[19]中华人民共和国专利局

[51]Int.Cl[6]
G06F 15/00

[12] 发明专利申请公开说明书

[21] 申请号 96121524.0

[43]公开日 1997 年 9 月 10 日

[11] 公开号 CN 1159033A

[22]申请日 96.12.12

[30]优先权

[32]95.12.19[33]US[31]574,820 / 95

[71]申请人 国际商业机器公司

地址 美国纽约

[72]发明人 弗兰克·威廉姆·吉尔切斯特
厄里克·奈尔斯·赫尼斯
厄里克·H·简尼
约翰·克里斯托弗·里泊斯切
乔治·詹姆斯·罗马诺

[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

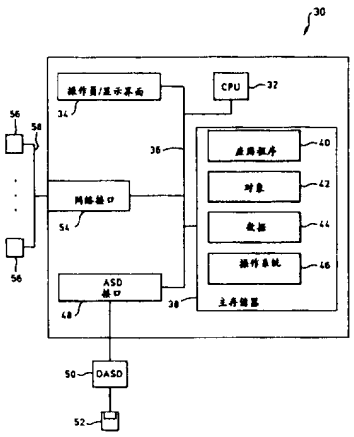
代理人 于 静

权利要求书 17 页 说明书 54 页 附图页数 34 页

[54]发明名称 面向对象的邮递服务器框架机构

[57]摘要

用于面向对象的编程系统的框架提供了公共消息处理系统结构，该结构能够被置于任何 OOP 平台上并得到适当配置以支持任何电子邮件消息协议标准或具体的邮递服务器功能。该框架把电子邮件消息定义为若干不同的对象，这些对象每一个都包含描述消息的某一部分的信息。实施了该框架的系统所接收的所有消息，都被定义在其核心对象结构上。另一组对象和方法定义了邮递服务器处理消息所需的处理步骤。



权 利 要 求 书

1.一种计算机系统，包括：

中央处理单元；

用户接口；以及

主存储器，它具有支持面向对象的程序设计环境的操作系统，该环境包含一个框架，该框架提供了可扩展的网络邮递服务器处理系统，而邮递服务器处理系统从发出电子邮件消息的网络用户接收电子邮件消息并将该电子邮件消息传送给该电子邮件消息的预期接收者的一或多个网络用户的目的地地址。

2.根据权利要求1的计算机系统，其中该框架定义了：控制消息处理的消息中心对象；消息类，它包含一组消息对象，而这些消息对象包括包含在电子邮件消息中的发出者、接收者和消息内容信息；以及，一组对象方法，它们被消息中心对象用来根据电子邮件消息的消息处理协议而将电子邮件消息的信息放置在消息对象中并对其进行相应的处理。

3.根据权利要求2的计算机系统，其中消息对象包括属于这样的类的对象，该类包括标明发出网络用户的网络地址的消息发出者清单、标识电子邮件消息的预期接收者的接收者清单、和包含消息协议的消息属性信息的信封清单。

4.根据权利要求3的计算机系统，其中消息发出者清单类包括定义发出网络用户地址和消息协议类型的对象。

5.根据权利要求4的计算机系统，其中消息协议类型由发出用户网络地址对象定义。

6.根据权利要求3的计算机系统，其中接收者清单类包括由接收者用户网络地址和接收者地址消息协议类型定义的对象。

7.根据权利要求6的计算机系统，其中电子邮件消息的接收者清单对象指定了多个不同的消息协议类型。

8.根据权利要求3的计算机系统，其中信封清单类包括由消息协议类型定义的对象。

9.根据权利要求8的计算机系统，其中电子邮件消息的信封清单对象指定了多个不同的消息协议类型。

10.根据权利要求3的计算机系统，其中消息对象进一步包括属于标明附在电子邮件消息上的信息的附件访问类的对象。

11.根据权利要求10的计算机系统，其中附件访问类对象指定了附在电子邮件消息上的信息的多个不同的协议类型。

12.根据权利要求3的计算机系统，其中对象方法包括：消息发出者清单产生方法，它产生发出者清单对象；接收者清单产生方法，它产生接收者清单对象；以及，信封清单产生方法，它产生信封对象。

13.根据权利要求3的计算机系统，其中对象方法包括信封清单产生方法，该产生方法确定了消息协议类型并产生相应的信封对象。

14.根据权利要求13的计算机系统，其中对象方法进一步包括加上方法，该方法将电子邮件消息放置在消息队列中以由消息中心对象进行处理。

15.根据权利要求13的计算机系统，其中对象方法进一步包括一个检索方法，该方法从消息队列中检索电子邮件消息，以由消息中心对象进行处理。

16.根据权利要求2的计算机系统，其中对象方法包括一个地址分解方法，该方法确定接收者清单类的每一个成员的接收者用户网络地址。

17.根据权利要求2的计算机系统，其中对象方法包括安全核准方法，该方法进行电子邮件消息的消息安全检查和核准。

18.根据权利要求2的计算机系统，其中对象方法包括附件访问清单方法，它产生包含附在电子邮件消息上的信息的附件清单对象。

19.根据权利要求2的计算机系统，其中消息对象进一步包括属于由消息发出者清单、原始接收者清单、接收者清单和信封清单组成的类的对象。

20.根据权利要求19的计算机系统，其中原始接收者清单类包含一个或多个条目对象，这些对象每一个都包含由接收者用户网络地址和分发类型定义的一个或多个对象。

21.根据权利要求20的计算机系统，其中接收者清单类包括由接收者

用户网络地址和接收者地址消息协议类型定义的对象。

22.根据权利要求21的计算机系统，其中各个接收者用户网络地址指定了消息协议类型。

23.根据权利要求21的计算机系统，其中消息对象包括不同消息协议的多个接收者清单对象。

24.根据权利要求21的计算机系统，其中对象方法包括接收者清单分解方法，该方法从原始接收者清单条目对象产生接收者清单对象。

25.根据权利要求24的计算机系统，其中接收者清单分解方法在原始接收者清单条目对象上反复进行操作，直到分发类型不再指定额外的目的地地址，从而使分发类型指定与单个的原始接收者清单条目对应的一组目的地地址。

26.一种计算机系统，包括：

中央处理单元；

用户接口；

网络接口，通过它接收电子邮件消息；以及

主存储器，它具有支持面向对象的程序设计环境的操作系统，该环境包含一个框架，该框架定义了：控制消息处理的消息中心对象；包含一组消息对象的消息类，这些消息对象定义了包含在接收的电子邮件消息中的发出者、接收者和消息内容信息；以及一组对象方法，这些对象方法被消息中心对象用来根据电子邮件消息的消息处理协议而将包含在接收的电子邮件消息中的信息置于消息对象中并对其进行相应的处理。

27.根据权利要求26的计算机系统，其中消息对象包括属于这样的类的对象，这些类包括了标明发出网络用户的网络地址的消息发出者清单、标明电子邮件消息的预期接收者的清单、以及包含消息协议的消息属性信息的信封清单。

28.根据权利要求27的计算机系统，其中消息发出者清单类包括定义发出用户网络地址和消息协议类型的对象。

29.根据权利要求28的计算机系统，其中消息协议类型由发出用户网络地址对象定义。

30.根据权利要求27的计算机系统，其中接收者清单类包括由接收者用户网络地址和接收者地址消息协议类型定义的对象。

31.根据权利要求30的计算机系统，其中电子邮件消息的接收者清单对象指定了多个不同的消息协议类型。

32.根据权利要求27的计算机系统，其中信封清单类包括由消息协议类型定义的对象。

33.根据权利要求32的计算机系统，其中电子邮件消息的信封清单对象指定了多个不同的消息协议类型。

34.根据权利要求27的计算机系统，其中消息对象进一步包括属于一个附件访问类的对象，该附件访问类标明了附在电子邮件消息上的信息。

35.根据权利要求34的计算机系统，其中附件访问类对象指定了附在电子邮件消息上的信息的多个不同协议类型。

36.根据权利要求27的计算机系统，其中对象方法包括产生发出者清单对象的发出者清单产生方法、产生接收者清单对象的接收者清单产生方法、以及产生信封对象的信封清单产生方法。

37.根据权利要求27的计算机系统，其中对象方法包括信封清单产生方法，该产生方法确定消息协议类型并产生相应的信封对象。

38.根据权利要求37的计算机系统，其中对象方法进一步包括相加方法，该相加方法将电子邮件消息置于消息队列中，以由消息中心对象进行处理。

39.根据权利要求37的计算机系统，其中对象方法进一步包括检索方法，该检索方法从消息队列中检索电子邮件消息，以由消息中心对象进行处理。

40.根据权利要求27的计算机系统，其中对象方法包括一个地址分解方法，该方法为接收者清单类的每一个成员确定了一个接收者用户网络地址。

41.根据权利要求27的计算机系统，其中对象方法包括进行电子邮件消息的消息安全检查和核准的安全核准方法。

42.根据权利要求27的计算机系统，其中对象方法包括附件访问清单

方法，该方法产生包含附在电子邮件消息上的信息的附件清单对象。

43.根据权利要求27的计算机系统，其中消息对象进一步包括属于这样一些类的对象—这些类包括消息发出者清单、原始接收者清单、接收者清单和信封清单。

44.根据权利要求43的计算机系统，其中原始接收者清单类包含一或多个条目对象，其每一个都包含由接收者用户网络地址和分发类型定义的一或多个对象。

45.根据权利要求44的计算机系统，其中接收者清单类包括由接收者用户网络地址和接收者地址消息协议类型定义的对象。

46.根据权利要求45的计算机系统，其中各个接收者用户网络地址指定了消息协议类型。

47.根据权利要求45的计算机系统，其中消息对象包括不同消息协议的多个接收者清单对象。

48.根据权利要求45的计算机系统，其中对象方法包括接收者清单分解方法，该方法从原始接收者清单条目对象产生接收者清单对象。

49.根据权利要求48的计算机系统，其中接收者清单分解方法在原始接收者清单条目对象上反复进行操作，直到分发类型不再指定额外的目的地地址，从而使分发类型指定了与单个的原始接收者清单条目相对应的一组目的地地址。

