[19]中华人民共和国专利局

[11] 公开号 CN 1113625A

[12] 发明专利申请公开说明书

[21]申请号 94119530.9

[51]Int.Cl[6]

[43]公开日 1995 年 12 月 20 日

H04L 29/10

[22]申请日 94.12.17

[30]优先权

[32]93.12.20[33]US[31]170,139

[71]申请人 美国电报电话公司

地址 美国纽约

[72]发明人 迈克·P·卡普兰 陆慧兰

李·斯拉兹曼

[74]专利代理机构 中国国际贸易促进委员会专利商标事务所

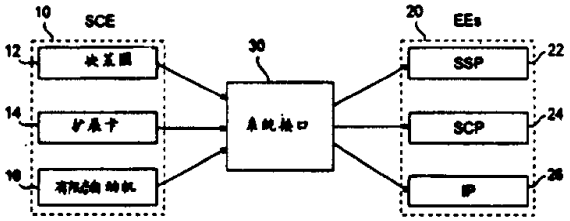
代理人 杨国旭

说明书页数: 附图页数:

[54]发明名称 面向反应的电信系统接口

[57]摘要

本发明提供了电信系统中用于接口不同业务生成环境（SCE）和执行环境（EE）的方法和装置。本发明的接口包括：接口到多个业务 EE 的一个 SCE；分析 SCE 的输出码以形厉分析树的分析程序，产生表示分析树节点的中间码的中间码生成器，以使中间码保存包含在 SCE 输出码的基本上所有信息；以及从中间码生成用于每个 EE 目标码的目标码生成器，从而在 SCE 中开发的电信业务可以提供给每个 EE。



权 利 要 求 书

1. 用于接口电信业务生成环境和业务执行环境的方法，包括下列步骤：

标识要被接口的一组所选择的业务生成环境；

定义一组适于表示在所述选择的业务生成环境中开发的电信业务的中间码操作；

分析来自所述选择的一个业务生成环境的输出码，以形成具有多个节点的分析树，所述分析树表示在所述业务生成环境中开发的所述电信业务之一；

根据所述的一组中间码操作和所述分析树，生成用以表示所述分析树所述节点的中间码，以使所述中间码保存在所述业务生成环境的输出码所包含的基本上所有信息；以及

根据所述中间码，为每个执行环境生成一个目标码，从而在所述选择的业 务 生成环境之一上开发的所述电信业务可以在每个所述执 行环境内提供。

2. 根据权利要求 1 的方法，其中用于生成中间码的步骤进一步包括，生成具有多个字节组的中间码。每个所述字节组对应于所述输出码分析树的一个节点。

3. 根据权利要求 1 的方法，其中所述生成中间码的步骤进一

步包括，生成能够表示所述分析树的语句节点的中间码。

4.根据权利要求1的方法，其中生成中间码的所述步骤进一步包括，生成能够表示所述分析树的说明节点的中间码。

5.根据权利要求1的方法，其中分析输出节点，生成中间码和生成一目标码的步骤，进一步包括步骤：

分析所述业务生成环境输出码，以产生第一传送码；

使用所述第一传送码和储存在第二传送数据库的第二传送宏指令，生成第二传送码，所述第二传送码对应于所述中间码；

使用所述第二传送码和储存在第三传递数据库的第二传送码，生成第三传送码，所述第三传送码对应于述执行环境的所述目标码。

6.根据权利要求2的方法，其中所述业务生成环境是一个图形编辑程序，它使用多个相互关联的符号以图形表示法的方式描述所述电信业务。

7.根据权利要求6的方法，其中用在所述图形编辑程序的每个所述符号对应于所述中间码的每个所述字节组。

8.根据权利要求7的方法，其中所述字节组中的一个操作符的操作数，指示出一个所述符号与所在所述图形编辑程序的所述图形表示法的其它符号之间的相互关系。

9.根据权利要求6的方法，其中所述图形编辑程序是一个决策图编辑程序，其使用由分支相互连接的许多决策图节点，以决策

图的方式表示所述电信业务。

10. 根据权利要求 9 的方法，其中所述决策图节点之一是百分比节点，并且所述字节组包括每个所述分支的一对操作数，表示连接到一个特定分支的特定决策图节点的相似程度。

11. 根据权利要求 9 的方法，其中所述决策图节点之一是时间节点，并且所述字节组包括用于指示时间的操作数，在该时间后，一个特定的决策图节点将被连接到一个特定的分支。

12. 根据权利要求 7 的方法，其中所述图形编辑程序是一个有限态自动机编辑程序。

13. 根据权利要求 1 的方法，其中所述中间码被用以生成适用于在所述业务执行环境中直接执行的业务逻辑语言码。

14. 根据权利要求 1 的方法，其中所述电信业务是电子表决业务。

15. 根据权利要求 14 的方法，其中所述电子表决业务是在有限态自动机编辑程序中，使用业务规范描述语言来开发的。

16. 电信系统接口包括：

至少一个要接口到多个业务执行环境的业务生成环境；

分析程序，用于分析所述业务生成环境的输出码以形成具有多节点的分析树，所述分析树表示在所述业务生成环境中开发的电信业务；

中间码生成器，用于根据一组中间码操作和所述分析树产生

表示所述分析树的所述节点的中间码，以使所述中间码保存包含在所述业务生成环境输出码中的基本上所有信息；以及

目标码生成器，用于根据所述中间码生成一个目标码，该目标码用于每一所述执行环境，从而在所述业务生成环境开发的所述电信业务可以提供给每个所述执行环境。

17. 根据权利要求 16 的接口，其中所述中间码包括多个字节组，每个所述字节组对应于所述分析树的一个节点。

18. 根据权利要求 16 的接口，其中所述业务生成环境是一个图形编辑程序，它描述以图形表示法表示的所述电信业务。

19. 根据权利要求 16 的接口，其中所述执行环境是一个电信系统业务交换处理器。

20. 根据权利要求 16 的接口，其中所述执行环境是一个电信系统业务控制处理器。

说　　明　　书

面向应用的电信系统接口

本发明通常涉及综合不同的电信系统和业务的技术。特别地，本发明涉及一个标准接口，它使得在一个业务生成环境(*SCE*)中产生的电信业务应用在许多不同的执行环境(*EE*)中。

电信业务可通过执行在诸如电话网络交换机和计算机的电信硬件中的机器码来提供。执行机器码的电信硬件通常指执行环境(*EE*)。比如说，*EE* 可以是发送数据通信网络内的数据信息包或引导电话网络交换系统中的呼叫。一组 *EE* 可以给网络用户提供交互业务，和通过网络，发送数据、文件及图象。示范性的 *EE* 包括：业务切换处理器(*SSP*)，业务节点(*SN*)，智能外设(*IP*)和业务控制点(*SCP*)。它们都可以是一个智能网络(*IN*)体系的组成部分。关于 *IN* 更详细的情况，比如说可在 *Proc. of the International Council for Computer Communication Intelligeat Networks Conference*, *May*4—6, 1992 的第 430—440 页，*O. Miauno* 等人的文献"*Service Specification Description and Services Logic Program Generation for Intelligent Networks*"中发现，该文献在这里用作对比文件。

当前情况下，在每个 *EE* 中所提供的电信业务通常都是在高级编程环境中，使用特定的开发工具编程的。高级编程环境通常指业务生成环境(*SCE*)。示例性的开发工具包括：决策图编辑程序，扩展卡，计算机辅助系统工程(*CASE*)工具和有限态自动机(*FSM*)编辑程序。一种示例性类型的 *FSM* 编辑程序使用标准 *CCITT* 业务规范描述语言(*SDL*)，它以图形格式表示电信业务，便于业务的开发。要在每个 *EE* 中提供的业务通常是使用一个专用 *SCE* 开发的，该专用 *SCE* 在计算机工作站上运行，并且提供一套基于特定用户规范的业务开发工具。*SCE* 通常以由卖方开发的合适格式产生业务描述文件。然后，合适业务描述文件被转换为低级码，该低级码适用于指导一个 *EE* 去提供所希望的业务，因此，通常只能用相同或相关卖方的相应的 *EE*。

当前实际应用中 *SCE* 和 *EE* 的深度耦合存在很多问题。一个问题是，在一个 *SCE* 开发的网络业务通常只能应用在一个相应的 *EE*，不能在不同的 *EE* 中再用。另一个相关问题是，一个卖方 *SCE* 和 *EE* 通常不能与其他卖方 的相接口。了提供一个当前存在的网络业务和许多不同 *EE* 的接口，因此需要重新设计现存业务以实现该接口。由于这些问题，网络提供者，如本地交换机公司(*LEC*)，长途公司和国外邮政、电话和电报公司(*PTT*)都有困难以即时可靠的方式提供电信业务。这些问题比如在 *Proc. of the International Council for Gomputer Communciation Intelligent Net-*

*works Conference, May*4—6，1992 第 162—174 页，*D. Singh and D. Garrison* 的文献“*Requirements for Setvice Creation Platform in an Open Systems Enviroment*”中有描述。上述文献在这里用作对比文件。

几种不同的解决 *SCE—EE* 接口问题的方法已被发现。一种可能的方法涉及使用一个通用的面向应用的语言（*UAOL*）设计编程所有 *SCE* 中的业务。一个相关的方法涉及使用一个标准化的业务执行处理器（*SEP*）。在 *Proc. of the International Council for Computer Communication Intelligent Networks Conference, May*4—6，1992 的第 441—450 页，*S. Esaki* 等人的文献“*Service Logic Execution of IN Multi—Vendor Enviroment —Introduction of SLP Execution Processor*”中公开了一个示范性标准 *SEP*。该文献在这里用作对比文件。虽然如果电信工业最初建立在一个标准的 *UAOL* 或 *SEP* 上，上述这些方法是可施行的，但这种标准当前并没被广泛使用。另外，能够适用于多种业务应用的 *UAOL* 通常是一个复杂的高级语言，因此不容易使用。目前要实现 *UAOL* 或标准 *SEP* 方法可能将需要重新设计使用非标准语言或处理器编程的网络业务。

另一种接口能在不同的 *SCE* 和 *EE* 之间转换目标码。目标码的转换通常涉及把源目标码转换为通用的汇编语言，然后以汇编语言产生最终的目标码。目标码转换对于业务使用者具有透明的优点，

并且具有特定 *EE* 要求的用于生成低级码的有效装置。但是，这种方法在应用中不容易于改变，并且不能很好地适用于运行时解释的程序。

在现有技术电信系统接口中所使用的技术是交叉编译。通常，交叉编译涉及设计一个不同的编译器，用于有每个 *SCE* 和每个 *EE* 之间转换代码。对于每个接口使用分离的编译器，任何 *SCE* 中开发的业务可以在任何其它的 *EE* 上使用，因此避免了上述很多问题。但是，这种方法在应用上非常昂贵，它包括大量的 *SCE* 和 *EE*。编译器开发的造价通常是 *SCE* 和 *EE* 数量的二次方。例如，如果电信系统包括 m 个 *SCE* 和 n 个 *EE*，交叉编译方法通常需要开发和支持 $m \times n$ 个编译器。当前使用在高级编程语言中的接口技术是基于公用中间语言的。这种在 1961 年使用的方法用以开发一种通用计算机有向语言(*UNCOL*)。参看 1961 年 *Proc. Western Joint Comp. Conf.* 第 371—377 页，*To. Steel* 的文献“*A First Version of UNCOL*”。公用中间语言通常包括 n 个前端，每个前 端把编程语言中的高级码转换为中间语言；它还包括 m 个后端，每个后端把公用中间语言的程序转换为特定的机器语言或目标码。结果，只需要 $n+m$ 个编译，就可把 n 个 *SCE* 的每一个与 m 个 *EE* 的每一个连接起来，而不需要交叉编译方法中使用的 $n \times m$ 编译器。例如，参看 *Communications of the ACM, Vol*, 26, *No.* 9, *September* 1983 中，*A. Tanenbaum* 等人的文献“*A Practical Todkit for Making*

Portable Compilers"(后面称"*Tanenbaum*")。一个相似的方法被用在美国专利4,667,290号的文章"*Compilers Using A Unirersal Intermediate Laugnage*"(后面称"*Goss*")中,该专利被授予*Goss*等人。

但现存的公用中间语言存在许多突出的问题,因此,在电信业务应用中的使用受到限制。*Tanenbaum*方法的一个主要问题是,当把接口输入码转换为中间语言时,将丢失原始输入信息。原始输入信息的丢失,是因为*Tanenbaum*使用了基本上是用于简单堆栈机的汇编语言。*Goss*专利主要是指导把增强型的或过程型格式的高级编程语言与不同数据处理器接口起来。*Goss*使用带有4个操作字段的"*quad*"作为中间语言的基本码结构。参见*Goss*的第6行第5栏到第38行第6栏。这种不灵活结构不能充分地表示特定的*SCE*输出,如非过程型编程码。另外,*Goss*的中间语言是按照高级编程语言码的自上而下分析而设计的。参见*Goss*的第34—36行的第5栏。因此,*Goss*方法不能很好地适用于产生代表典型*SCE*输出码的中间码。例如,产生于*SCE*输出码的*Goss*中间码不能完全地保存*SCE*输出码的全部信息,从而不能使有效中间码最优化。因此,*Goss*专利不能提出一个合适的中间语言结构用于接口,比如电信业务的*SCE*和*EE*。

从上面可知,需要一个有效且灵活的电信系统接口,它能使在不同*SCE*开发的业务应用在不同的*EE*中,而不需要*SCE*和*EE*之间

的分离的接口，并且没有现存中间语言技术中的问题。

本发明提供了在电信系统中接口不同SCE和EE的一种方法和装置。本发明的方法包括如下步骤：定义一组中间码操作，该中间码操作适于表示在选择的SCE中开发的电信业务。分析从一个选择的SCE中的输出码，用以形成一个具有多个节点的分析树，所述分析树表示在SCE中开发的一种电信业条。从该套中间码操作和分析树中产生表示分析树的节点的中间码，从而中间码就可基本上保存包含在SCE输出码内的所有信息；以及从中间码产生用于每个EE的目标码，从而在选择的SCE中开发的电信业务可提供给每个EE。

按照本发明的一个方面，提供了一个电信系统接口，它包括：要与多个业务EE接口的至少一个SCE；分析SCE的输出码，以形成分析树的分析程序，该分析树表示在SCE中开发的电信业务，产生表示分析树节点的中间码的中间码生成器，从而使SCE输出码中包含的基本上所有的信息保存在中间码中；以及目标码发生器，用于从中间码中生成用于每个EE的目标码，从而在SCE中开发的电信业务可以提供给每个EE。

按照本发明的另一方面，通过使用一行中间码—这里指字节组，中间码基本上保存了包含在SCE输出码的所有信息，用以表示SCE输出码分析树的每个节点。因此，本发明的电信系统接口包括一种中间语言，它能表示诸如SCE输出码分析树的语句和说明节

