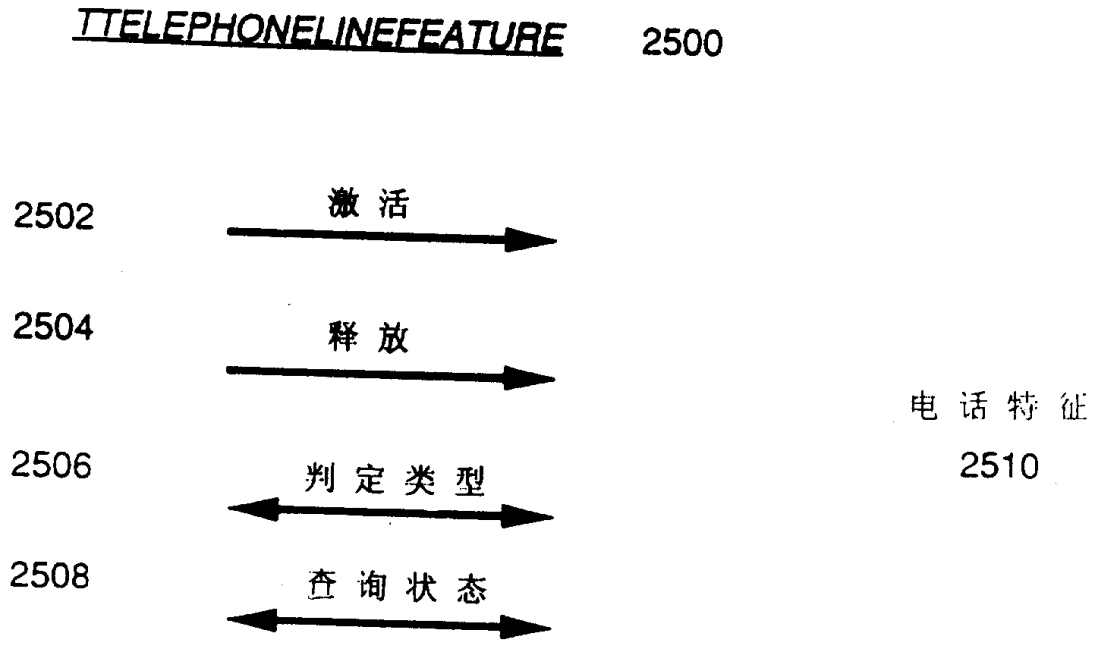

TTELEPHONELINEFEATURE
2500
2502
激 活
2504
释 放
电 话 特 征
2510
2506
判 定 类 型
2508
查 询 状 态

图 25