50. 一种在与网络联接的计算机系统中使用的面向对象的框架，该计算机系统从发出电子邮件的消息的网络用户接收电子邮件消息并具有支持面向对象的程序设计环境的操作系统，该环境保持着：控制电子邮件消息处理的消息中心对象；一个消息类，它包括一组消息对象，该消息对象包括在接收的电子邮件消息中所包含的发出者、接收者和消息内容信息；以及，一组对象方法，这些对象方法被消息中心对象用来根据接收的电子邮件消息的消息处理协议而将接收的电子邮件消息的信息置入消息对象并对其进行相应的处理。

51.根据权利要求50的框架，其中消息类对象包括属于这样的类的对象—即这些类包括：消息发出者清单，它标明发出电子邮件消息的网络用

户的网络地址；接收者清单，它标明电子邮件消息的预期接收者；以及，信封清单，它包含消息协议的消息属性信息。

52.根据权利要求51的框架，其中消息发出者清单类包括定义发出电子邮件信息的用户网络地址和消息协议类型的对象。

53.根据权利要求52的框架，其中消息协议类型由发出电子邮件信息的用户网络地址对象定义。

54.根据权利要求51的框架，其中接收者清单类包括由接收者用户网络地址和接收者地址消息协议类型定义的对象。

55.根据权利要求54的框架，其中电子邮件消息的接收者清单对象指定了多个不同的消息协议类型。

56.根据权利要求51的框架，其中信封清单类包括由消息协议类型定义的对象。

57.根据权利要求56的框架，其中电子邮件消息的信封清单对象指定了多个不同的消息协议类型。

58.根据权利要求51的框架，其中消息对象进一步包括属于一个附件访问类的对象，该附件访问类标明了附在电子邮件消息上的信息。

59.根据权利要求58的框架，其中附件访问类对象指定了附在电子邮件消息上的信息的多个不同的协议类型。

60.根据权利要求51的框架，其中对象方法包括产生发出者清单对象的发出者清单产生方法、产生接收者清单对象的接收者清单产生方法、以及产生信封对象的信封清单产生方法。

61.根据权利要求51的框架，其中对象方法包括确定消息协议类型并产生相应的信封对象的信封清单产生方法。

62.根据权利要求61的框架，其中对象方法进一步包括一个相加方法，该相加方法将电子邮件消息置于消息队列中以由消息中心对象进行处理。

63.根据权利要求61的框架，其中对象方法进一步包括检索方法，该检索方法从消息队列检索电子邮件消息，以由消息中心对象进行处理。

64.根据权利要求51的框架，其中对象方法包括为接收者清单类的每

一个成员确定接收者用户网络地址的地址分解方法。

65.根据权利要求51的框架，其中对象方法包括进行电子邮件消息的消息安全检查和核准的安全核准方法。

66.根据权利要求51的框架，其中对象方法包括一个附件访问清单方法，该方法产生包含附在电子邮件消息上的信息的附件清单对象。

67.根据权利要求51的框架，其中消息对象进一步包括属于包括消息发出者清单、原始接收者清单、接收者清单和信封清单的类的对象。

68.根据权利要求67的框架，其中原始接收者清单类包含一或多个条目对象，其每一个都包含由接收者用户网络地址和分发类型定义的一或多个对象。

69.根据权利要求68的框架，其中接收者清单类包括由接收者用户网络地址和接收者地址消息协议类型定义的对象。

70.根据权利要求69的框架，其中各个接收者用户网络地址指定了消息协议类型。

71.根据权利要求69的框架，其中消息对象包括不同消息协议的多个接收者清单对象。

72.根据权利要求69的框架，其中对象方法包括接收者清单分解方法，该方法从原始接收者清单条目对象产生接收者清单对象。

73.根据权利要求72的框架，其中接收者清单分解方法反复在原始接收者清单条目对象上进行操作，直到分发类型不再指定额外的目的地地址，从而使分发类型指定了与单个的原始接收者清单条目相对应的一组目的地地址。

74.一种用于计算机系统中的程序产品，该计算机系统具有支持面向对象的程序设计环境的操作系统，该程序产品包括：

可记录介质；

记录在该可记录介质上的框架，该框架提供了可扩展网络邮递服务器处理系统，该系统从发出电子邮件消息的网络用户接收电子邮件消息并将该电子邮件消息传送到作为该电子邮件消息的预期接收者的网络用户的一或多个目的地地址。

75.根据权利要求74的程序产品，其中框架定义了：一个消息中心对象，该消息中心对象控制消息处理；一个消息类，它包括一组消息对象，该消息对象包括包含在电子邮件消息中的发出者、接收者和消息内容信息；以及，一组对象方法，这些方法被消息中心对象用来根据电子邮件消息的消息处理协议而将电子邮件消息的信息置入消息对象并对其进行相应的处理。

76.根据权利要求75的程序产品，其中消息中心对象包括属于这样的类的对象，即这些类包括标明发出电子邮件消息的网络用户的网络地址的消息发出者清单、标明电子邮件消息的预期接收者的接收者清单、以及包含消息协议的消息属性信息的信封清单。

77.根据权利要求76的程序产品，其中消息发出者清单类包括定义发出电子邮件消息的用户网络地址和消息协议类型的对象。

78.根据权利要求77的程序产品，其中消息协议类型由发出电子邮件消息的用户网络地址对象定义。

79.根据权利要求76的程序产品，其中接收者清单类包括由接收者用户网络地址和接收者地址消息协议类型定义的对象。

80.根据权利要求79的程序产品，其中电子邮件消息的接收者清单对象指定了多个不同的消息协议类型。

81.根据权利要求76的程序产品，其中信封清单类包括由消息协议类型定义的对象。

82.根据权利要求81的程序产品，其中电子邮件消息的信封清单对象指定了多个不同的消息协议类型。

83.根据权利要求76的程序产品，其中消息对象包括属于一个附件访问类的对象，该附件访问类标明了附在电子邮件消息上的信息。

84.根据权利要求83的程序产品，其中附件访问类对象指定了附在电子邮件消息上的信息的多个不同的协议类型。

85.根据权利要求76的程序产品，其中对象方法包括产生发出者清单对象的发出者清单产生方法、产生接收者清单对象的接收者清单产生方法、以及产生信封对象的信封清单产生方法。

86.根据权利要求76的程序产品，其中对象方法包括确定消息协议类型并产生相应的信封对象的信封清单产生方法。

87.根据权利要求86的程序产品，其中对象方法进一步包括相加方法，该相加方法将电子邮件消息置于消息队列中以由消息中心对象进行处理。

88.根据权利要求86的程序产品，其中对象方法进一步包括获取方法，该方法从消息队列中获取电子邮件消息，以由消息中心对象进行处理。

89.根据权利要求76的程序产品，其中对象方法包括地址分解方法，该方法为接收者清单类的每一个成员确定接收者用户网络地址。

90.根据权利要求76的程序产品，其中对象方法包括安全核准方法，该方法进行电子邮件消息的消息安全检查和核准。

91.根据权利要求76的程序产品，其中对象方法包括附件访问清单方法，该方法产生包含附在电子邮件消息上的信息的附件清单对象。

92.根据权利要求76的程序产品，其中消息对象进一步包括属于包括消息发出者清单、原始接收者清单、接收者清单和信封清单类的对象。

93.根据权利要求92的程序产品，其中原始接收者清单类包含一或多个条目对象，其每一个都包含由接收者用户网络地址和分发类型定义的一或多个对象。

94.根据权利要求93的程序产品，其中接收者清单类包括由接收者用户网络地址和接收者地址消息协议类型定义的对象。

95.根据权利要求94的程序产品，其中各个接收者用户网络地址指定了消息协议类型。

96.根据权利要求94的程序产品，其中消息对象包括不同消息协议的多个接收者清单对象。

97.根据权利要求94的程序产品，其中对象方法包括接收者清单分解方法，该方法从原始接收者清单条目对象产生接收者清单对象。

98.根据权利要求97的程序产品，其中接收者清单分解方法在原始接收者清单条目对象上反复进行操作，直到分发类型不再指定额外的目的地地址，从而使分发类型指定了与单个的原始接收者清单条目相对应的一组

目的地地址。

99.用于分发程序产品的方法，该方法包括以下步骤：

在网络上的第一计算机系统与第二计算机系统之间建立联接；以及

从第一计算机系统向第二计算机系统传送该程序产品，其中该程序产品包括一个面向对象的框架，该框架当在第二计算机系统的面向对象的程序设计环境中执行时提供了可扩展的网络邮递服务器处理系统，从而使该可扩展网络邮递服务器处理系统从始发网络用户接收电子邮件消息并将该电子邮件消息传送到作为该电子邮件消息的预期接收者的网络用户的一或多个目的地地址。

100.根据权利要求99的程序产品分发方法，其中传送的程序产品的框架定义了：一个消息中心对象，该消息中心对象控制第二计算机系统中的消息处理；消息类，它包含由电子邮件消息中的发出者、接收者和消息内容信息组成的一组消息目标；以及，一组对象方法，它们被消息中心对象用来根据电子邮件消息的消息处理协议而将电子邮件消息的信息置于消息对象中并对其进行相应的处理。

101.根据权利要求100的程序产品分发方法， 其中消息中心对象包括属于这样一些类的对象，即这些类包括标明始发网络用户的网络地址的消息发出者清单、标明电子邮件消息的预期接收者的接收者清单、以及包含消息协议的消息属性信息的信封清单。