点。因此，为产生一个最优化中间码提供了足够的信息。

作为本发明的一个进一步特征，在特定的 *SCE* 上开发的电信业务可以使用在多种不同的 *EE* 中。*SCE* 和 *EE* 之间的深度耦合被消除，因此不再需要设计一个位于 *SCE* 和 *EE* 间的分离的接口。业务接口设计、操作和维持费只是 *SCE* 和 *EE* 数量的线性函数，而不是当前使用的交叉编译方法的二次方函数。

作为本发明的另一个特征，所提供的系统接口在 *SCE* 上没有操作的限制，并且不使用特有的格式，所以 *SCE* 和 *EE* 的选择可与卖方无关。该接口可允许卖方设计一个单一业务，它能在一组不同的 *EE* 上并行使用，并且允许卖方有效地维护和更新他们的业务。

作为本发明的另一个附加特征，提供一个接口可以在不同的 *EE* 上重新使用当前存在的业务，而不用重新设计或改变现存业务。可以设计中间码以适应多个当前使用的 *SCE* 输出语言。于是，业务库可被维护。因此，比如说当开发不同的 *EE* 时，不需重新设计业务。

作为本发明的进一步特征，提供了一个面向应用的程序语言（*AOPL*），它包括能很好地表示多个不同 *SCE* 输出码的中间码结构。依照本发明的中间码结构是基于一个灵活 *AOPL* 码行，这里指字节组，它有多个可变化的字段。比如说，该结构比上面所述的通常需要 4 个字段的 *G oss* 四元组结构更灵活。本发明的字节组结构特别适于表示 *SCE* 输出码的分析树，它经常用非过程型的程序语言。本发明因此比现存中间语言编译器提供了更多的优点，比如提高了

中间码的最优化以及在多种 *EE* 中更有效的生成目标码。

上述特征以及附加特征和本发明的优点，通过参照下面详细描述以及附图将变得更加明显。

图 1 是按照本发明电信系统的示例性框图。

图 2 是按照本发明电信系统接口的更详细框图。

图 3 是按照本发明的，用以说明在电信系统接口内一组示例性码处理步骤的框图。

图 4 是在 *SDL* 图形表示法中开发的部分示例性业务应用的流程图。

图 5 到图 8 说明用于示例性电子表决业务应用中的 *SDL* 图形表示法。按照本发明，它可以与多个不同 *EE* 接口。

本 发明提供了一种在电信系统中用于把不同的 *SCE* 和 *EE* 进行接口的方法和装置。尽管下面的描述说明了电信系统接口在特定的 *SCE* 和 *EE* 中的使用，但应明白本发明还可以应用到更多种其它电信系统中。而且，尽管这里使用多个简单的编程设计语言为例用以说明目的，但应明白本发明特别适用于电信系统应用。

图 1 给出了按照本发明的电信系统的方框图。该示例性系统包括一个具有多种不同的开发工具的业务生成环境(*SCE*)10。在 *SCE*10 中的示例性开发工具包括一个决策图编辑程序 12，一个扩展卡工具 14 和一个有限态自动机(*FSM*)编辑程序 16。决策图编辑程序和 *FSM* 编辑程序通常被认为是图形编辑程序。*SCE* 也可以包

括其它类型的开发工具，如计算机辅助系统工程(*CASE*)工具。通常，不同系统的卖方可以使用每一个开发工具12、14和16用以开发电信业务。使用不同的开发工具在*SCE*10中开发的电信业务在一组不同的执行环境(*EE*)20中被执行。在*EE*组20中示例性的*EE*包括一个业务交换处理器(*SSP*)22，一个业务控制处理器(*SCP*)24和一个信息外设(*IP*)26。通常，*SCP*24附有一个业务结点(*SN*)或与其组合一起来使用。*IP*26可以是比如一台计算机，一台传真机，一个视频终端或是一个文字到语音的转换机。通常*EE* 22，24和26被作为智能网络(*IN*)单元。关于*IN*的其它详细情况，可以在*AT&T Technical Journal*，*Vol.* 70，*Nos.* 3—4，*Summer*1991中，*M. Morgan*等人的"*Service Creation Technologies for the Inteligent Network*"文章和*AT&T Technical Journal*，*Vol.* 70，*No.* 5，*Summer*1991中，*G. Wyatt*等人的"*The Evolution of Global Intelligent Network Architecture*"文章中发现。上述两文章都在这里用作对比文献。

在现有的电信系统中，通常一个分离的接口用以把在*SCE*10中使用每个开发工具12，14和16开发的业务与每个*EE* 22，24和26相结合在一起。例如，决策图程序12可以设计成具有一个接口以允许该程序只用*SSP*22运行。在当前系统接口方法中，比如说，把一个电信系统的卖方的决策图程序12与另一个卖方设计的*SCP*24一起使用是不可能的。本发明部分通过使用一个基于面向应用的程

序语言(*AOPL*)的系统接口 30，从而避免了现有技术的这些问题。本发明的 *AOPL* 是以这样一种方式构成的，它能容易地把由使用不同的 *SCE* 开发工具产生的业务与多种 *EE* 综合在一起。这里所使用的术语"*AOPL*"是指以合适方式定义的公用中间代码的一种类型，用以表示 *SCE* 输出码分析树。

图 2 是本发明电信系统接口 30 的一个示例性实施例。接口 30 包括一个 *SCE* 侧 42 和一个 *EE* 侧 44，该接口用以把一个 *SCE* 与一个或多个 *EE* 互连。*SCE* 侧 42 可以表示，比如说图 1 所示的决策图程序 12 的输出，而 *EE* 侧 44 可以表示 *SSP*22 的输入。决策图程序 12 还可以被当作位于 *SCE* 侧 42 内的一个图形用户接口(*GUI*)。通过 *AOPL*46 定义的一个公用中间语言，接口 30 的 *SCE* 侧 42 和 *EE* 侧 44 被互连。通常，*SCE* 侧 42 可以根据在从一个 *SCE* 的线路 48 上接收的 *SCE* 输出码，从 *AOPL* 码的形式生成一个业务说明文件。从 *SCE* 输出码到 *AOPL* 码的转换是在分析程序 50 中完成的。其中分析程序 50 目前处在接口 30 的 *SCE* 侧 42 内。分析程序 50 把决策图程序 12 的高级 *SCE* 输出码转换为一个合适的 *AOPL* 码。然后，通过接口 30 把 *AOPL* 码传输到接口的 *EE* 侧 44。在 *EE* 侧 44，*AOPL* 码驱动码发生器 52，该码发生器把 *AOPL* 码转换成一个希望的目标码。然后，通过连线 54，把目标码提供给一个合适的 *EE*。

一般来说，可以使用不同的分析程序 50 把特定 *SCE* 的操作码转换为公用中间 *AOPL* 码。同样，可以使用不同码发生器 52 来接收

AOPL 码。并为每个 *EE* 产生一个合适的目标码。因此，接口 30 的 *SCE* 侧 42 和 *EE* 侧 44 可以 在 *SCE* 和 *EE* 内实现，而不是在，例如说一个独立的接口单元内。但是，现存的 *SCE* 和 *EE* 中，希望在一个独立的接口单元内包括分析和码生成功能。通过使用系统接口 30，在 *SCE*10 中用每个开发工具 12，14 和 16 形成的业务，可以与每个 *EE* 22，24 和 26 接口起来，如图 1 所示。此外，通过分析程序 50，*AOPL*46 和码发生器 52，可以把先前以高级语言 *SCE* 开发的现存业务转换为特定的 *EE* 的希望目标码。用这种方式，在不同 *SCE* 中产生的电信业务就可以在多种不同的 *EE* 中被再使用，而不需要重新设计 *SCE*。现存的电信业务，如那些已在特定的 *SCE* 中开发，用以在电话网络 *SSP* 和 *SCP* 中执行的业务，可使用一个合适的分析程序 50，通过简单地把特殊 *SCE* 的业务说明文件转换为 *AOPL* 码进行重新使用。图 2 所示的 *AOPL*46 将在下文中更详细地描述。

本发明的电信系统接口使用一个面向应用的中间语言—*AOPL*，其结构特别适合于某些电信应用。总得来说，*AOPL* 是有效描述 *SCE* 输出码分析树 的语言。在本领域中，分析编程码以产生分析树是已公知的。例如在 *Addison—Wesley*，*March*，1988 的第 40—42 页，*A. Aho* 等人的文章“*Compilers*，*Principles*，*Techniques*，*and Tools*”中所描述的。上述文章在这里作为对比文件。本发明的接口使用一个中间码结构，它表示给定的电信业务的 *SCE* 输出码分析树的节点。通过表示分析树节点，中间码基本上保存了在 *SCE* 输出码内

的所有信息。*SCE* 输出码中的术语"基本上所有"是指那些通常在输出码分析树中所发现的信息，比如象 *SCE* 输出码的结构，说明、语句和运行等信息。

对于一个给定的接口，可以选择多种不同的 *SCE*，然后限定一组适于代表在一个所选择的环境中开发的电信业务的中间码操作。大多数 *SCE* 可通过限定一组合适的中间码操作进行调节。下面将给出几个中间码操作的例子。

本发明的 *AOPL* 包括一个通用语法，结构和设计指南。尽管在给定应用中使用的 *AOPL* 的特定实施例通常将按照被接口的 *SCE* 和 *EE* 而变化，但语言的通用结构可以如下描述。一个示例性的 *AOPL* 文件通常包括许多不同的码行，在这里指字节组，它代表 *SCE* 输出码的分析树。每个 *AOPL* 字节组都被分号分开，并且包括许多由逗号分隔的不同字段。一个示例性的 *AOPL* 字节组包括三个字段，它们分别相应于字节组的标号、类型和操作。标号字段包括除逗号和分号以外的任何可印刷的字符序列，并且提供方便的方法用以标识特定的字节组。按照本发明的类型字段，既可以是 *P—node*，也可以是 *S—node*。该类型字段用以表明由特定的字节组表示的分析树节点的类型。*P—node* 类型字节组可以包括说明、表达式和语句，而 *S—node* 型字节组常包括较长的程序序列，比如数据结构。*AOPL* 字节组的操作字段包括一个操作符，它可以表明如在 *SCE* 输出节点分析树的一个节点的操作。操作字段还可以指示在那种操作中

使用的操作数。示例性的 *AOPL* 操作符例如可以包括 *LEAF*、*VAR-REF*、*CONSTREF* 和 *SEQUENCE*。

一个 *AOPL* 字节组还可以包括许多附加的字段。例如，字节组可以包括在操作字段内标识的每个操作符的附加字段。这些附加字段这里称为操作数字段。操作数字段可根据标号来标识，比如与 *SCE* 输出码分析树的其它节点所对应的不同字节组。操作数字段代表由操作符字段上规定的操作符运行的其它字节组。可在特定 *AOPL* 字节组上包括的其它附加字段是名称和数值属性字段。名称属性字段通常包括一个字符串，它对应于 *SCE* 输出码的一个记号。数值属性字段通常包括一个整数值，指示，比如在字节组中运行的数据的大小和地址。

本发明的 *AOPL* 识别三个不同种类的记号。这些记号是在 *SCE* 输出码中具有集合意义的字符序列。由 *AOPL* 识别的记号包括标识符、文字和分隔符。标识符可以是字母、数字、和/或特殊符号的任意序列。标识符的第一字符可以是除了数字以外的任何字符。在上述操作符字段内使用的操作符被留作 *AOPL* 关键字，因此 不能作为在 *SCE* 输出码中的标识符。文字可以是两种类型之一，即字符串(也被称作串文字) 或整(*integer*)文字。被本发明 *AOPL* 识别的第三类记号是分隔符。分隔符可以包括一些的字符，如“；”、“,”、“(”和“)”。*AOPL* 以 *SCE* 的高级 语言或输出码标识这些记号，并且产生代表记号的功能性互连的分析树。通过取出与任何特定记号相

关的最长字符串，以*SCE*输出码标识这些记号。*SCE*输出码的某些部分，称为“白色空间”，是不作为记号的。这些白色空间包括空格、制表、换行和换页。

在下面语言模块中总结了上面所描述的*AOPL*的语法。

```
    <AOPL-program>: <AOPL-node> | <AOPL-program>
<AOPL-node>
    <AOPL-node>: <node-head> <node-tail>;
    <node-head>: <label>, <node-type>, <opcode>,
<node-list>
    <label>: identifier
    <node-type>: P_node |
                 S_node
    <opcode>: <AOPL_opcode> (integer-literal)
    <AOPL_opcode>: keyword
    <node-list>: <label> |
                 <node-list>, <label>
    <node-tail>: empty |
                 <attribute-list>
    <attribute-list>: <list-entry> |
                 <attribute-list>, <list-entry>
    <list-entry>: <name-attribute> |
                  <value-attribute>
    <name-attribute>: string-literal
    <value-attribute>: integer-literal
```

在上述语言模块中，符号“<>”包围所示的每个示例性字节组的某些元素，并且符号“|”表示逻辑“或”操作符。

本发明的*AOPL*使用一个简单的*C*程序加以说明。但是，必须再一次强调，*AOPL*结构特别适合于电信系统应用，并且这里显示用以说明*AOPL*的示 例性*C*程序主要是为了说明目的。在这个*C*语言程序例子中，*SCE*10中使用的一个开发工具提供了如下格式输出码：

```
main()

int x;
int Y;
```

```
        y = 2;
        x = y*y + 2;
        printf("%d", x);
}
```

使用如上面描述的本发明的 *AOPL* 结构，上面所示的示例性 *C* 程序可以中间 *AOPL* 码的形式如下表示：

```
main, P_node, SEQUENCE(5), x_decl, y_decl, stm1, stm2,
stm3;
x_decl, P_node, INTDECL(1), x_var;
x_var, P_node, VARREF(2), x_leaf, x_decl;
x_leaf, P_node, LEAF(1), int, 'x';
y_decl, P_node, INTDECL(1), y_var;
y_var, P_node, VARREF(2), y_leaf, y_decl;
y_leaf, P_node, LEAF(1), int, 'y';
stm1, P_node, ASSIGNMENT(2), y_var, 2_const;
2_const, P_node, CONSTREF(1), 2_leaf;
2_leaf, P_node, LEAF(1), int, '2';
stm2, P_node, ASSIGNMENT(2), x_var, r_stm2;
r_stm2, P_node, PLUS(2), expr1, 2_const;
expr1, P_node, MULT(2), y_var, y_var;
stm3, P_node, FUNCTION(2), printf, printf_arg;
printf, P_node, LEAF(1), int, 'printf';
printf_arg, P_node, ARG_LIST(2), arg1, x_var;
arg1, P_node, CONSTREF(1), %f_leaf;
%f_leaf, P_node, LEAF(1), string, '%d';
```

在上面的 *AOPL* 码中，字节组“*main*”表示示例性 *C* 程序的分析树的根节点。*AOPL* 码还包括许多不同的操作符，以大写字母的方式显示。一个示例性的操作符是 *SEQUENCE* 操作符。*SEQUENCE* 操作符引导 *AOPL* 接口，为跟随其后的五个示例性操作