102.根据权利要求101的程序产品分发方法， 其中消息发出者清单类包括定义发出用户网络地址和消息协议类型的对象。

103.根据权利要求102的程序产品分发方法， 其中消息协议类型由发出用户网络地址对象定义。

104.根据权利要求101的程序产品分发方法， 其中接收者清单类包括由接收者用户网络地址和接收者地址消息协议类型定义的对象。

105.根据权利要求104的程序产品分发方法， 其中电子邮件消息的信封清单对象指定了多个不同的消息协议类型。

106.根据权利要求101的程序产品分发方法， 其中信封清单类包括由

消息协议类型定义的对象

107.根据权利要求106的程序产品分发方法，其中电子邮件消息的信封清单对象指定了多个不同的消息协议类型。

108.根据权利要求101的程序产品分发方法，其中消息对象进一步包括属于标明附在电子邮件消息上的附件访问类的对象。

109.根据权利要求108的程序产品分发方法，其中附件访问类对象指定了附在电子邮件消息上的信息的多个不同的协议类型。

110.根据权利要求101的程序产品分发方法，其中该程序产品的对象方法包括产生发出者清单对象的发出者清单产生方法、产生接收者清单对象的信封清单产生方法、和产生信封对象的信封清单产生方法。

111.根据权利要求101的程序产品分发方法，其中该程序产品的对象方法包括确定消息协议类型并产生相应的信封对象的信封清单产生方法。

112.根据权利要求111的程序产品分发方法，其中对象方法进一步包括一个相加方法，该相加方法将电子邮件消息置于消息队列中以由消息中心对象进行处理。

113.根据权利要求111的程序产品分发方法，其中对象方法进一步包括一个检索方法，该检索方法从消息队列获取电子邮件消息以由消息中心对象进行处理。

114.根据权利要求101的程序产品分发方法，其中对象方法包括一个地址分解方法，该方法为接收者清单类的每一个成员确定接收者用户网络地址。

115.根据权利要求101的程序产品分发方法，其中对象方法包括进行消息安全检查和消息核准的安全核准方法。

116.根据权利要求101的程序产品分发方法，其中对象方法包括一个附件访问清单方法，该方法产生包含附在电子邮件消息上的信息的附件清单对象。

117.根据权利要求101的程序产品分发方法，其中消息对象进一步包括属于由消息发出者清单、原始接收者清单、接收者清单和信封清单组成的类的对象。

118.根据权利要求117的程序产品分发方法，其中原始接收者清单类包含一或多个条目对象，这些条目对象每一个都包含由接收者用户网络地址和分发类型定义的一或多个对象

119.根据权利要求118的程序产品分发方法，其中接收者清单类包括由接收者用户网络地址和接收者地址消息协议类型定义的对象。

120.根据权利要求119的程序产品分发方法，其中每一个接收者用户网络地址指定了一个消息协议类型。

121.根据权利要求119的程序产品分发方法，其中消息对象包括不同消息协议的多个接收者清单对象。

122.根据权利要求119的程序产品分发方法，其中对象方法包括接收者清单分解方法，该方法从原始接收者清单条目对象产生信封清单对象。

123.根据权利要求122的程序产品分发方法，其中接收者清单分解方法反复在原始接收者清单条目对象上进行操作，直到分发类型不再指定额外的目的地地址，从而使分发类型指定了与单个的原始接收者清单条目相对应的一组目的地地址。

124.一种在计算机系统中执行一个应用程序的方法，该计算机系统具有：控制计算机系统的处理的中央处理单元、用户接口、网络接口和具有支持面向对象的程序设计环境的操作系统的主存储器，该方法包括以下步骤：

提供一个面向对象的框架，该框架提供了可扩展的网络邮递服务器处理系统；

接收对框架的用户定义扩展，从而产生一个可执行的网络邮递服务器处理系统；

执行该可执行网络邮递服务器处理系统，从而接收来自始发网络用户的电子邮件消息并将该电子邮件消息传送到作为该电子邮件消息的预期接收者的网络用户的一或多个目的地地址。

125.根据权利要求124的方法，其中所提供的框架定义了：一个消息中心对象，它控制计算机系统中的消息处理；一个消息类，它包含一组消息对象，这些消息对象包括在电子邮件消息中的发出者、接收者和消息内

容信息；以及，一组对象方法，这些对象方法被消息中心对象用来根据电子邮件消息的消息处理协议而将电子邮件消息的信息置于消息对象中并对其进行相应的处理。

126.根据权利要求125的方法，其中所提供的框架的消息类对象包括这样的对象，即这些对象包括标明始发网络用户的网络地址的消息发出者清单、标明电子邮件消息的对象接收者的接收者清单、以及包含消息协议的消息属性信息的信封清单。

127.根据权利要求126的方法，其中消息发出者清单类包括定义始发用户网络地址和消息协议类型的对象。

128.根据权利要求127的方法，其中消息协议类型由始发用户网络地址对象定义。

129.根据权利要求126的方法，其中接收者清单类包括由接收者用户网络地址和接收者地址消息协议类型定义的对象。

130.根据权利要求129的方法，其中电子邮件消息的接收者清单对象指定了多个不同的消息协议类型。

131.根据权利要求126的方法，其中信封清单类包括由消息协议类型定义的对象。

132.根据权利要求131的方法，其中电子邮件消息的信封清单对象指定了多个不同的消息协议类型。

133.根据权利要求126的方法，其中消息对象进一步包括属于一个附件访问类的对象，该附件访问类标明了附在电子邮件消息上的信息。

134.根据权利要求133的方法，其中附件访问类对象指定了附在电子邮件消息上的信息的不同协议类型。

135.根据权利要求126的方法，其中所提供的框架的对象方法包括产生发出者清单对象的消息发出者清单产生方法产生接收者清单对象的接收者清单产生方法和产生信封对象的信封清单产生方法。

136.根据权利要求126的方法，其中所提供的框架的对象方法包括确定消息协议类型并产生相应的信封对象的信封清单产生方法。

137.根据权利要求136的方法，其中对象方法进一步包括一个相加方

法，该相加方法将电子邮件消息置于一个消息队列中以由消息中心对象进行处理。

138.根据权利要求136的方法，其中对象方法进一步包括一个检索方法，该方法从消息队列检索电子邮件消息以由消息中心对象进行处理。

139.根据权利要求126的方法，其中对象方法包括一个地址分解方法，该方法为接收者清单类的每一个成员确定了接收者用户网络地址。

140.根据权利要求126的方法，其中所提供的框架的对象方法包括一个安全核准方法，该方法进行电子邮件消息的消息安全检查和核准。

141.根据权利要求126的方法，其中所提供的框架的对象方法包括一个附件访问清单方法，该方法产生包含附在电子邮件消息上的信息的附件清单对象。

142.根据权利要求126的方法，其中消息对象进一步包括属于这样的类的对象，即这些类包括消息发出者清单、原始接收者清单、接收者清单和信封清单。

143.根据权利要求142的方法，其中原始接收者清单类包含一或多个条目对象，这些条目对象每一个都包含由接收者用户网络地址和分发类型定义的一或多个对象。

144.根据权利要求143的方法，其中接收者清单类包括由接收者用户网络地址和接收者地址消息协议类型定义的对象。

145.根据权利要求144的方法，其中每一个接收者用户网络地址都指定了消息协议类型。

146.根据权利要求144的方法，其中消息对象包括不同消息协议的多个接收者清单对象。

147.根据权利要求144的方法，其中对象方法包括接收者清单分解方法，该方法从原始接收者清单条目对象产生接收者清单对象。

148.根据权利要求147的方法，其中接收者清单分解方法反复在原始接收者清单条目对象上进行操作，直到分发类型不再指定额外的目的地地址，从而使分发类型指定了与单个的原始接收者清单条目相对应的一组目的地地址。

149.一种用于处理在一个计算机系统中接收的消息的方法，该计算机系统具有控制计算机系统中的处理的中央处理单元、用户接口、网络接口、和具有支持一个面向对象的程序设计环境的操作系统的主存储器，该方法包括以下步骤：

在该操作系统中安装一个面向对象的框架，该框架提供了可扩展网络邮递服务器处理系统；

给该框架提供扩展，这些框架定义了具有预定电子邮件消息协议的具体消息对象并定义了根据这些预定协议来处理电子邮件消息的具体对象方法；

用框架扩展产生一个可执行网络邮递服务器处理系统；

用可执行网络邮递服务器处理系统来处理接收的电子邮件消息，从而将从始发网络用户接收的电子邮件消息传送到作为该电子邮件消息的预期接收者的网络用户的一或多个目的地地址。

150.根据权利要求149的方法，其中安装的框架定义了：控制消息处理的消息中心对象；一个消息类，它包含一组消息对象，这些消息对象包括包含在电子邮件消息中的发出者、接收者和消息内容信息；以及，一组对象方法，这些对象方法被消息中心对象用来根据接收的电子邮件消息的消息处理协议而将电子邮件消息的信息置于消息对象中并对其进行相应的处理。

151.根据权利要求150的方法，其中安装的框架的消息中心对象包括这样的对象，即这些对象包括标明始发网络用户的网络地址的消息发出者清单、标明接收电子邮件消息的预期接收者的接收者清单、以及包含消息协议的消息属性信息的信封清单。

152.根据权利要求151的方法，其中消息发出者清单类包括定义始发用户网络地址和消息协议类型的对象。

153.根据权利要求152的方法，其中消息协议类型由始发用户网络地址对象定义。

154.根据权利要求151的方法，其中接收者清单类包括由接收者用户网络地址和接收者地址消息协议类型定义的对象。

155.根据权利要求 154 的方法，其中电子邮件消息的接收者清单对象指定了多个不同的消息协议类型。

156.根据权利要求 151 的方法，其中信封清单类包括由消息协议类型定义的对象。

157.根据权利要求 156 的方法，其中电子邮件消息的信封清单对象指定了多个不同的消息协议类型。

158.根据权利要求 151 的方法，其中消息对象进一步包括属于一个附件访问类的对象，该附件访问类标明了附在电子邮件消息上的信息。

159.根据权利要求 158 的方法，其中附件访问类对象指定了附在电子邮件消息上的信息的多个不同的协议类型。

160.根据权利要求 151 的方法，其中安装的框架的对象方法包括产生发出者清单对象的消息发出者清单产生方法、产生接收者清单对象的接收者清单产生方法、和产生信封对象的信封清单产生方法。

161.根据权利要求 151 的方法，其中安装的框架的对象方法包括确定消息协议类型并产生相应的信封对象的信封清单产生方法。

162.根据权利要求 161 的方法，其中对象方法进一步包括一个相加方法，该方法将接收的电子邮件消息置于一个消息队列中，以由消息中心对象进行处理。

163.根据权利要求 161 的方法，其中对象方法进一步包括一个检索方法，该方法从消息队列中检索接收的电子邮件消息，以由消息中心对象进行处理。

164.根据权利要求 151 的方法，其中对象方法包括一个地址分解方法，该方法为接收者清单类的每一个成员确定接收者用户网络地址。

165.根据权利要求 151 的方法，其中对象方法包括一个安全核准方法，该方法进行电子邮件消息的消息安全检查和核准。

166.根据权利要求 151 的方法，其中对象方法包括一个附件访问清单方法，该方法产生包含附在电子邮件消息上的信息的附件清单对象。

167.根据权利要求 151 的方法，其中消息对象进一步包括属于这样的类的对象，即这些类包括消息发出者清单、原始接收者清单、接收者清单、

和信封清单。

168.根据权利要求167的方法，其中原始接收者清单类包含一或多个条目对象，这些对象每一个都包含由接收者用户网络地址和分发类型定义的一或多个对象。

169.根据权利要求168的方法，其中接收者清单类包括由接收者用户网络地址和接收者地址消息协议类型定义的对象。

170.根据权利要求169的方法，其中各个接收者用户网络地址指定了消息协议类型。

171.根据权利要求169的方法，其中消息对象包括不同消息协议的多个接收者清单对象。

172.根据权利要求169的方法，其中对象方法包括从原始接收者清单条目对象产生接收者清单对象的接收者清单分解方法。

173.根据权利要求172的方法，其中接收者清单分解方法反复在原始接收者清单条目对象上进行操作，直到分发类型不再指定额外的目的地地址，从而使分发类型指定了与单个的原始接收者清单条目相对应的一组目的地地址。

174.根据权利要求150的方法，其中安装的框架的消息中心对象：

将属于消息对象的接收者清单类的消息对象扩展成标明消息的对象接收者的用户网络地址的地址对象；

根据接收者地址将该地址对象分解成定义消息协议类型的分发类型对象；

为标明消息的预期接收者的每一个接收者清单对象产生信封对象，并根据信封对象定义的消息协议类型进行消息对象处理；以及

启动将消息送向标明的用户网络地址的分送处理。

说 明 书

面向对象的邮递服务器框架机构

本发明一般地说涉及数据处理，更具体地说是涉及面向对象的编程系统和处理。

众所周知的电子邮件，指的是从一个计算机用户通过互联的计算机网络送到另一用户的消息。支持电子邮件的计算机系统提供一种手段，用于制成消息、将它们从消息发出者传送到接收者、通知接收者并在消息被接收时向发出者报告，并以适当的格式放置消息以在网络上进行传送，从而便利了这种消息传送。

早期的电子邮件系统包括：在共同的计算机上的用户之间或在采用共同的数据处理设备的不同的计算机上的终端—终端消息传送。例如，某些早期的电子邮件系统采用了用于网络内通信的简单的文件传送协议，它规定了用相应的网络终端节点来标明发出者和接收者的预定消息标头数据，其后是消息的内容。很多现代的电子邮件系统支持包括 ASCII 文本、模拟传真数据、数字传真数据、数字语音数据、视频文本和其他内容的信息。

为满足对计算机间网络通信的需要，制定了很多通信协议，以定义更为通用的电子邮件消息系统。网络通信协议的两个例子，一是为按照国际商用机器公司（ IBM 公司）的系统网络结构规程（ Systems Network Architecture specification)进行网络通信所规定的系统网络结构分布服务（ Systems Network architecture Distribution Services(SNADS ））协议，二是所谓的简单邮件传送协议（ Simple Mail Transfer Protocol（ SMTP ））。其他的网络包括国际标准化组织（ International Organization for Standardization(ISO ））制定的面向对象的文本互换系统（ MOTIS ）标准和—基于 Unix 的网络协议 USENET 。

邮递服务器系统方便地处理根据邮递服务器系统的协议构成的消息，该协议经常被称为“本机”(native)协议。以不同的协议通信的用户之间的消息传送，通常必须通过网络网关处理器，后者将该消息由外部协议变换

成本机协议。因此，用于互联网络中的消息传送的网关，诸如在通常称为“互联网络”的网络上使用的网关，从另一个网关或从相联的网络接收电子邮件消息。在联接的各网络上的用户可以经过局域网络（LAN）联接而与该网关相联。如果消息是来自另一网关，则进行检查以判定消息中标明的接收者是否是本地的。即，接收者可能是在接收网关的LAN上的计算机用户。如果该接收者对于该网关来说是本地的，则消息被传送给网关网络信箱或接收者能够从中获取消息的其他分送装置。如果接收者不是本地的，则消息被送到另一网关。如果消息是来自发出者—即只产生该消息的网关LAN用户，则检查消息的格式和句法是否正确。该消息随后被作为从另一网关接收的消息对待，意味着它是在被送往另一网络的网关的途中。

因此，各个网关可以包括根据本机协议运行的计算机网络。各个网关都可能被要求识别从不同的协议产生的消息（并确定其适当信息）。例如，一个SNADS协议网关可能接收到MOTIS标准消息。如果该SNADS网关要将消息送到其预定的接收者，该网关必须首先将该消息转换或变换到SNADS协议，以使该消息能够在SNADS网关网络上行进。接收该消息的网关可能需要进行类似的转换。协议之间的转换，由于一个协议所支持的特征可能不为另一协议所知道，而可以变得非常困难。另外，一个协议可能基于与另一个协议不同的具体通信模型。例如，SNADS系统是基于通信会话的（带有伴生的消息参数），而SMTP系统模型则不是基于会话的。

很多电子邮件系统不能有效地进行所需的消息处理任务，以在不同的消息协议之间进行转换。另外，很多电子邮件系统在它们能够支持并管理它们原来为之设计的协议以外的协议之前，需要大量的修正。因此，随着新的协议的开发或系统用户决定使用非本机的协议，这种系统将需要高的维护费用。如果电子邮件系统网关能够迅速而有效地支持和管理多个协议之间的消息处理并适应新的协议，那将是有利的。

从以上的论述，可以看出需要一种电子邮件协议间的网关机构，它能够有效地按一种协议接收来自一发送地的电子邮件消息并按另一种协议将这种消息传送到一目的地，且它能够适应新的协议，从而能够接收和传送按新协议的消息。本发明满足了这种需要。

根据本发明，用于与面向对象的程序设计（OOP）系统一起使用的可再用面向对象（OO）结构，提供了通用的消息处理系统结构，它能够被置于任何 OOP 平台上并能够得到适当的构成以支持任何电子邮件消息协议标准或具体的邮递服务器功能。该消息处理系统结构是基于一组确定的对象类，这些对象类形成了面向对象的框架。该框架的用户可以方便地对框架进行剪裁，以适合它们的需要并提供邮递服务器系统，从而减小与用户/服务器电子邮件消息处理有关的开发和支持费用。框架的设计提供了用标准构成部件定义消息的对象结构，这些部件与 OOP 特征相一致并定义了属性和行为。以此方式，消息处理能够被分成一系列依次的步骤，或 OOP 对象方法，它处理消息的具体部分。

框架确定的一系列依次的步骤，代表着邮递服务器系统的有组织的中性实施。该框架将电子邮件消息定义为若干个分立的对象，其每一个都包含描述消息的某一部分的信息。实施该框架的系统接收的所有消息，可在该核心对象结构上得到定义。另一组对象和方法定义了邮递服务器处理消息所需的处理步骤。消息被作为一类消息对象来接收，而这些对象被指定了一个消息类型，该类型确定了消息对象随后所要经受的处理步骤。例如，一个消息可被指定为 SNADS 类消息类型或 SMTP 类消息类型。当处理消息时，它所包括的对象被改变，从而使消息处理能够被中断，且随后得到恢复，而不损失或重复处理步骤。

由于邮递服务器处理系统是以 OOP 框架的形式提供的，处理与具体的电子邮件协议对应的消息对象的对象方法，能够方便地被结合到框架的实施中，而不改变邮递服务器系统。框架用户得到保证，在框架的类和次类结构上定义的电子邮件功能对象方法将借助核心框架对象和方法进行操作，以按照所希望的方式处理消息。以此方式，框架用户能够定义处理新电子邮件协议的对象方法，以根据它们的具体需要来裁剪邮递服务器系统，而不改变整个系统，且不用对系统编程进行重新组合。这减少了对电子邮件网关系统改变具体的邮递服务器所需的时间和费用。