数产生顺序码。正象上面提到的，跟随一个 *AOPL* 操作符的操作数还与在 *SCE* 输出码分析树中的各种节点相对应。在给出的示例性程序中，顺序操作符的每个操作数或是 *C* 程序中的说明，或是语句。因此，这些操作数的每一个代表输出码分析树的一个节点。每个还与在中间 *AOPL* 码中的不同字节组或基本码行相对应。包括 *LEAF* 形式的操作符的字节组代表 *SCE* 输出码分析树的终端节点，在 *AOPL* 语言的特定实施例中预先定义了该终端节点的操作数。在给出的示例性 *AOPL* 码中，预先定义的操作数以斜体方式表示，并且代表象整数和字符串那样的原始数据类型。通常，操作符和预先定义的操作数将随着 *SCE* 的功能度和所使用的开发工具的不同而变化。

上例中仅使用了 ***P—node—type*** 字节组。下面给出了一个具有 *AOPL* 码表示的包括 ***S—node—type*** 字节组的第二示例性 *C* 程序。

```
struct date {
        int day;
        int month;
        int year;
} due_date;
```

下面给出了与第二 *C* 程序相应的示范性 *AOPL* 码。

```
due_decl, P_node, RECDECL(1), due_var;
due_var, P_node, VARREF(2), due_leaf, due_decl;
due_leaf, P_node, LEAF(1), date_record, 'due_date';
date_record, S_node, RECORD(4), field1, field2, field3,
```

```
date_leaf;
field1, P_node, RECFIELD(2), day_leaf, date_record;
day_leaf, P_node, LEAF(1), int, 'day', 4;
field2, P_node, RECFIELD(2), month_leaf, date_record;
month_leaf, P_node, LEAF(1), int, 'month', 4;
field3, P_node, RECFIELD(2), year_leaf, date_record;
year_leaf, P_node, LEAF(1), int, 'year', 4;
date_leaf, P_node, LEAF(1), date_record, 'date', 12;
```

在上述示例性 *AOPL* 码中，操作符 *RECORD* 规定记录本身的名称和大小，而操作符 *RECFIELD*(2) 表明在记录中的每个字段的名称和类型。与 *RECORD* 操作符相应的操作数确定了记录字段。与运算符 *LEAF* 相对应的每个终端节点包括一个数值属性字段，该字段用以标识终端节点的大小。

为了有效地构造如上所述的 *AOPL* 码，本发明可以使用图 3 所示的一个多级 结构。可以把系统接口 60 分成几个不同的码转换操作或传送。每次传送可以包括一个进行码转换的独特的操作设置。比如说，来自任意开发工具 12、14 和 16 的 *SCE* 输出码可被提供到线路 64。在第一传送 66，把 *SCE* 输出码转换为第一传送码，通过输出 68，把第一传送码提供给第二传送 70。使用第一传送 66，把来自特定 *SCE* 的 *SCE* 输出码转换为能很容易地被转换为公用中间语言的语言。因此，在第一传送 66 使用的操作设置可以被限定，以使得它贴近地反映产生 *SCE* 输出码的 *SCE* 的特征。这样，在产生公用中间 *AOPL* 码之前，很容易改变接口，使得它适应于 *SCE* 中的任何变化，并且能标识和报告 *SCE* 码级的错误。后面将给出用于电子表决业务应用的 *SDL* 表示的一个示例性第一传送操作设置。在第一传送

66 使用的操作设置可以存放在业务生成环境其本身内。

第二传送 70 接收第一传送 66 产生的第一传送码，并且将用在所有的 *SCE* 中使用的公用中间语言产生第二传送码。第一传送 70 通常包括比在第一传送 66 中所用的操作设置更宽更一般的操作设置。因此可以在第一传送 66 引入新的操作而不影响第二传送 70。第二传送数据库 72 可以包括，比如用以扩展第一传送码中的一些节点的第二传送宏指令。第二传送 70 在输出 74 上提供将在第三传送中被处理的公用中间 *AOPL* 码。第三传送 76 设计成简化每个 *EE* 的目标码生成。通常定义 第三传送码的操作设置更接近于反映特定的 *EE* 属性。在第三传送中，存储在第三传送数据库 78 的一组第三传送宏指令　可用于扩展第二传送码的节点。通过连线 80，可以把所得到的目标码提供给特定的 *EE*，如在电话交换网络中的 *SCP* 或 *SSP*。尽管图 3 给出了一个示例性的 3 级结构用以实现接口 60，但应明白在一个给出的应用中，接口可以包括更多的级用以把 *SCE* 输出码转换为 *EE* 目标码。

在图 3 中所示的三次传送可以分布在 *SCE* 和 *EE* 上，其中第一传送码发生器在 *SCE* 中而第一传送码发生器在 *EE* 内，所以 *SCE* 产生来源于第二传送 70 的公用中间码，并且 *EE* 接收该公用中间码。这样第一传送 66 可在每个 *SCE* 内进行，而第三传送 76 可在每个 *EE* 中进行。为了产生公用中间码，*SCE* 的输出部分使用第一传送程序和预先定义的操作设置。在第二传送中，中间码发生器用于产

生并且优化公用中间码。最后，*EE* 接收该中间码并产生合适的目标码。通常，每次传送都涉及把先前传送的码转换为一个更合适的码。因此，图 3 中的 3 级结构的第一、第二和第三传送与图 2 的分析程序 50、*AOPL* 码发生器 46 和目标码发生器 52 各自所执行的操作相对应。在有些情况，没必要在每个传送都执行转换过程。作为替代，可以使用表示象“*Case*”和“*if*”的典型程序结构的模块作为直接置换。于是在数据库 72 和 78 中存储的操作设置可以包括简单的宏指令，用以把 *SCE* 输出码分析树的节点从高级 *SCE* 输出码转换为低级 *EE* 目标码。

本发明的电信系统接口已应用于多种不同的 *SCE* 和 *EE*。图 4 表示使用 *CCITT* 业务规范描述语言(*SDL*)的示例性业务状态机的流程图。当用于图形表示形式时，*SDL* 语言指 *SDL/GR*，当用在文字、正文表示形式时，*SDL* 语言指 *SDL/PR*。使用常规文字编辑器可以产生 *SDL/PR* 格式的程序。但是，*SDL/GR* 程序通常用，例如专为操作 *SDL/GR* 符号而设计的图形编辑器来开发。图 4 到图 8，使用了 *SDL/GR* 图形符号。*SDL/GR* 图形符号通常提供以图形符号格式表示电信业务功能的手段。例如，在 *CCITT* 蓝皮书 *Vol. X*, *Fascicle X. 1*, “*Foctional Specification and Description Language*,” *Recommendation Z. 100* 和 *Annexes A. B. C. and E*, 和 *Recommendation Z. 110*, 1988 中，将有更详细的描述，在这里，它们构成对比文件。*SDL/GR* 图形表示法通常类似于分析树，因此

它可以用字节组描述。用“起始”标注的标号100与用第一字节组表示的*SDL STATE*符号相对应。标号102、104是*SDL/GR*图形表示法中的*INPUT*符号，并且可用不同的字节组表示它们。标号106、108也表示*SDL/GR*图形表示法的*STATE*符号。每个*STATE*符号100、106、108的内容标识一个状态，个而每个*INPUT*符号102、104的内容标识一个事件。可以用字节组的形式表示*SDL/GR*图形表示法，从而这种表示法内的每个标号都可与一个唯一*AOPL*操作相对应，其中*AOPL*操作的操作数反映了与标号的连接。*STATE*和*EVENT*操作分别表示*STATE*和*INPUT*标号内执行的操作。每个操作符包括许多与每个标号输出相对应的操作数。图4所示的示例性的*SDL/GR*符号，第一传送码发生器内被转换为下面的第一传送码。

```
state, P_node, B_STATE(2), event1, event2, 'Initial';
event1, P_node, B_INPUT(1), state1,
'bri!terminating_call';
event2, P_node, B_INPUT(1), state2,
'bri!send_answer_completed';
state1, P_node, B_STATE(0), 'Send_Answer';
state2, P node, B STATE(0), 'Wait_For_Vote';
```

象上面所述，第一传送码可通过分析*SCE*输出的高级码产生。此外，在系统接口单元内使用分析程序也可以产生第一传送码。使用第二传送把上面的第一传送码转换为下面能够提供给任何不同的*EE*的公用中间*AOPL*码。

```
state1, P_node, STATE(2), state1_leaf, event_list1;
state1_leaf, P_node, LEAF(1), state, 'Initial';
event_list1, P_node, SEQUENCE(2), event1, event2;
event1, P_node, EVENT(2), event1_var, handler1;
event1_var, P_node, VARREF(2), event1_leaf, event1_decl;
event1_leaf, P_node, LEAF(1), event1_record,
'bri!terminating_call';
handler1, P_node, NEXTSTATE(1), state2;
state2, P_node, STATE(2), state2_leaf, event_list2;
state2_leaf, P_node, LEAF(1), state, 'Send_Answer';
event2, P_node, EVENT(2), event2_var, handler2;
event2_var, P_node, VARREF(2), event2_leaf, event2_decl;
event2_leaf, P_node, LEAF(1), event2_record,
'bri!send_answer_completed';
handler2, P_node, NEXTSTATE(1), state3;
state3, P_node, STATE(2), state3_leaf, event_list3;
state3_leaf, P_node, LEAF(1), state, 'Wait_For_Vote';
```

上面的第二传送码，或中间 *AOPL* 码，特别适于表示用业务逻辑语言（*SLL*）开发的业务。*SLL* 是开发用以按照互相关的 *FSM* 描述电信业务应用的面向应用的语言。图 4 所示的 *SDL* 符号的不同状态是用操作符 *STATE* 表示的。与标号 102、104 相应的事件是用操作 *EVENT* 表示的。在 *STATE* 操作符中，第一操作数标识用以确定状态名称的 *LEAF* 节点，第二操作数标识由 *SEQUENCE* 节点表示的事件处理程序清单。*EVENT* 操作符有用以标识 *VARREF* 节点的第一操作数和在一个事件处理程序内标识一个动作特殊子树的根的第二操作数。*VARREF* 操作符有用以标识规定事件名称的

LEAF 节点的第一操作数和标识用于说明特殊事件的 *DEFEVENT* 节点的第二操作数。应注意在上述 *AOPL* 码中没有指定标号 *event—list2* 和 *event—list3*，这是由于与它们相产的状态只在前面提到，并没有在本例的 *SCE* 输出码示例性段中定义。图 4 中所示的标号的内容是用于说明的目的，并不与一个特定的程度语言相对应。总的来说，图 4 中示例性图形符号内的文字可以使用任何程序语言的语法。

在 上面示例性的中间码中，一些说明和记录节点，如在 *event1—decl*、*event1—record*、、*event2—decl* 和 *eveat2—record* 中的那些假定被预先定义。从上例中可以看出，附加码操作出现在第二传送码上。第二传送码更加通用，它可以描述，例如多种不同的 *FSM*。相反，第一传送产生的码与 *SCE* 输出码更加相关，它可以不适于表示其它 *SCE* 的码。上面给出的中间 *AOPL* 码可以转换为一个希望的 *EE* 的特定的目标码。另外，比如说，如果目标码与 *SLL* 非常相似，就不需要转换目标码。

另外，一个用在很多 *SCE* 中的示例性开发工具是决策图编辑程序。决策图编辑程序如 *FSM* 编辑程序，是在业务设计和开发过程中作为辅助用以描述各种电信业务流程的一种图形编辑程序。通常，决策图的结构类似于分析树，因此，可以用一个 *AOPL* 字节组有效地表示每个决策图。决策图可包括各种不同的决策图节点，它们对应于业务操作并且由分支互连。每个操作可以包括多个子操作，如

示例性操作的时间、日期和百分比。示例性操作这里称为 *DECISION*。在决策图中用于百分比(*percent*)子操作符的第一传送码可以如下给出:*Percent*—1, *P*—*node*, *DECISION*(2*n*) ,Val_1,br_1,… Val_n,br_n,'*Percent*', 其中 *n* 表示来自百分比节点的分支输出的数目。该码行还包括对于每个分支的一对操作数,它们用以确定从分比决策图节点到一个特定分支的分支相似性。在决策图中,时间子操作符节点的第一传送码可以如下给出:*time*—1,*P*—*node*,*DECISION*(2*n*+1),Val_1,br_1,…Cal_n,br_n,*value*, '*time*'。对于时间节点,*DECISION* 操作符有一个附加操作数"*value*",它用以确定一个时间。在这个时间后,特定决策图节点将被连结到一个特定的分支。在第二传送后,第一传送码的上述两个示例性行被扩展成下面所示的第二传送码。对于百分比子操作符,第二传送可以产生下面的码:

```
percent-2, P_code, DGNODE(2), percent_call, br_list;
percent_call, P_code, FUNCTION(2), percent_leaf, arg_list;

percent_leaf, P_code, LEAF(1), int, 'percent';
arg_list, P_code, ARG_LIST(n), const1, ..., constn;
br_list, P_node, LIST(n), br1, ..., brn;
```

对于时间子操作符,第二传送可以产生如下的码:

```
time-2, P_code, DGNODE(2), time_call, br_list;
time_call, P_code, FUNCTION(2), time_leaf, arg_list;
time_leaf, P_code, LEAF(1), int, 'time';
arg_list, P_code, ARG_LIST(n+1), const1, ..., constn+1;
br_list, P_node, LIST(n), br1, ..., brn;
```

在本发明的另一个实施例中，系统接口可以产生由用于特定的*EE*的汇编语言组成的目标代码。为了简化汇编码的产生，可以选择相应的第三传送操作，使它包括一个给定分析树的被预先定义终端的操作数。作为一个例子，特定的*EE*可以包括双地址格式的指令：

操作符　　源　　目的地

这种格式的指令中，“操作符”是指汇编语言的操作符，“源”和“目的地”表示操作数字段。示例性的汇编语言操作符包括*MOVE*，*MULT*和*ADD*。因此，对于上面给出的第一示例性*C*程序，可以产生目标码。所得的第三传送码，或目标码，可以如下面所示：

```
label1, P_node, move(2), y_leaf, register_leaf;
label2, P_node, mult(2), y_leaf, register_leaf;
label3, P_node, add(2); 2_leaf, register_leaf;
label4, P_node, move(2), register_leaf, x_leaf;
x_leaf, P_node, leaf(1), memory, 'x';
y_leaf, P_node, leaf(1), memory, 'y';
2_leaf, P_node, leaf(1), int, '2';
register_leaf, P_node, leaf(1), register, 'r6';
```

在这个示例性的第三传送码中，操作数“*memory*”、“*register*”和“*INT*”假设被预先定义，并且它们分别表示储器位置、寄存器和一个常量。

本发明便于实现在很多不同*EE*中的电信业务，而不需要为每个*EE*设计一个分离的接口。图5到图8给出了用于示例的“电子表

决”业务的完整的*SDL/GR*表示法。其中电子表决业务能使选民在电信系统上进行表决。例如，电子表决业务可以在电话交换系统上完成。但是，业务本身通常是以图形表示法(如*SDL/GR*)被最先开发。通常，*SL*表示法包括许多不同的内部相关符号，按照逻辑流程图描述该业务。这些符号依一定顺序或优选级来安排，该顺序确定了符号的范围。在图形编辑程序中，位于接近编辑屏幕的左上角的给定符号要比在这个符号下面或右面的其它符号具有更高的优先级。例如，图5中的*DECLARATION*符号500，在该例中表示全局说明，这是因为该符号位于其它象由*START*符号508定义的*FSM*的符号之上。符号508比符号506有更高的优先级，因此，在标号506中的说明的范围是与*START*符号508相应的*FSM*。