从以下对最佳实施例（它们以举例的方式显示了本发明的原理）的描述，可以理解本发明的其他特征和优点。

图 1 是显示了本发明的系统所实施的原理的动物园管理框架的示意图。

图 2、3、4、5、和 6 是用于图 1 的动物园管理框架的例子的类图。

图 7 是图 1 至 6 的框架的例子的对象图。

图 8 是根据本发明构成的计算机处理系统的功能框图。

图 9、10 和 11 是处理图，它显示了图 8 的计算机处理系统所进行的消息处理步骤。

图 12 是表示图 8 的计算机处理系统实施的面向对象的框架的分类图。

图 13 是图 8 所示的计算机处理系统实施的消息中心类别和有关类的类图。

图 14 是表示图 8 所示的计算机处理系统实施的消息类别的类图。

图 15 是表示图 8 的计算机处理系统实施的发出者清单类别的类图。

图 16 是表示图 8 所示的计算机处理系统所实施的信封清单类别的类图。

图 17 是表示图 8 所示的计算机处理系统所实施的接收者清单类别的类图。

图 18 是表示图 8 的计算机处理系统实施的附件访问清单类别的类图。

图 19 是表示图 8 的计算机处理系统实施的原始接收者清单类别的类图。

图 20 是表示图 8 的计算机处理系统实施的回答对象清单类别的类图。

图 21 是表示图 8 的计算机处理系统实施的报告对象清单类别的类图

图 22 是表示图 8 的计算机处理系统实施的报告内容清单类别的类图。

图 23 是表示图 8 的计算机处理系统实施的、属于消息中心类别的地址产生器清单类的类图。

图 24 是表示属于图 8 的计算机处理系统实施的、属于消息中心类别的信封产生器清单类的类图。

图 25 是表示属于图 8 的计算机处理系统实施的消息、属于中心类别的附件访问产生器清单类的类图。

图 26 是表示当处理进入的消息和产生消息对象时由图 8 所示的主处理

器执行的处理步骤的对象图。

图27是表示当产生地址对象时由图8所示的主处理器执行的处理步骤的对象图。

图28是表示当产生信封对象时由图8所示的主处理器执行的处理步骤的对象图。

图29是表示当产生附件对象时由图8所示的主处理器执行的处理步骤的对象图。

图30是表示当处理消息对象时由图8所示的主处理器执行的处理步骤的对象图。

图31是表示当扩展消息对象的未分解的接收者清单的地址条目对象时由图8的主处理器执行的处理步骤的对象图。

图32是表示当分解消息对象的未分解的接收者清单的地址条目对象时由图8的主处理器执行的处理步骤的对象图。

图33是对象图，表示了当处理消息的信封清单的信封对象时由图8所示的主处理器执行的处理步骤。

图34是对象图，显示了当处理消息的附件访问清单的附件访问对象时由图8的主处理器执行的处理步骤。

图35是对象图，显示了当进行邮递服务器系统的安全鉴别功能时由图8所示的主处理器执行的处理步骤。

图36是对象图，显示了当处理到达的消息以向本地（网络）分送时由图8所示的主处理器执行的处理步骤。

图37是对象图，显示了当处理到达的消息以送向另一网络时由图8所示的主处理器执行的处理步骤。

图38是对象图，显示了当处理到达的消息以将该消息送回或报告其内容时由图8所示的主处理器执行的处理步骤。

图39是对象图，显示了当处理消息的附件访问清单中的任何附件访问对象时，由图8所示的主处理器执行的处理步骤。

图40是对象图，显示了当进行邮递服务器系统的统计功能时由图8所示的主处理器执行的处理步骤。

面向对象技术的概述

如在概述部分中所述的，本发明是利用面向对象的框架技术开发出来的。面向对象的框架技术领域的人员可能希望进行到本说明书的详细描述部分。然而，对框架技术或一般的 OO 技术不那样熟悉的人，应该读一下本概述部分，以对本发明的好处和优点有最好的理解。

面向对象的技术与过程技术

虽然本发明涉及具体的面向对象的技术（即面向对象的框架技术），读者首先必须理解的是，一般地说 OO 技术明显地不同于传统的基于过程的技术（经常被称为过程技术）。虽然两种技术可被用来解决同一问题，但对问题的最终解决方案总是很不同的。这种不同来自于这一事实，即过程技术的设计焦点与 OO 技术的设计焦点完全不同。基于过程的设计的焦点在于解决问题的总体过程；而 OO 设计的焦点在于如何将问题分成一系列独立的实体，而这些实体能够一起用于提供一个解决方案。OO 技术的独立实体被称为对象。换言之，OO 技术与过程技术显著地不同，因为问题被分解成了相关的对象组，而不是嵌套的计算机程序或步骤的结构。即，过程技术以数据变量和过程函数来定义一个系统，而 OO 技术以对象和类来定义系统。

术语“框架”

已经建立起了很多术语和短语，它们对本领域的技术人员具有特定的含意。然而，读者应该注意的是，OO 领域中最不严格的一个定义就是词“框架”的定义。词“框架”对不同的人意味着不同的事物。因此，当比较两个假定的 OOP 框架的特性时，读者应该注意保证这种比较的确是“苹果与苹果”的比较。如在随后的描述中将说明的，术语“框架”在本说明书中用于描述一种 OO 技术系统，该系统已被设计成具有核心功能和扩展功能。其核心功能是不受框架购买者修正的框架部分。而扩展功能是明确设计成可被框架购买者将其作为其实施的一部分而用户化和扩展的那部分框架。

OO 框架

虽然一般地说 OO 框架能够被适当地描述为对编程问题的一种 OO 解决方案，但在框架和基本的 OO 编程解决方案之间仍然有基本的不同。这种不同在于框架是以这样的方式设计的，即允许并鼓励 OO 解决方案的某些方面的用户化和扩展，而基本的 OO 解决方案则包括类和对象的具体集合或库。换言之，框架提供了一个 OO 编程解决方案，它能够得到用户化和扩展，以应付随时间改变的要求。当然，框架的用户化/扩展性质对于购买者（以下称为框架用户）是非常有用的，因为用户化或扩展框架的费用远低于更换或重新设置已有编程 解决方案的费用。

因此，当框架设计者要解决具体问题时，它们应该不仅仅是设计单独的对象并规定这些对象如何相互联系。它们还应该设计框架的核心功能（即框架中不受框架用户的潜在用户化和扩展的部分）以及框架的扩展功能（即框架中受到潜在的用户化和扩展的部分）。总之，框架的最终价值不仅在于对象的设计质量，而且在于涉及框架的哪些方面代表核心功能且哪些方面代表扩展功能的设计选择。

ZAF —框架的一个例子

虽然本领域的技术人员明白框架设计一定是交错和迭代过程，在以下的描述中给出了用于简单框架的设计选择的一个例子。然而，应该理解的是，这只是框架的一个例子，它在本说明书中被用来显示并最佳地描述框架，以使读者能够对本发明的利益和优点有更好的理解。

框架设计者，通过从所谓的问题域中选出对象，而确定框架机构所需的对象。问题域是所面临的具体问题的抽象理解。为显示框架而选择的问题域的一个例子，即动物园管理框架（ZAF），它协助动物园管理者看管和喂养动物园的动物。 OO 框架设计者将研究动物园问题域并决定任何 ZAF 将必须涉及代表动物园管理人员与动物之间的关系（即代表动物园工作人员如何管理动物）的抽象。框架设计者还应该认识到动物园动物通常生活在笼子、水池等之中。因此，框架设计者还应该从这样的想法出发，

即框架应该涉及代表所有这些基本实体和关系的抽象和机构。

如何设计 ZAF

为了开始设计，框架设计者可能从所谓的类别图开始。类别图被用来在高层描述框架，并定义框架各组成部分彼此的关系。图 1 是用于示例性的框架 ZAF 的类别图。图 1 以及本说明书的其他图中的记号，将在本部分的结束处的记号部分中得到详细描述。类别图中的各个实体或图符，代表执行具体功能的数据对象的分组。为了说明，假定框架设计者决定 ZAF 应该由四个部分组成，这些部分在高层描述中，将被称为机构：动物园管理机构、动物园工作人员机构、动物机构、以及圈养单元机构。

如图 1 所示，动物园管理机构已设计成利用动物园工作人员机构管理动物园。因而说动物园管理机构与动物园工作人员机构具有“利用”关系。（请参见本说明书的记号部分中对该关系和本说明书中使用的其他记号的说明）。

如上所述，动物园管理机构已经设计成负责对 ZAF 进行总体控制。相应地，动物园管理机构负责建立动物园工作人员机构的运行时间表。注意框架设计者还把动物园管理机构设计成 ZAF 的核心功能，这意味着它已经设计成不受潜在的用户化和扩展。在用于动物园管理机构的类别框中的大写字母 C，表示了这种事实。注意动物园管理机构与动物园工作人员机构之间的“使用”关系也设计成核心功能，从而使它不受框架用户的最终用户化。

动物园工作人员机构被适当地设计成大体上负责动物的照顾和喂养。因此，它采用了动物和圈养单元机构执行其任务。然而，与动物园管理机构的设计不同，框架设计者已经将动物园工作人员机构设计成一种可扩展功能，这再次意味着动物园工作人员机构已经被设计成可由框架用户修正和/或扩展的，以应付将来的照顾和喂养要求。这一事实由动物园工作人员机构类别框中的大写字母 E 表示。

框架设计者已经设计了动物机构，以代表动物园动物与动物园工作人员之间的相互作用中动物一方。由于动物园中的动物数目是有规律地变化

的，动物机构也被类似地设计成可扩展功能。圈养单元机构通过代表各个圈养单元（诸如围栏、水池、和笼子），而与动物园工作人员机构相互作用。象动物机构一样，圈养单元机构被设计成可扩展功能，以使其能够应付将来的用户化和扩展要求。然而，应该注意的是，即使动物园工作人员、动物园动物和圈养单元机构都被设计为可扩展功能，但这些机构之间的关系被设计成ZAF核心功能。换言之，即使希望在动物园工作人员、动物园动物和圈养单元机构方面给予ZAF用户灵活性，但不希望使ZAF用户改变这些机构的相互关系。

框架设计者随后设计构成图1的机构的类和关系。类是一组类似的对象的定义。因此，类可被认为是对象的抽象或一个类型的对象的定义。从计算机系统的观点看，单个的对象代表封装的数据组和由计算机系统对该数据进行的操作或操作组。事实上，在安全计算机系统中，对对象所控制的信息的唯一存取，是通过对象本身。这就是为什么包含在对象中的信息被称为被对象所封装。

各个类定义包括定义由对象控制的信息的数据定义，和定义对象对各个对象控制的数据进行的操作的操作定义。换言之，类定义通过定义对所定义的数据进行的操作或操作组，来定义对象如何作用和反作用。（应该注意的是操作也被称为方法，方法程序，和/或成员功能）。当联系在一起时，所定义的操作和数据被称为对象的行为。本质上，类定义限定了其成员对象或对象的行为。