在*SDL/GL*表示法中的每个标号通常包括有文字。一个标号中的文字可能对应于，例如一个状态，事件或*FSM*的标识符，一个数据说明，一个在*SLL*语法中的语句或一个注释。为了适当在一个符号内可能容不下的长行文字，使用字符“\”指示一行的连续。比如说，当把*SDL/GR*转换为*SLL*用以在特定的*EE*内执行时，可以删去行连续字符“\”。象图4所地的这个例子中，这些符号包含了与一个*SLL*语法相应的文字。当然，所使用的该特定语言可随着应用的不同而变化。

图5包括在描述电子表决业务的*SDL/GR*中的几个*DECLARATION*标号500到506。在*SDL/GR*中的*DECLARATION*符号

相当于标号 500 和 506，并且被用以在 *SDL/GR* 程序内说明和初始化数据。比如，标号 500 说明一组动态变量，该动态变量对应于该电子表决例中五个不同候选者的每一个的选择计数器，而标号 504 表明每个字符串长度为 80 个字符的以字符串的形式表示的五个候选者的名称。

图 5 中标号 508 在 *SDL/GR* 表示法中称作一个 *START* 符号，而标号 510 是一个 *STATE* 符号。*START* 符号通常用以定义一个 *FSM*，并且连接到一个明确规定 *FSM* 的初始状态的 *STATE* 符号上。因此，*START* 和 *STATE* 符号内的文字分别规定了 *FSM* 的名称和它的初始状态。因为 *FSM* 的范围是其定义的整个 *SDL/GR* 程序，所以在 *SDL/GR* 程序内，*FSM* 必须是唯一的。但是，由于 *STATE* 的范围是与它相联的 *FSM*，所以可以在 *SDL/GR* 程序内复制 *STATE* 名称。标号 508 和 510 定义了名称为“选举”的 *FSM*，其有一个“空闲”初始状态。图 5 也示出了一个缺省的 *STATE* 符号 550，它定义对应于缺省事件处理程序的一组 *INPUT* 符号 552、554 和 556。缺省事件处理程序接收各种系统输入如一个未连接指示符，并终止 *FSM* 操作如 *STOP* 符号 558 所示。例如，当用户终止一个呼叫时，缺省事个处理程序提供一种用于终止 *FSM* 处理的方法。

图 6(*a*)表示“空闲”状态后的处理步骤。

从 *STATE* 符号 602，*FSM* 可以接收两个不同输入之一。图 6(*a*)的标号 604 是 *SDL/GR* 表示法的一个 *INNPUT* 符号。*INPUT*

符号内的文字规定了对应于事件处理程序的名称。由于事件处理程序总是与一个状态有关，因此象标号 604 的一个 *INPUT* 符号前必须有 *STATE* 符号。如果前面的状态是缺省状态，即由一个包含星号的 *STATE* 符号来指示的，则事件处理程序就是指缺省事 件处理程序。*INPUT* 符号 604 指示出一个事件处理程序“ *bri ! send—answer—completed*”接收到一个用以开始电子表决业务的一个可能输入。

图 6(*a*)中的标号 606 是 *SDL/GR* 表示法的一个 *DECSION* 符号，可以表示“ *if*”或“*test*”复合语句。在“*if*”复合语句情况下，标号内的文字包括关键字“*if*”和布尔(*boolean*)逻辑表达式，并且至多有两个分支从 *DECISON* 符号分出。该标号 606 是“*if*”型的。另外，在“ *test* 复合语句的情况下，标号内的文字包括关键字“*test*””和表达式，并且可从这个符号分出许多分支。图 7(*c*)的 *DECISION* 符号是“ *test*”型的。图 6(*a*)的 *DECISION* 符号指示出当接收到在 *INPUT* 符号 604 中规定类型的输出时，根据变量“*Called—dn*”作出判断。其中“ *Called—dn*”指用户输入的电话号码。如果“*Calledd—dn*”不等于用户所拔的用以接收总表决计数信息的数字“ *Tally—No*”，则进入 *PROCEDURE CALL* 符号 608，并且调用过程“*not—tally*”。发调用的过程执行后，*FSM* 就进入 *STATE* 符号 610 所示的状态“*Wait—For—Vote*”。另外，如果该计数数字已被用户调用，则调用 *PROCEDURE CALL* 符号 612 所指示的过程“ *is—tally*”，然后 *FSM* 进

入 *STATE* 符号 614 所示的状态“ *Wait—for—PIN*”。过程“*not—tally*”和“*is—tally*”分别示于图 7(*a*)和 7(*b*)。从起始的“*Idle*”状态，该“选举”*FSM* 也可接收 如 *INPUT* 符号 616 所示的“ *bir ! terminating— call*” 输入。如果接收到这个输入，*PROCEDURE CALL* 符号 618 就将 *FSM* 引入过程“ *bt—call*” 。执行完该过程后，*FSM* 返回 *STATE* 符号 620 中用“—”所指示的当前状态。在 *SDL/GR* 表示法使用的短线是表示 *FSM* 当前状态的简写。过程“*bt—call*”在图 7(*e*)中给出。

图 6(*b*)给出了 *STATE* 符号 614 所示的’*Wait—For—PIN* 状态后的处理步骤。*INPUT* 符号 616 接收由业务用户输入的数字。这此数字，例如可通过使用按键电话的双音多频(*DTMF*)命令输入。*DECISON* 符号 618 指示所接收到的数字是否与，比如说，被授权接收关于决表决计数信息的个人识别码(*PIN*)表一致。如果是，*PROCEDURE CALL* 符号 620 指示执行“*is—col—pin*”过程，然后 *FSM* 返回到 *STATE* 符号 622 所指示的当前状态。否则，如果接收到的数字不符合一个授权的 *PIN*，用户则接收一个如 *TASK* 符号 624 所示的消息，拒绝其接收总表决计数信息的权利。*SDL/GR* 表示法的 *TASK* 符号用以指示比如以 *SLL* 语句格式的事件处 理程序的操作。然后 *FSM* 如 *STATE* 符号 628 所示返回当前状态。

图 6(*c*) 表示通过与图 6(*a*)的 *STATE* 符号 610 相应的“*Wait—For—Vote*”状态所访问的符号。*INPUT* 符号 642 指示出从

STATE 符号 640 所示的状 态“ *Wait—For—Vote*” 接收到一个输入“ *Collected—digits*”。该输入可对应于由可从电话进入的系统用户对特定候选人的选择。比如，以 *DTMF* 命令格式的几个数字。如果接收到该输入，*DECISION* 标号 644 指出对照变量“ *Max—Cand*” 来检验数字计数器，其中“*Max—Cand*”对应于大一个给定的选举中最大可能的候选人选择号。如果数字计数器大于“*Max—Cand*”，则输入对应一个有效的候选人选择，而 *DECISION* 符号 646 表示，要进行另一个决策看看是否输入数字“ 9”。在这个例子中，数字“9”表示用户想要达到的总表决计数信息。

如果输入数字“9”，则根据所输入的数字，*TASK* 符号 648 向用户提供一个通知，即做出了一个不正确的选择。然后 *FSM* 返回由 *DECISION* 符号 630 所表示的当前状态。另一方面，如果输入了数字“9”，*TNSK* 符号 654 指示出要提示用户输入一个 *PIN* 用以验证，比如说，用户被授权接收总的表决计数。在提示用户输入 *PIN* 后，*FSM* 进入 *STATE* 符号 656 所示的状态“ *Wait—For—PIN*”。如果所收集的数字计数器指示用户已输入一个有效的候选人选择数字，那么 *PROCEDURE CALL* 符号 658 就指示应执行图 7(c)所示的“*it—max—cand*”过程。执行完该过程后，*FSM* 返回 *STATE* 符号 660 所示的当前状态。

图 7 示出了更详细的在示例性电子表决的 *SDL/GR* 表示法中所调用的过程。标号 700、716、724、740 和 756 对应于 *PROCEDURE*

START 符号，而标号 715、723、739、755 和 759 对应于 *PROCEDURE STOP* 符号。图 7(*a*)示出了从图 6(*a*)的 *PROCEDUCE CALL* 符号 608 处调用的过程“*not—tally*”的处理步骤。比如，*TASK* 符号 702 可以引导文字—语音转换器，把消息提供给用户。该消息用对应于 *SDL/GR* 表示法中一个 *COMMENT* 符号的括号示于 *TASK* 符号的右边。示例性的消息是向用户指出他们正表决的选举的名称。为了标识选举中的 5 个候选人，几个不同 *MACRO CALL* 符号 704 到 712 是顺序接入的。与符号 704 到 712 相应的宏功能在图 8(*a*)中给出，并且引导用户输入一个合适的数字以对一个特定的候选人进行表决，然后 *TASK* 符号 714 可以提示用户输入一个特定的数字，此例中为“9”，以进入每个候选人的当前的表决计数。

图 7(*b*)示出从图 6(*a*)的 *PRROCEDURE CALL* 符号 612 调用的过程“*is—tally*” 的步骤。*TASK* 符号 720 提示用户输入一个 *PIN*，然后 *MACRO CALL* 符号 722 调用图 8(*b*)所示的一个宏指令“*tts—digits*”。图 7(*c*) 示出从图 6(*c*)的 *PROCEDURE CALL* 符号 658 调用的过程“*lt—max—cand*”的步骤。。*DECISION* 符号 726 指示出，对收集的数字进行测试以确定用户选择了哪一个有效的候选人。例如，通过输入“1”，用户可以选举第一候选人。于是，*MACRD CALL* 符号 728 到 736 指示出调用图 8(*c*)所示的一个宏指令“*send*(n, s)”。图 7(*d*)示出了从图 6(*b*) 的 *PROCEDURE CALL* 符号 620 调用的过程“*is—col—pin*”的步骤，表明用户被授权

接收总的表决计数。*TASK* 符号 702 提供一个消息给用户，即将给出每个候选人当前的总数。大 *MACRO CALL* 符号 744 到 754，调用一个宏指令“*cand—text*”，以向用户提供每个候选人的正确的总数。图 8(*d*)中更详细地给出了宏指令”*cand—text*1”。图 7(*e*)表示从图 6(*a*)的 *PROCEDURE CALL* 符号 618 调用的过程“*bt—call*” 618 的步骤。

图 8 更详细地给出了在电子表决 *SDL/GR* 程序中调用的每个宏指令的处理步骤。图 8(*a*) 到图 8(*d*) 分别给出了宏指令“*cand—text*(*s*,*r*,*n*)”，“*tts—digits*”，“*send*(*n*,*s*)”和“*cand—text*1(*s*,*r*)”其它的详细情况。标号 800、810、820 和 830 对应于 *MACRO START* 符号。而标号 809、813、823 和 837 对应于 *MACRO STOP* 符号。标号 808 和 836 相当于出连接符，它们分别指示对应于 *DECISION* 符号 802 和 832 的决策操作的结束。

应注意到上述示例性电子表决的 *SDL/GR* 表示法和用于符号文字的语法可以随着应用的不同而改变。但是，上述电子表决示例用以说明本发明接口所提供的优越性。象在 *SDL/GR* 图形编辑程序中开发的诸如电子表决的业务应用，作为这里所提供的面向应用的接口结果，现在可以应用在任何一组不同的 *EE* 中。

按照本发明的图 5 至图 8 的电子表决业务所产生的一组第一传送码如下面所示：

```
P1s1,P_node,B_TEXT_G(1),P1s5,dynamic';
P1s5,P_node,B_TEXT_G(1),NULL,rc';
P1s2,P_node,B_START(2),P1s4,P1s3,'election';
P1s4,P_node,B_TEXT_F(1),P1s12,access';
P1s12,P_node,B_TEXT_F(1),NULL,dynamic';
P1s3.P node,B STATE(0),'Idle';
P1s6,P_node,B_STATE(3),P1s9,P1s11,P1s7,'*';
P1s9,P_node,B_INPUT(1),P1s17,completed';
P1s11,P_node,B_INPUT(1),P1s17,'bri!disconnected';
P1s7,P_node,B_INPUT(1),P1s17,'collect_failed';
P1s17,P_node,B_STOP(0),'';
P2s1,P_node,B_STATE(2),P2s6,P2s2,'Idle';
P2s6,P_node,B_INPUT(1),P2s13,answer_completed';
P2s2,P_node,B_INPUT(1),P2s4,terminating_call';
P2s13,P_node,B_DECISION(4),P2s14,P2s13T1,P2s16,P2s13T2,Tal
ly_No';
P2s13T2,P_node,B_TEXT(0),'false';
P2s13T1,P_node,B_TEXT(0),'true';
P2s4,P_node,B_P_CALL(1),P2s5,'bt_call';
P2s5,P_node,B_STATE(0),'-';
P2s16,P_node,B_P_CALL(1),P2s17,'not_tally';
P2s14,P_node,B_P_CALL(1),P2s15,'is_tally';
P2s17,P_node,B_STATE(0),'Wait_For_Vote';
P2s15,P_node,B_STATE(0),'Wait_For_PIN';
P3s1,P_node,B_STATE(1),P3s2,'Wait_For_PIN';
P3s2,P_node,B_INPUT(1),P3s3,'collected_digits';
P3s3,P_node,B_DECISION(4),P3s4,P3s3T3,P3s7,P3s3T4,COL_PIN'
;
P3s3T3,P_node,B_TEXT(0),'true';
P3s3T4,P_node,B_TEXT(0),'false';
P3s7,P_node,B_TASK(1),P3s6,'tts!send(port=Port_num,text="S
orry, no authorization.")';
P3s4,P_node,B_P_CALL(1),P3s5,'is_col_pin';
P3s5,P_node,B_STATE(0),'-'
P3s6,P_node,B_STATE(0),'-'
P4s1,P_node,B_STATE(1),P4s2,'Wait_for_Vote';
P4s2,P_node,B_INPUT(1),P4s3,'collected_digits';
P4s3,P_node,B_DECISION(4),P4s6,P4s3T7,P4s9,P4s3T8,Max_Cand
';
P4s3T8,P_node,B_TEXT(0),'true';
P4s3T7,P_node,B_TEXT(0),'false';
P4s6,P_node,B_DECISION(4),P4s4,P4s6T5,P4s7,P4s6T6,"9"';
P4s6T6,P_node,B_TEXT(0),'false';
P4s6T5,P_node,B_TEXT(0),'true';
P4s9,P_node,B_P_CALL(1),P4s10,'lt_max_cand';
```

```
P4s10,P_node,B_STATE(0),'-';
P4s7,P_node,B_TASK(1),P4s8,'tts!send(port=Port_num,text="S
orry, incorrect choice.")';
P4s4,P_node,B_TASK(1),P4s5,port=Port_num,prompt=speech_tex
t,
max_digits=4)';
P4s5,P_node,B_STATE(0),'Wait_For_Pin'
P4s8,P_node,B_STATE(0),'-';
P5s14,P_node,B_M_START(1),P5s15,'cand_text(s,r,n)';
P5s8,P_node,B_M_START(1),P5s9,'tts_digits(a,b,c,d)';
P5s25,P_node,B_M_START(1),P5s26,'cand_text1(s,r)';
P5s15,P_node,B_DECISION(2),P5s16,P5s15T9,"-"';
P5s9,P_node,B_TASK(1),P5s13,'tts!collect_digits(port=a,dia
ling_ terminator=b,prompt=c,max_digits=4);
P5s15T9,P_node,B_TEXT(0),'true';
P5s16,P_node,B_TASK(1),P5s20,",","."';
P5s26,P_node,B_DECISION(2),P5s27,P5s26T10,"-"';
P5s26T10,P_node,B_TEXT(0),'true';
P5s13,P_node,B_M_STOP(0),'';
P5s27,P_node,B_TASK(1),P5s29,string(r)';
P5s20,P_node,B_TASK(1),P5s23,n';
P5s1,P_node,B_P_START(1),P5s3,'is_tally';
P5s29,P_node,B_LABEL(1),P5s30,'*';
P5s23,P_node,B_LABEL(1),P5s24,'*';
P5s30,P_node,B_M_STOP(0),'';
P5s3,P_node,B_TASK(1),P5s5,PIN"';
P5s24,P_node,B_M_STOP(0),'';
P5s49,P_node,B_P_START(1),P5s50,'bt_call';
P5s5,P_node,B_M_CALL(1),P5s7,(Port_num,"#",speech_text,4)'
;
P5s50,P_node,B_TASK(1),P5s51,bri!send_answer(port=Port_num
)'; P5s7,P_node,B_P_STOP(0),'';
P5s51,P_node,B_P_STOP(0),;
P6s1,P_node,B_P_START(1), r ' ' G '1y ;
P6s3,P_node,B_TEXT_EXP(0),":';
P6s2,P_node,B_TASK(3),P6s5,P6s3,P6s4,';
P6s4,P_node,B_TEXT_EXP(0),election."';
P6s5,P_node,B_M_CALL(1),P6s6, (E_Cand1,"1",1)'; P6s6,
P_node,B_M_CALL(1),P6s8, (E_Cand2,"2",2)';
P6s8,P_node,B_M_CALL(1),P6s9, (E_Cand3,"3",3)';
P6s9,P_node,B_M_CALL(1),P6s10, (E_Cand4,"4",4)';
P6s11,P_node,B_TASK(1),P6s12,tts!collect_digits\'J'J1'_=r'
J1'_11U1')1')1=',111C1X._U1v11'
P6s10,P_node,B_M_CALL(1),P6s11, (E_Cand5,"5",5);
P6s12,P_node,B_P_STOP(0),'';
P7s12,P_node,B_P_START(1),P7s13,'is_co1_pin';
P7s13,P_node,B_TASK(1),P7s16,are,"';
P7s16,P_node,B_M_CALL(1),P7s23,
(E_Cand1,Election_Counter1);
```

```
P7s23,P_node,B_M_CALL(1),P7s24,
(E_Cand2,Election_Counter2);
P7s24,P_node,B_M_CALL(1),P7s25,
(E_Cand3,Election_Counter3);
P7s25,P_node,B_M_CALL(1),P7s26,
(E_Cand4,Election_Counter4);
P7s26,P_node,B_M_CALL(1),P7s21,
(E_Cand5,Election_Counter5);
P7s21,P_node,B_TASK(1);
P7s22,'tts!send(port=Port_num,text=speech_text)';
P7s22,P_node,B_P_STOP(0),'';
P8s6,P_node,B_P_START(1),P8s2,'1t_max_cand';
P8s2,P_node,B_DECISION(10),P8s3,P8s2T11,P8s4,P8s2T12,P8s5,
P8s2T13,P8s7, P8s2T14,P8s8,P8s2T15;
P8s2T15,P_node,B_TEXT(0),'5';
P8s2T14,P_node,B_TEXT(0),'4';
P8s2T13, P_node,B_TEXT(0),'3';
P8s2T12,P_node,B_TEXT(0),'2';
P8s2T11,P_node,B_TEXT(0),'1';
P8s7,P_node,B_M_CALL(1),P8s9,'send(E_Cand4,Election_Counte
r4)';
P8s5,P_node,B_M_CALL(1),P8s9,'send(E_Cand3,Election_Counte
r3)';
P8s3,P_node,B_M_CALL(1),P8s9,'send(E_Cand1,Election_Counte
r1)';
P8s4,P_node,B_M_CALL(1),P8s9,'send(E_Cand2,Election_Counte
r2)';
P8s8,P_node,B_M_CALL(1),P8s9,'send(E_Cand5,Election_Counte
r5)';
P8s9,P_node,B_LABEL(1),P8s10,'*';
P8s10,P_node,B_P_STOP(0);
P8s11,P_node,B_M_START(1), 'S1G, senc1'n,s);
P8s12,P_node,B_TASK(1),P8s13,tts!send(port=Port_num,text=s
peech_ text)';
P8s13,P_node,B_M_STOP(0),'';
```

用于示例性的电子表决业务应用的一组示例性公用中间 *AOPL* 码如下面所示：

```
@94840,P_node,S11PROG(2),@109072,@109296;t''7
@109072,P_node,SEQUENCE(2),@109104,@109240;
@109104,P_node,SIGNATURE(0),dynamic';
@109240,P_node,SIGNATURE(0),rc';
@109296,P_node,SEQUENCE(1),@109328;
@109328,P_node,S11FSM(5),@109360,@109488,@109552,@109728,@
110320;
@109360,P_node,LEAF(1),$fsm,'election';
@109488,P_node,STATE(2),@109520,@110352;
```

```
@109520,P_node,LEAF(1),$state,'Idle';
@110352,P_node,SEQUENCE(2),@110384,@79976;
@110384,P_node,EVENT(2),@110416,@110472;
@110416,P_node,LEAF(1),$event,'bri!send_answer_completed';
@110472,P_node,SEQUENCE(1),@110504;
@110504,P_node,IF(4),@110536,@110592,NULL,@112312;
@110536,P_node,SIGNATURE(0),Tally_No';
@110592,P_node,SEQUENCE(2),@110696,@112280;
@110696,P_node,SEQUENCE(2),@110728,@111944;
@110728;P_node,SIGNATURE(0),PIN"';
@111944,P_node,SEQUENCE(1),@111976;
@111976,P_node,SIGNATURE(0),'tts!collect_digits(port=Port_
num, dialing_terminator="#",prompt=speech_text);
@112280,P_node,NEXTSTATE(1),@112224;
@112224,P_node,STATE(2),@112136,NULL;
@112136,P_node,LEAF(1),$state,'Wait_For_PIN';
@112312,P_node,SEQUENCE(2),@112416,@79912;
@112416,P_node,SEQUENCE(7),@112448,@76688,@77304,@77968,@7
8600,
@79232,@79680;
@112448,P_node,SIGNATURE(0),election."';
@76688,P_node,SEQUENCE(1),@76720;
@76720,P_node,IF(4),@76752,@76832,NULL,NULL;
@76752,P_node,SIGNATURE(0),"-"';
@76832,P_node,SEQUENCE(2),@76864,@77016;
@76864,P_node,SIGNATURE(0),",."';
@77016,P_node,SIGNATURE(0),1';
@77304,P_node,SEQUENCE(1),@77336;
@77336,P_node,IF(4),@77368,@77448,NULL,NULL;
@77368,P_node,SIGNATURE(0),"-"';
@77448,P_node,SEQUENCE(2),@77480,@77632;
@77480,P_node,SIGNATURE(0),",."';
@77632,P_node,SIGNATURE(0),2';
@77968,P_node,SEQUENCE(1),@78000;
@78000,P_node,IF(4),@78032,@78112,NULL,NULL;
@78032,P_node,SIGNATURE(0),"-"';
@78112,P_node,SEQUENCE(2),@78144,@78296;
@78144,P_node,SIGNATURE(0),",."';
@78296,P_node,SIGNATURE(0),3';
@78600,P_node,SEQUENCE(1),@78632;
@78632,P_node,IF(4),@78664,@78744,NULL,NULL;
@78664,P_node,SIGNATURE(0),"-"';
@78744,P_node,SEQUENCE(2),@78776,@78928;
@78776,P_node,SIGNATURE(0),",."';
@78928,P_node,SIGNATURE(0),4';
@79232,P_node,SEQUENCE(1),@79264;
@79264,P_node,IF(4),@79296,@79376,NULL,NULL;
@79296,P_node,SIGNATURE(0),"-"';
@79376,P_node,SEQUENCE(2),@79408,@79560;
@79408,P_node,SIGNATURE(0),",."';
```

```
@79560,P_node,SIGNATURE(0),5';
@79680,P_node,SIGNATURE(0),tts!collect_digits(port=Port_nu
m, prompt=speech_text,max_digits=1);
@79912,P_node,NEXTSTATE(1),@79856;
@79856,P_node,STATE(2),@79768,@113360;
@79768,P_node,LEAF(1),$state,'Wait_For_Vote';
@113360,P_node,SEQUENCE(1),@113392;
@113392,P_node,EVENT(2),@113424,@113480;
@113424,P_node,LEAF(1),$event,'collected_digits';
@113480,P_node,SEQUENCE(1),@113512;
@113512,P_node,IF(4),@113544,@113600,NULL,@117456;
@113544,P_node,SIGNATURE(0),Max_Cand';
@113600,P_node,SEQUENCE(1),@113704;
@113704,P_node,SEQUENCE(1),@113736;
@113736,P_node,TEST(3),@113768,@113800,NULL;
@113768,P_node,SIGNATURE(0),collected_digits.digits';
@113800,P_node,CASE(10),@113864,@113920,@115496,@115584,@1
15992,
@116048,@116456,@116544,@113864,P_node,SIGNATURE(0),'1';
@113920,P_node,SEQUENCE(1),@115248;
@115248,P_node,SEQUENCE(1),@115280;
@115280,P_node,SIGNATURE(0),tts!send(port=Port_num,text=sp
eech_ text);
@115496,P_node,SIGNATURE(0),'2';
@115584,P_node,SEQUENCE(1),@115744;
@115744,P_node,SEQUENCE(1),@115776;
@115776,P_node,SIGNATURE(0),tts!send(port=Port_num,text=sp
eech_ text);
@115992,P_node,SIGNATURE(0),'3';
@116048,P_node,SEQUENCE(1),@116208;
@116208,P_node,SEQUENCE(1),@116240;
@116240,P_node,SIGNATURE(0),tts!send(port=Port_num,text=sp
eech_ text);
@116456,P_node,SIGNATURE(0),'4';
@116544,P_node,SEQUENCE(1),@116704;
@116704,P_node,SEQUENCE(1),@116736;
@116736,P_node,SIGNATURE(0),tts!send(port=Port_num,text=sp
eech_ text);
@116952,P_node,SIGNATURE(0),'5';
@117016,P_node,SEQUENCE(1),@117176;
@117176,P_node,SEQUENCE(1),@117208;
@117208,P_node,SIGNATURE(0),tts!send(port=Port_num,text=sp
eech_ text)';
@117456,P_node,SEQUENCE(1),@117488;
@117488,P_node,IF(4),@117520,@117576,NULL,@117840;
@117520,P_node,SIGNATURE(0),"9"';
@117576,P_node,SEQUENCE(2),@117608,@117808;
@117608,P_node,SIGNATURE(0),tts!collect_digits(port=Port_n
um, prompt=speech_text,max_digits=4
@117808,P_node,NEXTSTATE(1),@117752;
```

```
@117752,P_node,STATE(2),@117664,NULL;
@117664,P_node,LEAF(1),$state,'Wait_For_Pin';
@117840,P_node,SEQUENCE(1),@117872;
@117872,P_node,SIGNATURE(0),'tts!send(port=Port_num,text="
Sorry, incorrect choice.")';
@79976,P_node,EVENT(2),@80008,@80064;
@80008,P_node,LEAF(1),$event,'bri!terminating_call';
@80064,P_node,SEQUENCE(1),@80168;
@80168,P_node,SEQUENCE(1),@80200;
@80200,P_node,SIGNATURE(0),bri!send_answer(port=Port_num)
@109552,P_node,SEQUENCE(2),@109584,@109640;
@109584,P_node,SIGNATURE(0),access';
@109640,P_node,SIGNATURE(0),dynamic';
@109728,P_node,SEQUENCE(3),@109760,@109936,@110112;
@109760,P_node,EVENT(2),@109792,@109848;
@109792,P_node,LEAF(1),$event,'tts!send_completed';
@109848,P_node,SEQUENCE(1),@109880;
@109880,P_node,SIGNATURE(0),'end_call';
@109936,P_node,EVENT(2),@109968,@110024;
@109968,P_node,LEAF(1),$event,'bri!disconnected';
@110024,P_node,SEQUENCE(1),@110056;
@110056,P_node,SIGNATURE(0),'end_call';
@110112,P_node,EVENT(2),@110144,@110200;
@110144,P_node,LEAF(1),$event,'collect_failed';
@110200,P_node,SEQUENCE(1),@110232;
@110232,P_node,SIGNATURE(0),'end_call';
@110320,P_node,SEQUENCE(3),@109488,@80320,@79856;
@80320,P_node,STATE(2),@80232,@80376;
@80232,P_node,LEAF(1),$state,'Wait_For_Pin';
@80376,P_node,SEQUENCE(1),@80408;
@80408,P_node,EVENT(2),@80440,@80496;
@80440,P_node,LEAF(1),$event,'collected_digits';
@80496,P_node,SEQUENCE(1),@80528;
@80528,P_node,IF(4),@80560,@80616,NULL,@113240;
@80560,P_node,SIGNATURE(0),CO1_PIN';
@80616,P_node,SEQUENCE(1),@80720;
@80720,P_node,SEQUENCE(7),@80752,@82088,@82584,@83128,@836
40, @84152,@113152;
@80752,P node,SIGNATURE(0),are,"';
@82088,P_node,SEQUENCE(1),@82120;
@82120,P_node,IF(4),@82152,@82232,NULL,NULL;
@82152,P_node,SIGNATURE(0),"-"';
@82232,P_node,SEQUENCE(1),@82264;
@82264,P_node,SIGNATURE(0),string(Election_Counter1);
@82584,P_node,SEQUENCE(1),@82616;
@82616,P_node,IF(4),@82648,@82728,NULL,NULL;
@82648,P_node,SIGNATURE(0),"-"';
@82728,P_node,SEQUENCE(1),@82760;
@82760,P_node,SIGNATURE(0),string(Election_Counter2);
@83128,P_node,SEQUENCE(1),@83160;
```

```
@83160,P_node,IF(4),@83192,@83272,NULL,NULL;
@83192,P_node,SIGNATURE(0),"-"';
@83272,P_node,SEQUENCE(1),@83304;
@83304,P_node,SIGNATURE(0),string(Election_Counter3);
@83640,P_node,SEQUENCE(1),@83672;
@83672,P_node,IF(4),@83704,@83784,NULL,NULL;
@83704,P_node,SIGNATURE(0),"-"';
@83784,P_node,SEQUENCE(1),@83816;
@83816,P_node,SIGNATURE(0),string(Election_Counter4);
@84152,P_node,SEQUENCE(1),@84184;
@84184,P_node,IF(4),@84216,@112920,NULL,NULL;
@84216,P_node,SIGNATURE(0),"-"';
@112920,P_node,SEQUENCE(1),@112952;
@112952,P_node,SIGNATURE(0),string(Election_Counter5);
@113152,P_node,SIGNATURE(0),'tts!send(port=Port_num,text=s
peech_ text';
@113240,P_node,SEQUENCE(1),@113272;
@113272,P_node,SIGNATURE(0),'tts!send(port=Port_num,text=
"Sorry, no authorization.")';
```