图2是OO类图，它显示了框架设计者为ZAF设计的框架的基本类。每个类表示，显示了其与图1所示的机构的关系。例如，动物园工作人员类被表示为来自动物园工作人员机构。ZAF的基本类包括：动物园管理者类，它是动物园管理机构的一部分；动物园工作人员登记类，它也是动物园管理机构的一部分；动物登记类，它是动物园工作人员机构的一部分；动物园工作人员类，它也是动物园工作人员机构的一部分；圈养单元登记类，它也是动物园工作人员机构的一部分，动物类，它是动物机构的一部分；以及圈养单元类，它是圈养单元机构的一部分。应该注意的是，类之间的关系，被设计成ZAF的核心功能，从而使它们不受ZAF用户的最终

修正。

动物园管理类，是负责ZAF的总体控制的对象的定义。这里，OO类只定义进行交互作用以提供对问题的解决方案的对象。然而，通过揭示类定义的特性，使我们能够理解框架机构的对象如何被设计成提供能被用户化和/或扩展以应付将来要求的实际解决方案。

动物园管理类得到适当设计，以与动物园工作人员登记具有“使用”关系。框架设计者将动物园管理和动物园登记类设计成ZAF的核心功能，因为设计者决定不让ZAF的用户修正作为这些类定义成员的对象的行为。动物园工作人员登记—它与动物园工作人员类具有所谓的“访问包含关系”，只是定义一个对象的类，该对象包含了所有的动物园工作人员对象。因此，动物园工作人员登记包括用于list_zoo_keepers()操作的定义。如在后面所要描述的，该操作用于向请求该清单的其他对象提供动物园工作人员对象的清单。

图3显示了动物园管理者类的较低层面。由于动物园管理者类的对象负责ZAF的总体控制，动物园管理者类被设计成包括执行面向动物园管理的任务。该类定义包括以下五种操作：5_minute_timer()、add_animal()、add_containment_unit()、add_zoo_keeper()、以及start_zoo_admin()。

start_zoo_admin()操作用于启动ZAF。即，用户或系统管理者将与start_zoo_admin()操作相互作用，以开始通过ZAF对动物园进行管理。start_zoo_admin()操作用于启动5_minute_timer()操作，从而使5_minute_timer()每五分钟就命令动物园工作人员对象出来并检查动物园动物。add/delete_zoo_keeper()操作用于与ZAF的用户进行相互作用，以定义增加的动物园工作人员（即增加的动物园工作人员类），并加上增加的动物园工作人员（即动物园工作人员对象），以及除去动物园工作人员类和/或对象。如从以下描述可见，各个动物园工作人员对象用于执行具体的动物园任务。因此，ZAF的用户自然很希望加上一个动物园工作人员定义和对象，以处理附加的动物园任务或除去不再需要的对象定义。ZAF框架设计者已经通过将动物园工作人员机构设计成可扩展功能来提供这种灵活性。

象 add/delete_zoo_keeper()操作一样， add/delete_animal()操作用于与用户进行相互作用，以定义增加的动物园动物类和对象，并除去不再需要的类和对象。同样，动物园需要加上和除去动物也是很自然的。add/delete_containment_unit()操作用于定义新的圈养单元类和对象，并用于除去不再需要的类和/或对象。同样，框架设计者通过将动物和圈养单元机构设计成可扩展功能，而提供了这种灵活性。

参见图 2，动物园工作人员类定义同动物登记、动物、圈养单元登记、以及圈养单元类具有“使用”关系。由于通过使 ZAF 的用户能够用户化和扩展动物园工作人员、动物、和圈养单元类，从而增加了 ZAF 的价值，ZAF 的框架设计者已经将这些类设计为可扩展功能。然而，改变动物和圈养单元登记种类的行为，将破坏 ZAF 的基本操作。因此，框架设计者将这些种类设计成 ZAF 的核心功能。

图 4 是动物园工作人员类的类图。然而，在描述图 4 的细节之前，先应该指出的是，图 4 所示的类定义是按照所谓的类层次结构的简单顺序排列的。代表类层次结构中最一般/抽象类的类，象动物园工作人员类，被称为层次结构的基本类。在类层次结构中类的顺序，从最一般至最不一般（即从一般至特殊）。不那样一般的类（例如喂养者类）从更为一般的类（在此情况下即为动物园工作人员类）继承了特性。这样，喂养者、兽医、和温度控制者类定义，被称为动物园工作人员类的子类。继承机构将结合图 5 而得到更为详细的描述。

如图 4 所示，动物园工作人员类定义包含单个操作定义，即 check_animals()操作定义。读者还应该注意的是，动物园工作人员类定义被标为抽象类。抽象类没有对象作为它们的成员，而是被用来定义用于它们的子类的公共接口/协议。当一个类的至少一个操作定义是纯虚拟操作定义时，该类被称为抽象类。纯虚拟操作定义只是为了定义该操作的子类定义的公共接口而设计的。换言之，实际行为（即数据和操作）的设计，留给了子类自己。在动物园工作人员类定义的情况下，喂养者、兽医和温度控制者子类定义了纯虚拟 check_animals()操作定义的具体实施，该定义被包含在动物园工作人员类中。当一个操作被设定为等于 0 时，它被标为纯虚拟操

作。

但重要的是要注意，纯虚拟操作定义的公共接口必须被所有的子类所确认，从而要求对象（称为用户对象）能够在不需要知道服务器对象的具体子类的情况下使用子类成员对象（称为服务器对象）。例如，每当由动物园动物类定义的对象需要具体进行的行动时，它与动物园工作人员对象相互作用。由于至这些对象的接口以动物园工作人员抽象基本类定义并被保留在用于 check_animals()操作的子类定义中，动物园管理者对象不需要具体知道任何服务器对象的子类。这使得不需要在执行行动的过程中（即在动物园管理者对象的部分上）进行该行动。利用了抽象类特性的设计（诸如 ZAF 设计）被称为多形的。

多形对于 OO 框架设计是非常重要的，因为它允许某些事情进行（称为实施）的方式被改变或扩展，而不影响依赖于行动实际上完成的事实的机构。换言之，用户对象只需要理解某些对象执行某些功能，而不是这些功能实际上是如何执行的。这是适当设计的 OO 框架可被容易得到用户化和扩展以满足将来要求的一个方式。

如上所述，框架设计者适当设计 ZAF 框架，从而使动物园工作人员对象与动物和圈养单元对象相互作用，以执行它们各自的任务。图 5 是类图，显示了动物抽象类的类层次结构。由于动物类定义用于代表动物园动物的特性和行为，框架设计者以反映这种作用的方式设计动物抽象类。如所示，示例性的动物类定义包括数据定义 feed freq 、 location 、和 temp_range 以及操作定义 get_temp_range()、 feed()、 needs_ food()、 needs_vet_ visit()和 vet_visit()。

为了概述这种框架，不需要解释各个定义的细节。然而， temp_range 数据定义和 get_temp_range()及 feed()操作定义是良好的框架设计选择的例子。

feed()操作定义用于完成动物的实际喂养（即通过具体的喂养设备，未显示）。 feed()操作是纯虚拟操作。同样，这意味着类的设计是这样的，即执行所需功能的实际机构被留给子类来定义。在其中作为子类的成员而产生的对象具有特定的需要的情况下，要求子类定义是一种好的设计选择。

例如在 ZAF 框架中，各种动物可能需要特定的喂养设备，这不仅使类属 feed()操作变得困难，而且无价值。

通过比较，框架设计者明确地设计了 get_temp_range()操作，从而使它不是纯虚拟操作定义。这意味着 get_temp_range()被类属地定义为缺省操作。这样，它被认为是虚拟操作。缺省操作被用来向子类提供类属功能。子类可以简单地使用该缺省操作，或者它们可以通过再定义来用户化或扩展缺省操作。缺省操作的再定义被称为超越缺省操作。

哺乳动物是动物类的一个子类，且哺乳动物类因而继承了动物类的所有特性。哺乳动物类还被设计为抽象类，这再次意味着它未被设计成具有作为其成员而产生的对象，而是得到适当设计以为其子类提供公共接口。哺乳动物子类被进一步分成食肉动物类和食草动物类。

由于 feed()操作的定义被留给了子类，食肉动物和食草动物子类都具有它们自己的 feed()操作定义。同样，这是一个好的设计选择，因为食肉动物具有与食草动物不同的需要。

Temp_range 是与具体动物的自然习惯一致的温度范围的数据定义，且 get_temp_range()操作定义是用于获取具体动物的 temp_range 并将其送回到请求的用户对象。爬行动物子类包含其自己的 temp_range 数据定义和自己的 get_temp_range()操作定义。按此方式设计 ZAF，以指出数据定义能象操作定义一样被超越。由于很多爬行动物生活在沙漠条件下，在那里夜间非常冷而白天非常热，缺省的 temp_range 定义在爬行动物类中被超越，以包括时间和温度信息（在图 5 中没有明确显示）。这是另一个好的设计选择，因为它使得 ZAF 能够通过允许根据一天中的时间和圈养单元本身当前的温度来调节温度，从而以不同于其他圈养单元的方式处理爬行动物圈养单元。

图 6 是类图，显示了圈养单元类的较低层面。该圈养单元类包含虚拟操作定义 adjust temp()。adjust temp()定义限定了实际用于调节（即，经过未显示的加热和冷却装置）动物园的圈养单元温度的接口和机构。

动物园对象如何相互作用

除了设计构成解决具体的编程问题的对象之外，框架设计者还必须设计各个对象是如何相互作用的。换言之，对象必须以利用其设计方式的方式进行相互作用。如上所述，为对象所定义操作在为对象所定义的数据上进行操作的方式，被称为对象的行为。虽然对象可以具有独立实体的特征，但各个对象在与其他对象发生联系时呈现一致的行为仍然是非常重要的。一致的行为是重要的，因为对象依赖其他对象的一致行为，以使它们自己能够呈现出一致的行为。实际上，一致的行为是如此地重要，以致对象的行为经常被称为对象与其他对象具有的契约。当对象不呈现出一致的行为时，称它已经侵犯了与其他对象的契约。