用于电子表决 *SDL/GR* 的上述中间码保存了原始 *SDL/GR* 表示支的足够细节，比如说能有效地优化中间码。还有，通过简单地把中间码转换为一个给定 *EE* 的合适目标码，中间码可以在任何不同 *EE* 内执行。原始的 *SDL/GR* 表示法和中间码的基本结构不需要重新设计，或是改变以适合一个特定类型的 *EE*。

尽管上面详细描述主要按照特定的 *SCE* 和 *EE* 的应用来描述本发明，但应强调，所讨论的实施例仅仅是示例性的。在所给的结构中可以作许多改变，包括比如：电信系统的类型，业务生成和执行环境的类型，中间码语法和在系统接口中用以转换和产生码的传送的次数。对于本领域中的熟练技术人员来说，这些和其它的改变和变化将是很显然的，因此本发明只是被附加的权利要求所限制。

1/10

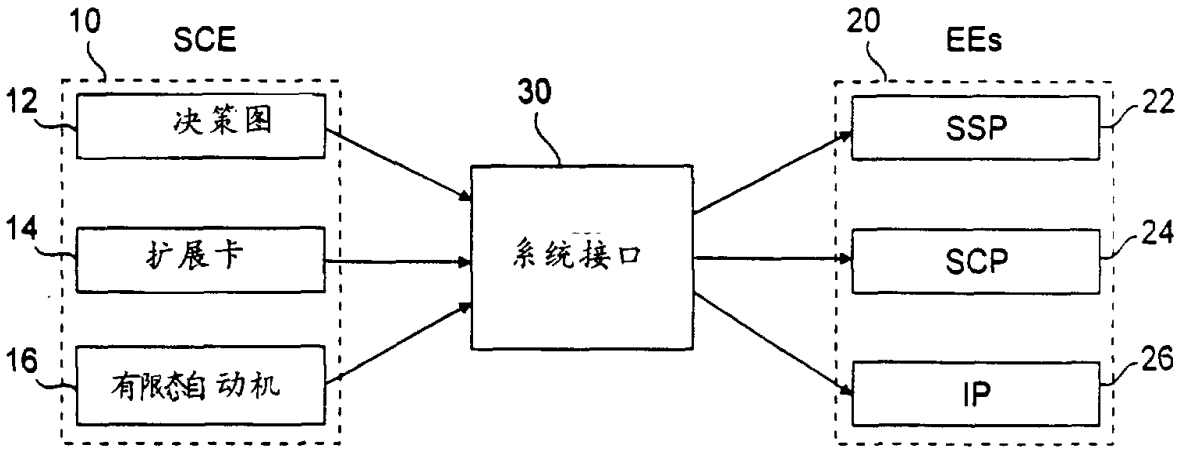

10
SCE
12
决策图
14
扩展卡
16
有限态自动机
30
系统接口
20
EEs
SSP
22
SCP
24
IP
26

图. 1

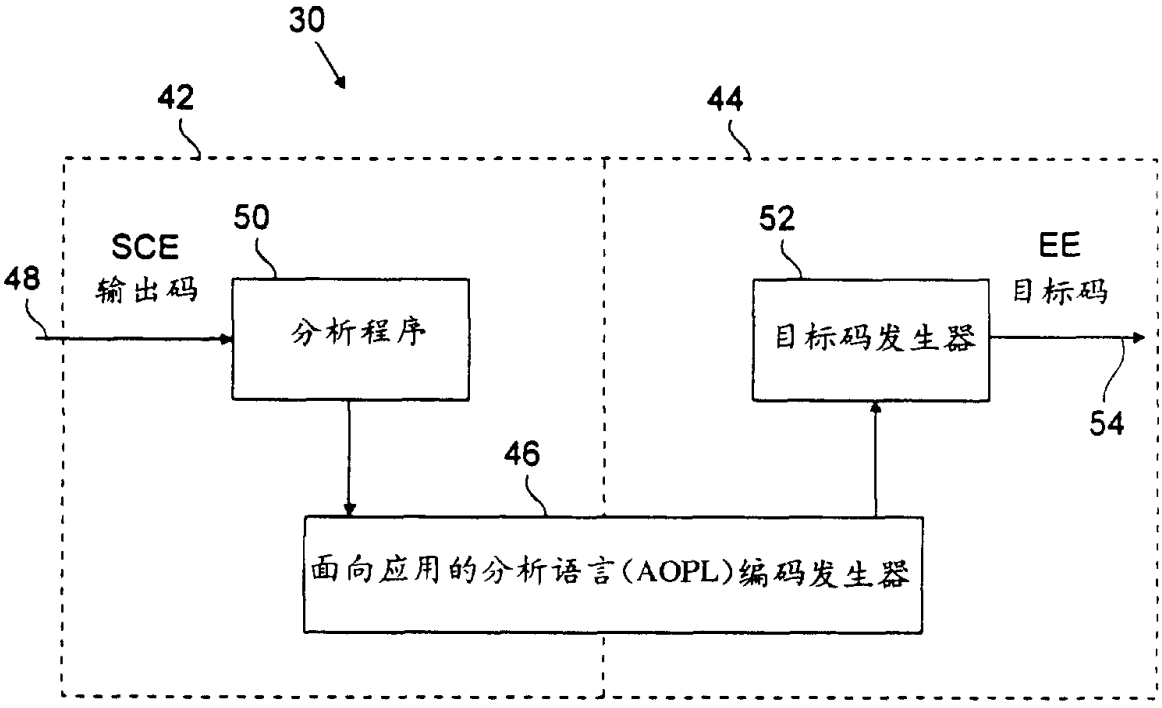

30
42
44
SCE
输出码
48
50
分析程序
52
EE
目标码
目标码发生器
54
46
面向应用的分析语言(AOPL)编码发生器

图. 2

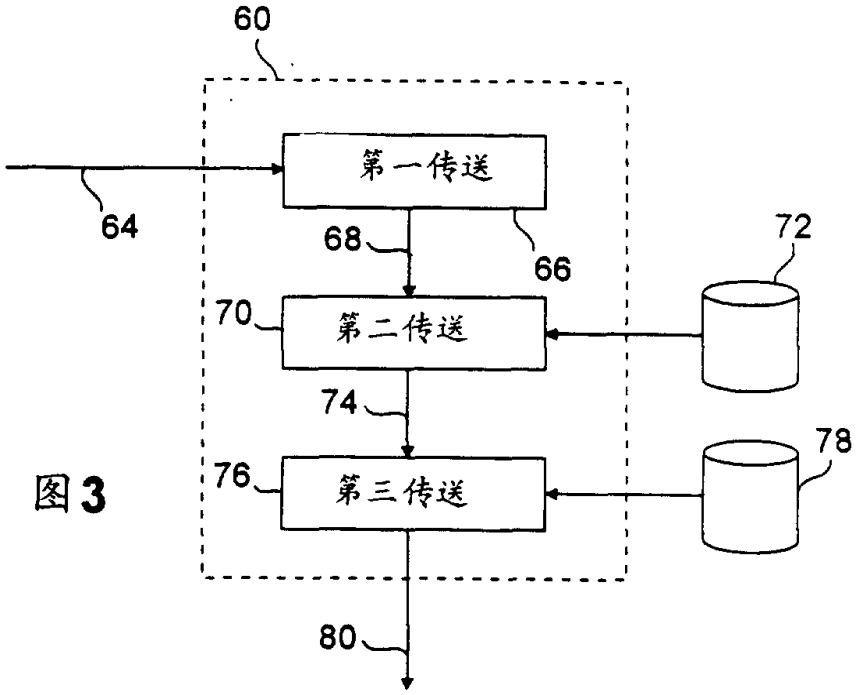

60
第一传送
64
68
66
72
70
第二传送
74
78
76
第三传送
80

图 3

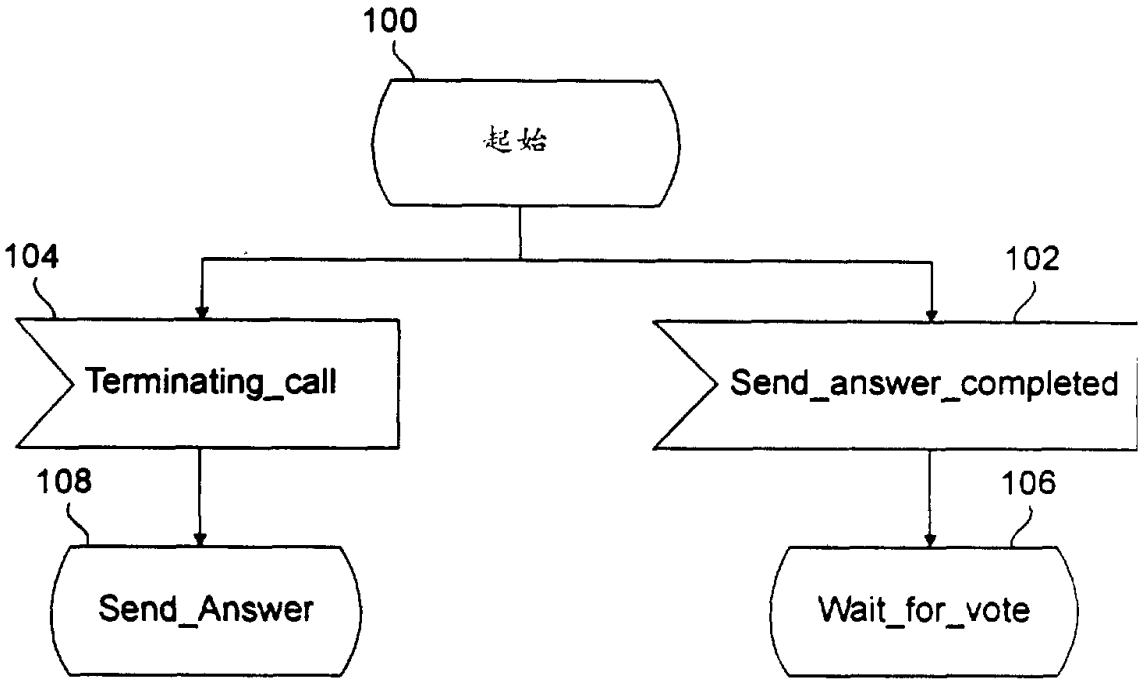

100
起始
104
102
Terminating_call
Send_answer_completed
108
106
Send_Answer
Wait_for_vote

图 4

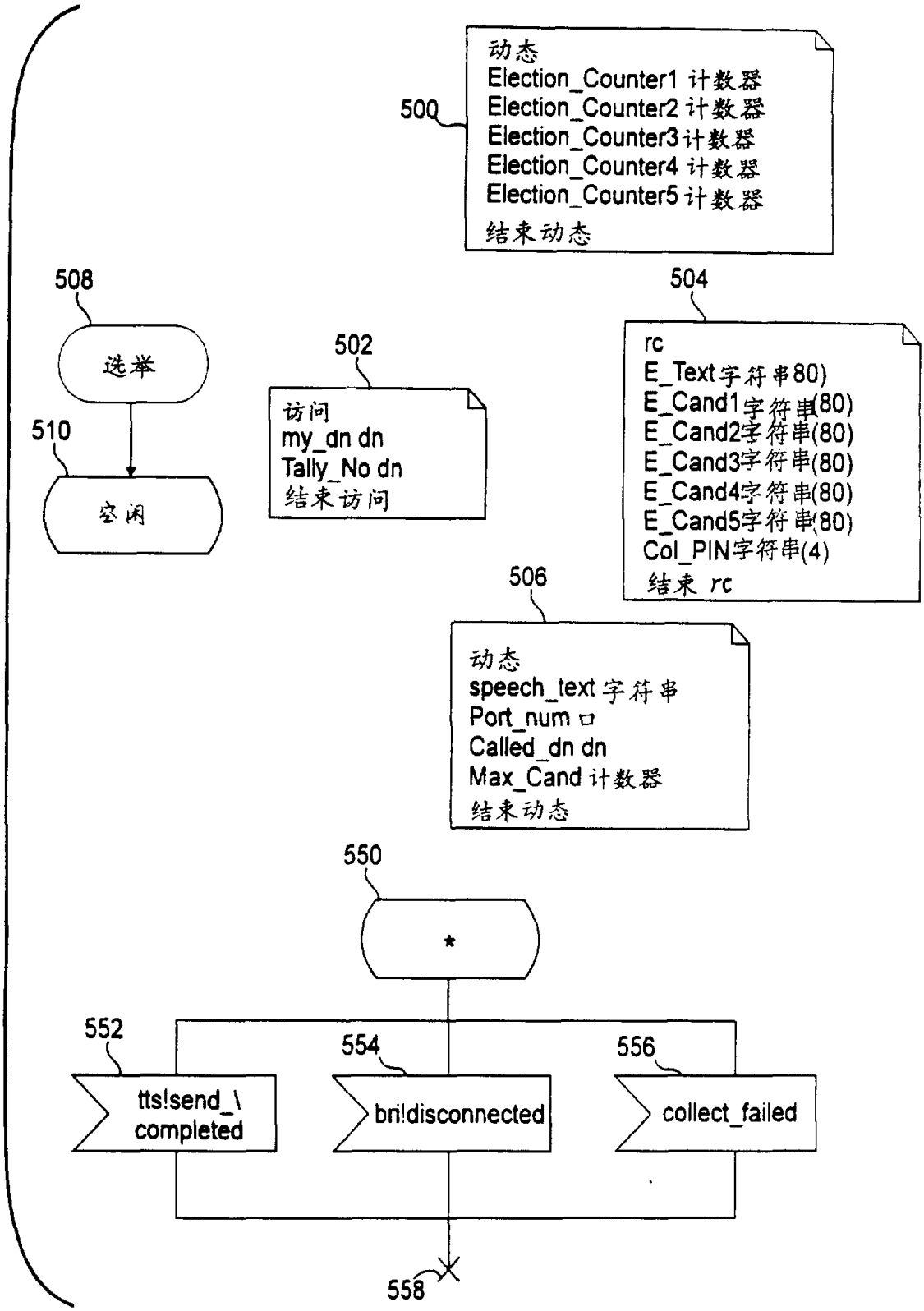

动态
Election_Counter1 计数器
Election_Counter2 计数器
Election_Counter3 计数器
Election_Counter4 计数器
Election_Counter5 计数器
结束动态
500
508
选举
510
空闲
502
访问
my_dn dn
Tally_No dn
结束访问
504
rc
E_Text字符串80)
E_Cand1字符串(80)
E_Cand2字符串(80)
E_Cand3字符串(80)
E_Cand4字符串(80)
E_Cand5字符串(80)
Col_PIN字符串(4)
结束 rc
506
动态
speech_text 字符串
Port_num 口
Called_dn dn
Max_Cand 计数器
结束动态
550
*
552
tts!send_\
completed
554
bri!disconnected
556
collect_failed
558

图5

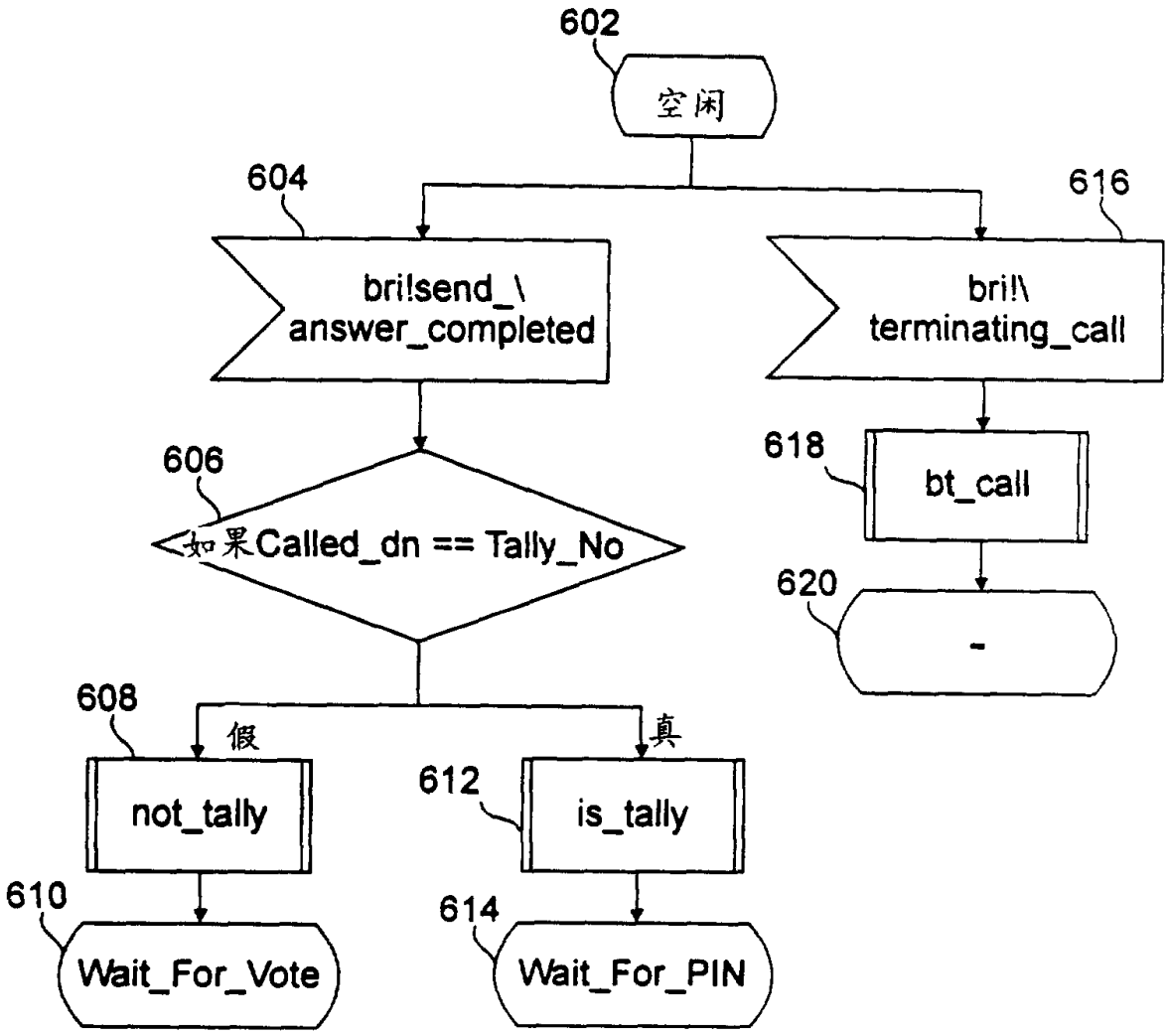

602
空闲
604
bri!send_\
answer_completed
616
bri!\
terminating_call
606
如果Called_dn == Tally_No
618
bt_call
620
-
608
假
真
not_tally
612
is_tally
610
Wait_For_Vote
614
Wait_For_PIN

图6(a)

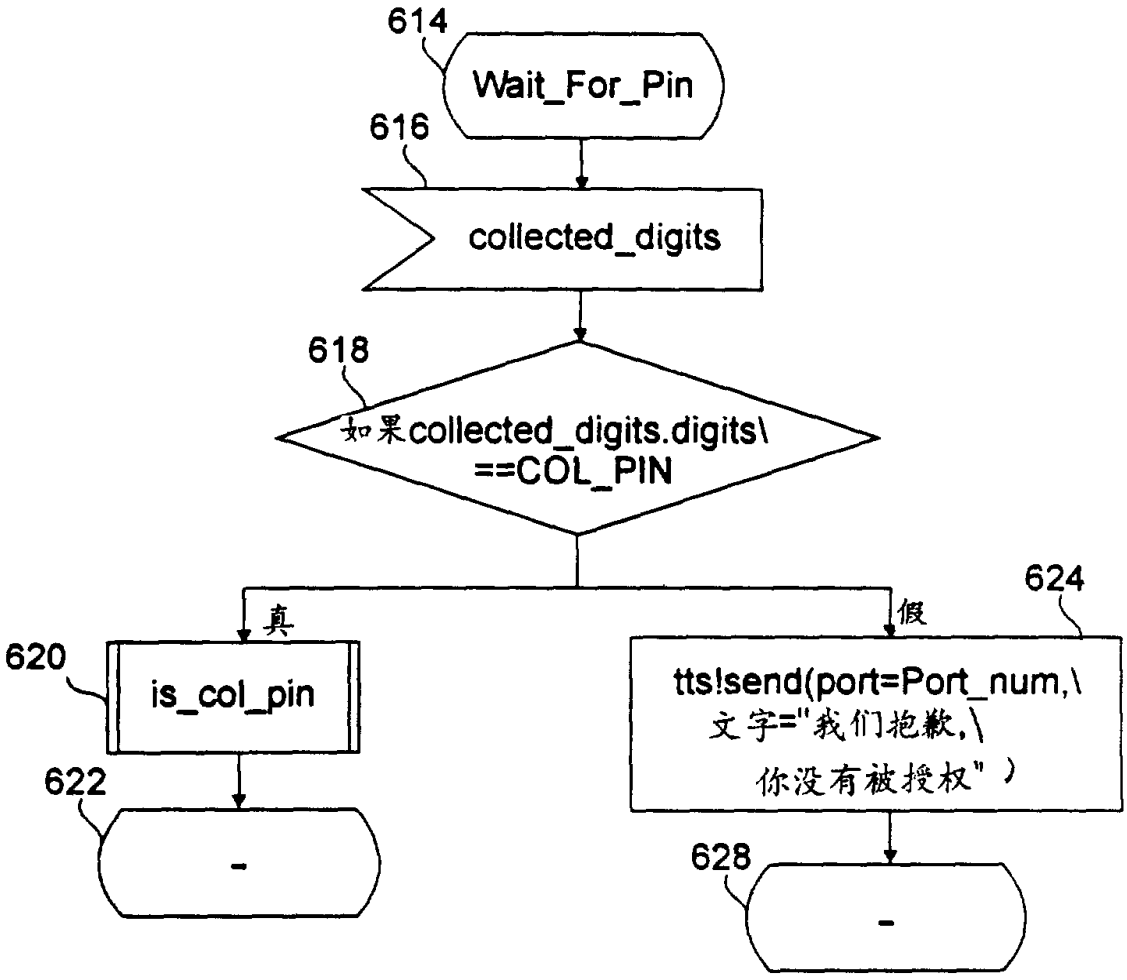

614
Wait_For_Pin
616
collected_digits
618
如果collected_digits.digits\
==COL_PIN
真
假
620
is_col_pin
622
-
624
tts!send(port=Port_num,\
文字="我们抱歉,\
你没有被授权")
628
-

图6(b)

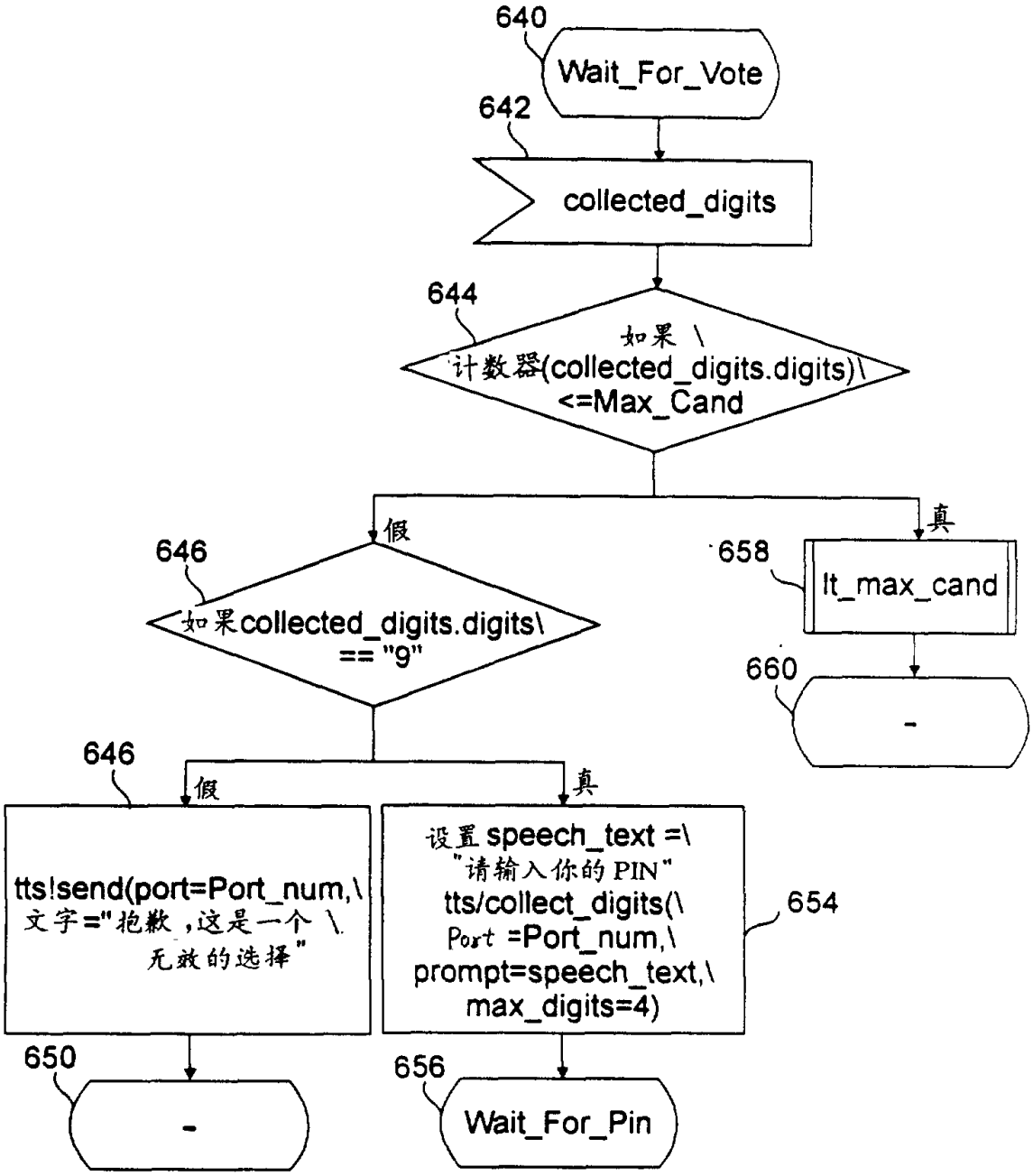

640
Wait_For_Vote
642
collected_digits
644
如果 \
计数器(collected_digits.digits)\
<=Max_Cand
假
真
646
如果collected_digits.digits\
== "9"
658
lt_max_cand
660
-
646
假
真
tts!send(port=Port_num,\
文字="抱歉，这是一个 \
无效的选择"
设置 speech_text =\
"请输入你的 PIN"
tts/collect_digits(\
Port =Port_num,\
prompt=speech_text,\
max_digits=4)
654
650
-
656
Wait_For_Pin

图6(c)

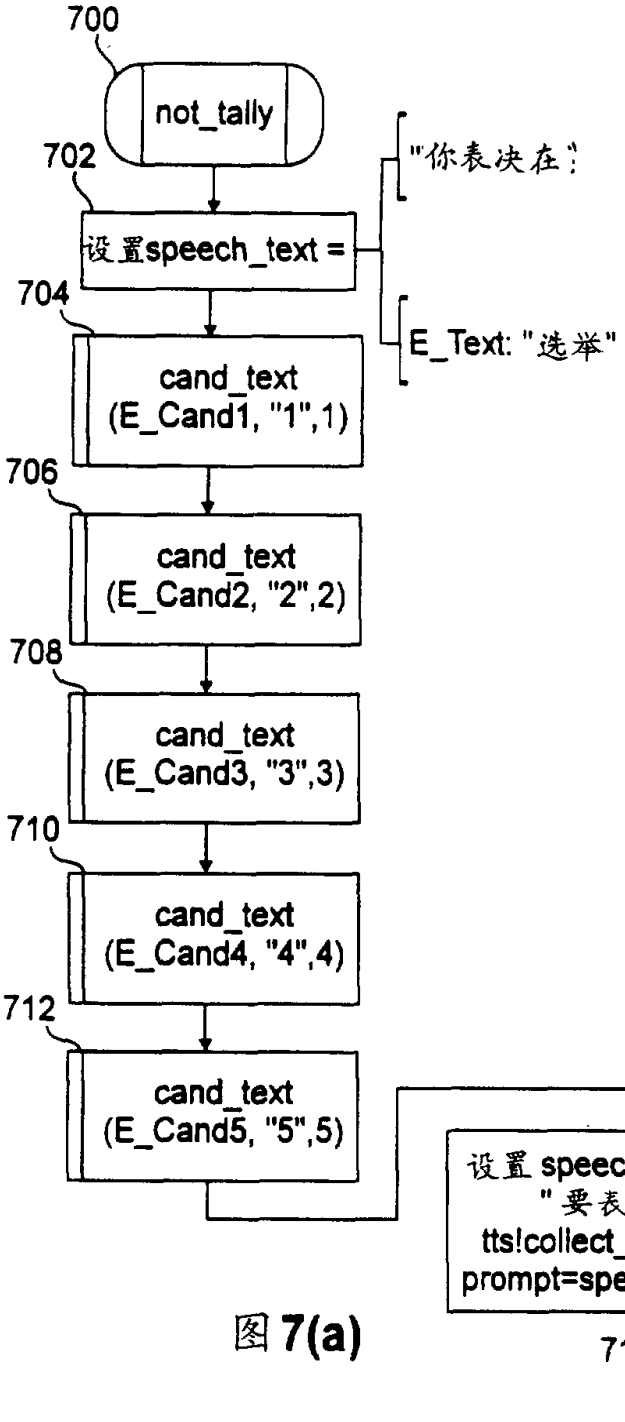

700
not_tally
702
"你表决在:"
设置speech_text =
704
E_Text: "选举"
cand_text
(E_Cand1, "1",1)
706
cand_text
(E_Cand2, "2",2)
708
cand_text
(E_Cand3, "3",3)
710
cand_text
(E_Cand4, "4",4)
712
cand_text
(E_Cand5, "5",5)
714
设置 speech_text = speech_text:\
"要表决总数，按 9"
tts!collect_digits(port=Port_num,\
prompt=speech_text, max_digits=1)
715