当一个对象的操作需要对第二个对象控制的数据进行存取时，就认为它是第二对象的用户。为了对第二对象控制的数据进行存取，用户的一个操作将调用或启动第二对象的一个操作，以获得对该对象控制的数据的存取。被调用的对象的一个操作（在此情况下即为服务器操作）随后得到执行，以存取和/或操纵被所调用的对象控制的数据。

图 7 是对象图，显示了 ZAF 的示例性对象是如何进行作用以帮助动物园工作人员管理动物园的。为此概述的目的，不需要对所有 ZAF 对象的作用进行详细分析。然而，读者应该看看以下的简单控制流程，以对 OO 环境中的对象如何相互作用来解决问题有初步的理解。

如上所述，对象是作为具体类的成员而产生的。因此，对象 Zelda，即动物园管理者 706，是一个对象，它是动物园管理者的一个成员（实际上是唯一的成员）。因此，对象 Zelda 负责 ZAF 的总体控制。所有的动物园工作人员对象都由动物园工作人员登记对象（对象 700）进行了登记。因此，对象 Zelda 通过调用动物园管理者登记器对象的 list_zoo_keepers() 操作（步骤 1）而获得了现行动物园工作人员的清单。该动物园管理者登记器对象 700 是作为动物园工作人员登记类的一个成员而产生的。为了说明，假定这作为 Zelda 的 5_minute_timer()操作的一部分而每五分钟发生一次。动物园管理者登记器对象随后以动物园工作人员清单进行响应（步骤 2）。动物园工作人员清单包括温度检查员 Tina（对象 714），兽医 Vince（对象 740），以及动物喂养员 Fred（对象 752）。各个动物园工作人员

都作为动物园工作人员类的成员而产生。特别地，对象温度检查员 Tina 、兽医 Vince 、和动物喂养员 Fred 分别是温度控制者、兽医和喂养者子类的成员。

一旦当前动物园工作人员的清单被送回到对象 Zelda 706，对象 Zelda 通过调用各个动物园工作人员对象的 check_animals()操作，来命令清单中的各个动物园工作人员检查动物。步骤 3 只显示了对温度检查员 Tina 的调用。应该注意的是，对象 Zelda 不需要明白动物园工作人员清单中的动物园工作人员的类型、清单中动物园工作人员对象的数量、或任何一个动物园工作人员对象的具体特性。对象 Zelda 利用同一接口（即 check_animals()操作）与各个动物园工作人员对象进行通信。然后就由各个动物园工作人员对象来执行各自的任务，这些对象正是为完成这些任务而创立的。各个动物园工作人员对象，通过使用其自己的 check_animals()操作，来执行其分配的任务。例如，对象 Tina 的 check_animals()操作，通过调用 List_animals()操作（步骤 4），从动物登记对象获取当前动物的清单，并随后通过调用 List_cont_units()操作从圈养单元登记对象获取圈养单元清单（步骤 6）。通过检验动物清单，对象 Tina 的 check_animals()操作确定在动物园中当前只登记了两个动物，蛇 Sam （对象 728 ）和狮子 Simba （对象 718 ）。

对象 Tina 的 check_animals()操作随后调用 get_temp_ range()操作，以从对象 Sam 和 Simba 得到温度范围。一旦温度范围已经被送回，对象 Tina 的 check animals()操作确定各个动物（即 Simba 和 Sam ）所在的圈养单元，并随后调用适当的圈养单元（即在对象 Simba 的情况下为狮笼 7，而在对象 Sam 的情况下为蛇坑 3 ）的 adjust_temp()操作，以调节圈养单元的温度（步骤 12 和 13 ）。

各个圈养单元的 adjust_temp()操作随后通过以适合于各个圈养单元中的动物的方式调节温度，而完成控制流程。（即，对于蛇坑 3 根据时间和温度，对于狮笼 7 只根据时间来调节温度）。读者应该注意的是， check_animals()操作与 adjust_temp()操作之间的关系是多形的。换言之，对象 Tina 714 的 check_animals()操作不要求具体知道各个 adjust_temp()操作

是如何执行其任务的。 check_animals()操作只需要留在接口并调用adjust_ temp()操作。在此之后，就由各个adjust_temp()操作来以适当的方式执行它们的任务。

在此处，需要再次指出的是，ZAF系统是一个非常简单的框架，它只是用于使不熟悉的读者理解框架的某些基本概念，以更好地理解本发明的好处和优点。从以下的详细描述，这些好处和优点将变得显而易见。

详细描述

图8是根据本发明构成的计算机系统30的框图。该计算机系统包括中央处理单元（CPU）32，它响应操作指令而运行，而这些指令是从操作/显示接口34接收的，且计算机系统通过系统总线36与该接口相联。该CPU还通过系统总线与一个主存储器38相联，而主存储器38由各种数据结构显示，包括应用程序40、对象42、数据44、和操作系统46。主存储器38被显示为单个的实体，但本领域的技术人员会理解，该主存储器可以包括随机存取存储器（RAM）、硬盘驱动器、光盘驱动器和包含逻辑分段存储单元的其他存储装置的组合。

操作系统46最好支持面向对象的程序设计环境，例如由C++程序设计语言提供的环境。应用程序40由用户通过操作/显示接口34激活或调用。该应用程序可以用包括C++在内的各种语言写成。对象42是面向对象的程序设计语言（诸如C++）的对象数据结构。

计算机系统30还包括直接存取存储装置（DASD）接口48，它与系统总线36相联，还与DASD 50相联。本领域的技术人员会理解，DASD 50能够接收并读取包含于机器可读存储装置52中的程序产品，诸如其上记录有程序指令的磁介质盘，而这些指令的执行将实施本发明的框架。存储装置52还可包括诸如光盘介质和其他机器可读存储装置。计算机系统30还可包括网络接口54，它使得CPU 32能够通过网络58与其他计算机系统56之间进行通信。其他的计算机系统56可包括例如在构造上与示例性的计算机系统30类似的计算机系统。以此方式，计算机系统30能够在已经用众所周知的方法建立起计算机系统之间的通信之后，通过网络58将数据接

收到主存储器38中，而这些众所周知的方法是本领域的技术人员能够理解的，不需要进一步的说明。网络接口54是这样的装置，即框架用户借助它接收来自其他网络用户的消息并将消息传送给其他网络用户。

消息处理序列图

在描述面向对象的框架机构之前，通过考虑图8的计算机系统进行的消息处理步骤序列，将能够更好理解本发明的最佳实施例。图9是处理图，显示了当消息由图8所示的系统处理时消息所受到的处理步骤。图8的系统启动框架机构，以从消息队列获取消息。该系统为接收者清单中的每一条目处理消息。因此，系统或者将消息提供给本地分送机构，或把消息送到远程地址，或者指明消息无效或由于其他原因而不能被分送。在开始时，消息被接收到输入队列中，如图9中的第一个圈圈所示，表明“排队”状态。当从输入队列获取到消息时，可以说进入了“处理”阶段，如图9中的第二圈圈所示。最后，当所有的处理都完成时，可以说消息进入了“已经处理”状态，如图9的第三个圈圈所示。通常，处理过的消息被简单地从处理系统的输入队列中删除。

图10代表了操作步骤的更为详细的显示，这些步骤包括了消息的“处理”阶段。这些操作步骤用四个圈圈表示。图10显示，在处理期间，消息首先受到寻址处理，随后是由预分送步骤表征的步骤。消息至下一个目的地的实际传送，由分送步骤表示。最后，在消息完成了分送处理步骤之后，处理阶段以管理处理步骤结束。

对“处理”阶段中的处理步骤的这种简化表示，在图11中得到了更为详细的显示；图11代表在消息处理阶段中进行的处理步骤序列。在图11中，从消息队列中获取消息被用“获得消息”事件表示，该事件使消息经历清单扩展（List Exp.）处理。在清单扩展期间，系统重复地把接收者清单条目扩展成目的地地址。该扩展可以是一对一的—如其中接收者清单条目代表了单个的接收者，也可以是一对多个的—如其中接收者清单条目代表了多个接收者的分布清单。以此方式，接收者清单得到扩展，从而识别一或多个电子邮件目的地地址。例如，一个电子邮件地址可以标明传统的

互联网络地址，该地址包括域名和子领名。在最佳实施例中，电子邮件目的地地址定义了接收者的消息协议类型。只要有要处理的接收者清单条目，系统就继续扩展它们。在消息受到清单扩展时，消息的内容没有改变。

图 11 显示，当接收者清单已经被完全处理成目的地地址清单且没有等待识别的接收者时，消息受到地址分解（Address Resolution）处理。在地址分解中，系统将各个目的地地址分解成分送的方法，诸如 SNADS 协议、SMTP 协议等等。图 11 中的显示为“分布清单改变”的、从地址分解处理圈圈向回指向清单扩展处理圈圈的箭头，表示地址分解处理可以造成接收者清单的改变。例如，可以在这样的情况下出现，即在清单扩展处理中识别出的目的地地址在地址分解期间被认为需要一个消息路由，而该消息路由要求的目的地地址不同于清单扩展所分配的地址。在此情况下，重复清单扩展处理。例如，如果必须将消息送到并非接收者与之相联的网络网关时，就会出现这种重新分配路由的情况。

如果地址分解不产生改变分配清单事件，则它产生得到适当寻址的消息。这意味着消息已经被分配了目的地地址、消息协议类型和状态。如图 11 所示，这意味着消息已经得到适当寻址且消息随后在信封处理阶段得到处理。

在最佳实施例的电子邮件处理系统中，表示为文字串的消息内容由系统“信封”所包裹，该信封表明协议、发出者、接收者等等。在信封处理期间，产生了适当的系统信封并使它与消息联系起来。图 11 的信封处理圈圈对应于图 10 所示的“预分送”阶段的开始。当包封处理完成时，进行附件转换。在最佳实施例的图 8 的系统中，消息的内容被称为附件。因此，消息被处理成信封和相联系的附件。一个附件可以包含文本文字、语音、图形、或视频数据，或任何其他能够以电子表示存储并在计算机网络上传送的信息。