图 7(a)

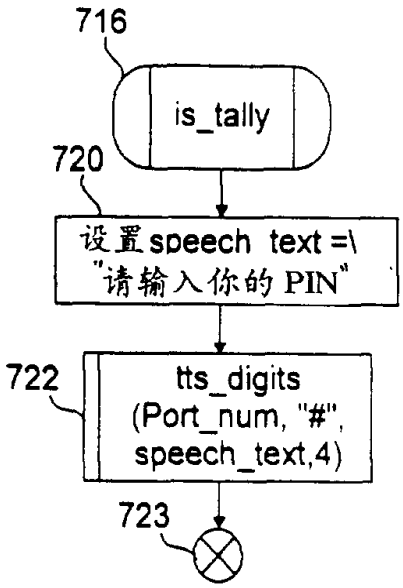

716
is_tally
720
设置speech text =\
"请输入你的 PIN"
722
tts_digits
(Port_num, "#",
speech_text,4)
723

图 7(b)

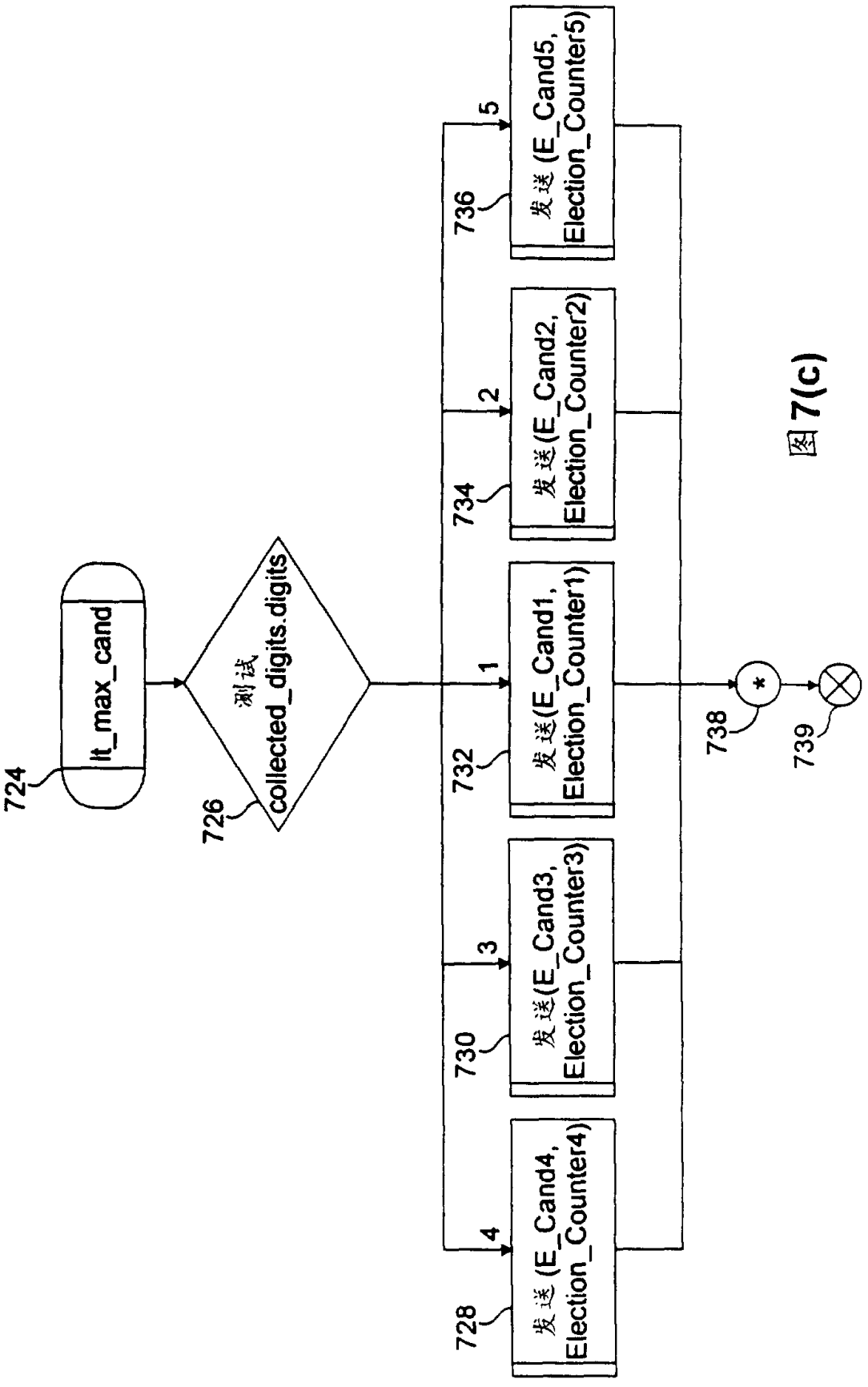

It_max_cand
724
测试
collected_digits.digits
726
1
发送(E_Cand1, Election_Counter1)
732
2
发送(E_Cand2, Election_Counter2)
734
3
发送(E_Cand3, Election_Counter3)
730
4
发送(E_Cand4, Election_Counter4)
728
5
发送(E_Cand5, Election_Counter5)
736
*
738
739

图 7(c)

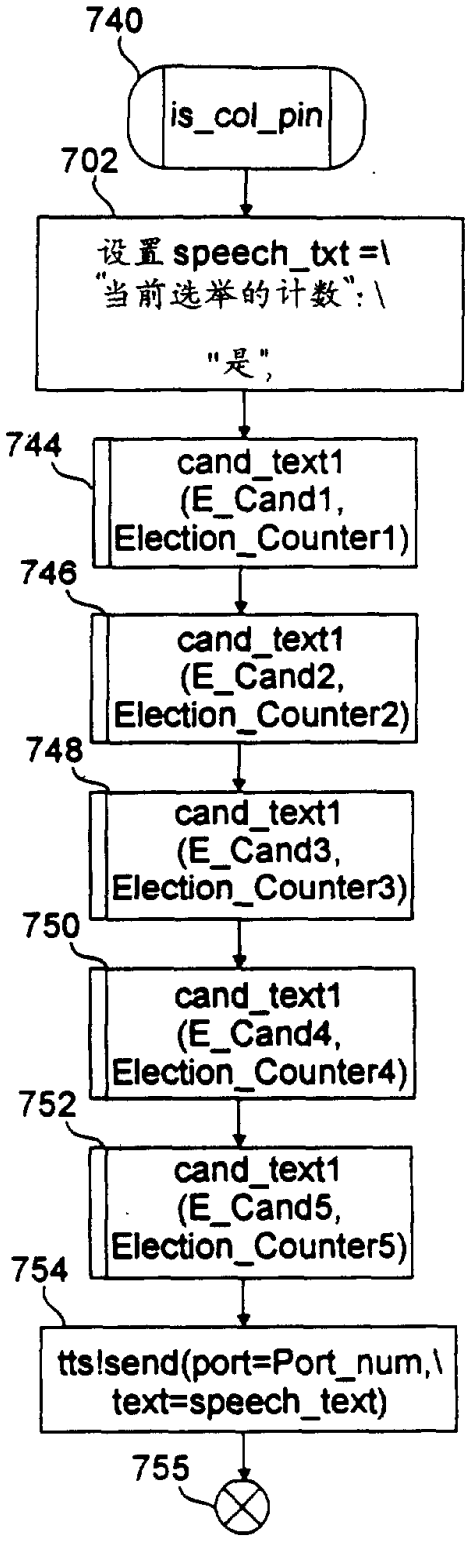

740
is_col_pin
702
设置 speech_txt =\
"当前选举的计数": \
"是",
744
cand_text1
(E_Cand1,
Election_Counter1)
746
cand_text1
(E_Cand2,
Election_Counter2)
748
cand_text1
(E_Cand3,
Election_Counter3)
750
cand_text1
(E_Cand4,
Election_Counter4)
752
cand_text1
(E_Cand5,
Election_Counter5)
754
tts!send(port=Port_num,\
text=speech_text)
755

图7(d)

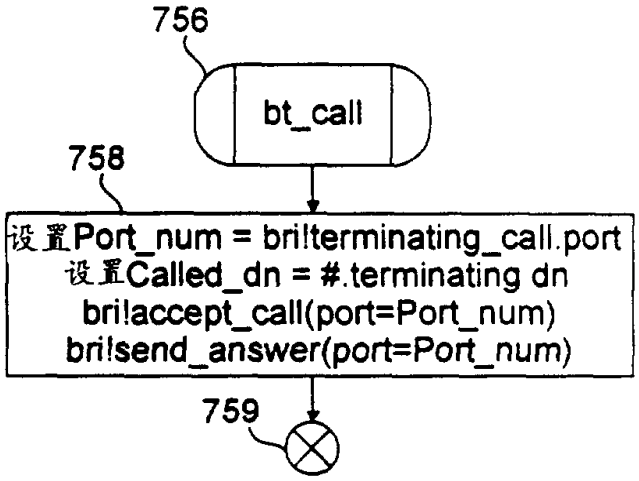

756
bt_call
758
设置Port_num = bri!terminating_call.port
设置Called_dn = #.terminating dn
bri!accept_call(port=Port_num)
bri!send_answer(port=Port_num)
759

图7(e)

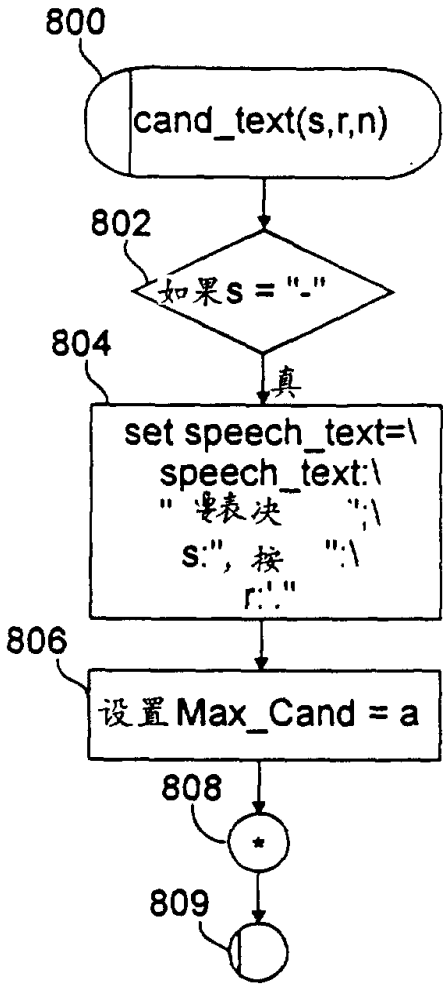


图 8(a)

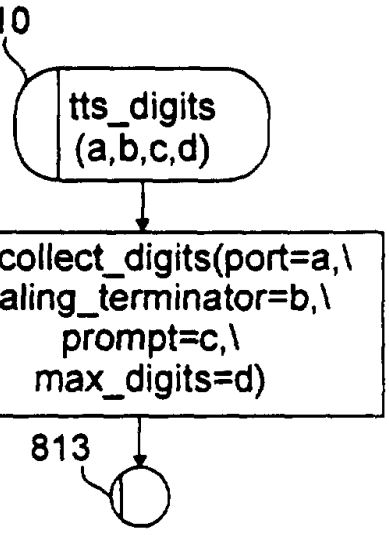


图 8(b)

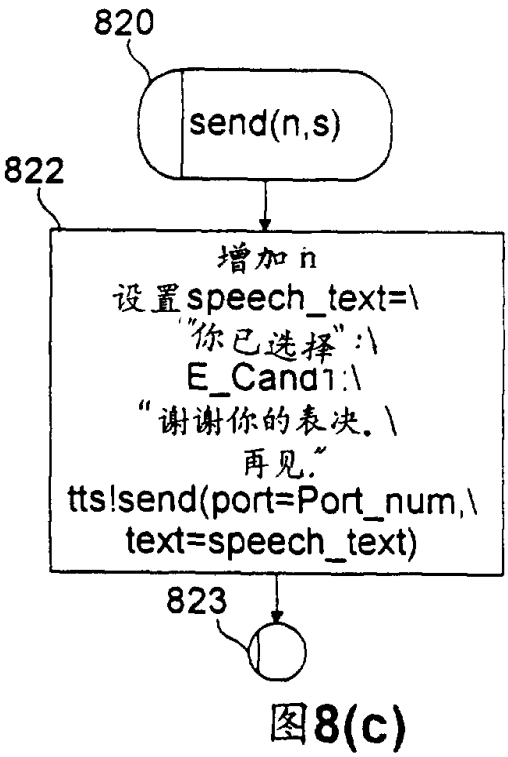


图8(c)

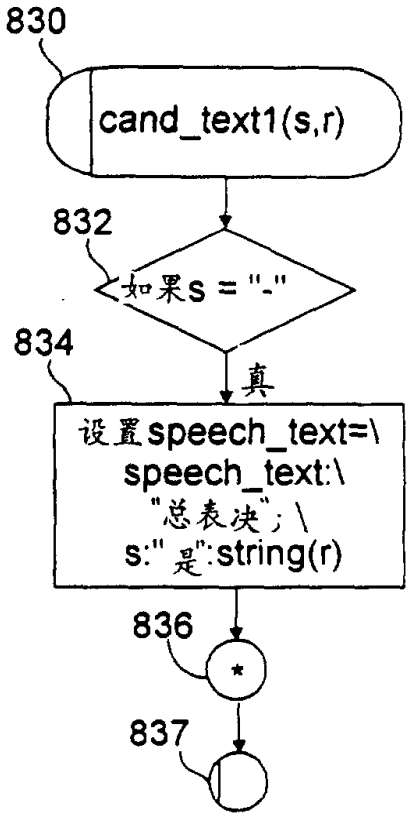


图8(d)