在附件转换中，消息的文字串根据与各个接收者相联系的电子邮件系统而得到处理。例如在一个电子邮件系统上的接收者可能要求文本用 ASCII 显示字符表示，而在另一系统上的接收者可能要求文本用 EBCDIC（扩展二进编码十进制互换码）字符表示。附件转换包括任何所需的文本

转换和用于目的地系统的其他的格式处理。框架用户可以指定用于不同协议的这种消息处理，而不修正框架的对象类，以此方式能够在初始系统组成之后方便地容纳所加的协议。

在附件转换之后，图 11 显示出消息处理移到安全和权限阶段。在此阶段，对消息进行保密检查。这种处理能够根据这些保密检查而改变分送状态，从而影响消息的文本。例如，根据保密检查，消息文本可能受到限制、掩码或编码。安全和权限完成了图 10 所示的消息处理的“预分送”阶段。下一个阶段是“分送”阶段。

为了分送消息，系统首先进行本地分送检查。即，如果消息具有“本地”状态，则假定消息的接收者位于处理该消息的当前系统上。因此，框架系统将消息送到一个本地消息分送处理器，以完成消息 处理。完成至各个目的地的消息分送所需的具体处理，不是框架系统进行的处理的一部分。

如果消息没有“本地”状态，则认为不能完成本地分送，且如图 11 所示，消息处理移到消息转送阶段。在消息转送阶段，检查各个消息接收者的状态。如果与接收者相联系的状态是“远程”状态，则消息被从本地网络移出且该消息被转送到另一网络供进一步处理。以此方式，框架实施的系统起着网络网关处理器的作用。

图 11 中所示的下一个处理状态，是未分送处理。在未分送处理中，检查各个接收者的消息状态。如果该状态指出一个未分送事件，则任何具体的处理都完成。例如，如果消息不能被分送到具体的接收者，则一个未分送表示可以被立即送回到发出者。其他的事件，诸如通信故障，如果需要的话，也能够得到显示。同样，框架用户能够方便地指定这种处理并能够在框架的规定之内方便地改变这种处理而不需要大的修正。

在未分送处理完成之后，图 11 所示的消息处理进行到附件管理阶段。该处理对应于图 10 中所示的“管理”阶段的开始。附件管理处理只在消息具有附件时进行。图 11 显示消息随后经历了会计处理。在这种会计处理中，消息的文本不改变。会计收费、处理费用等等能够得到解决，且费用被计帐或者产生帐单。另外，系统操作统计能够得到累积或更新。当会计状态

完成时，消息处理完成，且消息通常被从系统中删除。于是，框架机构操作完成，其他的系统处理能够得到恢复。

本发明提供了一种面向对象的框架。利用该框架发展的邮递服务器系统的操作，可以借助图 9 、 10 和 11 的状态图来理解。本领域的技术人员则理解框架对象和它们的关系，且它们的处理也以面向对象的程序设计表示得到了完全而准确的描述。因此，下面将根据上述的邮递服务器框架机构类，采用与上述关于动物园工作人员例子的图 1 至 7 描述的图件相类似的类图和对象图，描述最佳实施例的框架。

类别/类图

图 12 是图 8 的计算机系统中实施的框架的类别图。本领域的技术人员会理解的是，图 12 中所示的类别代表包含数据属性和行为并储存在图 8 的框图所示主存储器中的面向对象的程序设计（ OOP ）对象的集合。这种对象可在例如支持 C++程序设计语言的计算机系统操作环境中实施。

图 12 的类别图的框架表示，显示了框架的初级类类别成份或机构。所有机构都以它们相应类类别框中的 E 标出，以表示它们都是可扩展的类类别，意味着它们的对象类和属性能够得到扩展从而被框架用户用户化。被称为消息中心的类类别包含框架所要产生和处理的消息。图 12 显示，消息中心类类别与称为消息的类类别具有“使用”关系， 消息包含对象清单形式的电子邮件消息信息，而该对象存储的信息允许扩展框架以处理消息中心的电子邮件消息。

从消息中心类别框至消息类别框的连接线所表示的“使用”关系，被标为 C， 以表示它是框架用户所不能改变的核心关系。 该“使用”关系代表了这样的联系，即其中消息中心机构处理或使用来自消息的消息。图 12 显示，消息类类别又与称为原始接收者清单 、 接收者清单 、 信封清单 、 附件访问清单 、 报告内容清单 、 报告对象清单、发出者清单、和回答对象清单 的类类别具有“使用”关系。这些类类别将在下面详细描述。这些关系（连线）被标为 C，以表示它们不能由框架用户改变。即， 虽然类属性和行为是可由用户扩展（改变）的，但类之间的关系结构

是不能改变的。

图 13 是类图，它表示了消息中心类类别的对象。在消息中心类别中的对象和这些对象之间的关系，被认为是框架用户所不能修正的“核心”特性。消息中心 中的对象属于消息中心 类别，并用于启动初始产生所有消息成员对象的方法（也称为操作）。在最佳实施例中，以 C++ 程序设计语言实施，消息中心对象通过调用 C++ 对象产生器方法， 而产生消息对象。框架实施的这种产生方法将依赖于实施框架的具体程序设计语言。

每一个消息中心对象包括一个 handleIncomingMessage()方法，它产生所要处理的消息对象。 这种方法以一个数据串作为其输入，该数据串与相应的产生者清单中的对象所标明的产生方法一起使用，以产生消息对象的一个事例。消息中心对象的行为包括图 13 所示的 processMessage()方法。正是这种方法接收了所要处理的电子邮件消息的内容，将其包含在消息 对象中。即，该方法将消息的内容分析成框架设计所期望的对象。

该框架没有完全定义在消息内容上进行的处理。处理活动的定义是用户进行的框架扩展的一部分，该扩展完成了作为框架定义的处理电子邮件步骤组的一部分而被调用的方法。当框架的产生方法产生一个消息中心对象时，它还产生了称为消息队列的类和所示的三个产生器清单对象， 包括信封产生器清单 、 附件访问产生器清单和地址产生器清单。通常，只有一个单个的消息队列对象，虽然可以有多个消息中心 对象。processMessage() 方法连续地从消息队列 中除去下一个消息对象，并调用该消息对象方法。

图 13 显示出，消息 中心类别的消息中心类与消息队列类之间的关系是“具有”关系， 这意味着消息中心类中包含有消息队列对象。消息队列对象以先进先出（ FIFO ）的方式得到维护。

消息中心利用消息队列类中的对象，对消息对象进行跟踪。图 13 显示出，消息队列类包括称为 add( ) 和 remove()的方法。 add()方法由消息中心用于将对一个消息对象的引用加到(也称为登记)消息队列上，而 remove()方法用于从消息队列除去一个消息对象索引。 该对象索引可包括例如一个消息指针或消息指标。这种加和除去操作是用于程序设计框架的具体的面

向对象的程序设计语言（诸如C++程序设计语言）的特征。

在产生了消息对象之后，消息中心利用add() 方法将一个消息访问加到消息队列上，随后用remove( ) 方法除去FIFO队列中的下一个访问。在消息对象访问被从队列中除去之后，消息中心对象通过依次调用导致电子邮件处理的消息类方法，处理访问的消息对象。这种电子邮件处理方法由框架用户定义（如下面将进一步描述的）并在框架扩展时由用户通过继承而产生或实施。

图13还显示出，被称为消息队列 的类与被称为消息的类的具有关系，从而使消息队列“具有”多个消息 对象。即，消息队列包括消息对象的一个集合。一个消息队列对象被消息中心用作特定的容器对象， 以保持它已经产生但还未处理的各消息对象的关系。

图14是消息类的类图。在该消息类中的对象和这些对象之间的关系，被认为是不能由框架用户修正的“核心”特性。消息对象具有关于一段电子邮件的、处理它所需的所有信息。 图14的类图显示了与称为发出者清单 、 报告对象清单 、 原始接收者清单、附件访问清单、信封清单 、 报告内容清单、接收者清单和回答对象清单的类有“具有”关系的消息类。这些类中的每一个，都包含清单对象。 将这种信息保持在核心类中，有利于对这种存取进行组织和控制。以此方式，消息对象具有关于一段电子邮件的、 在框架内处理其所需的所有信息。每个清单对象包含被保存在核心对象中的信息，从而使信息能够被框架的扩展部分所存取。除了接收者清单 对象之外，对于每一个消息对象，最多有一个唯一的清单对象。

图14表明，消息类中的对象定义了若干方法。 在最佳实施例中，这些方法包括（按照字母顺序） accounting( ) 、 attachmentProcessing()、deliverLocal()、deliverRemote()、expandList()、handleNonDelivery()、manageAttachments() 、 processAuthorization()。这些方法每一个都包括系统的处理步骤。执行这些方法的顺序和与它们有关的处理行为，将在下面结合框架的对象图进行更为详细的描述。选择这些方法来提供系统所希望的灵活性，并提供增强的系统操作。

消息对象从图14 的发出者清单 、 报告对象清单 、 原始接收者清

单、 附件清单 、 信封清单 、 报告内容清单、接收者清单和回答对象清单类获取对象。在消息类中的对象能够利用下面描述的适当操作， 从这些类获取对象。下面将更详细地描述所用的这些类和操作。

图 15 是发出者清单的类图。即，图 15 显示了构成框架的发出者清单类的对象。所示的类与称为地址的类具有“具有”关系。连线上的 n 表示多重地址 类。在地址类中的对象与发出者清单类的关系， 对于框架来说被认为是“核心”或不可修正的。可以有任何数目的发出者清单对象，但每一个都由唯一的消息 对象使用或访问。发出者清单对象包含特定电子邮件消息的发出地址。框架不定义电子邮件地址格式或它们的内容。这种定义是框架用户进行的框架扩展的一部分。发出者清单对象定义两种方法：当把一个新地址加到发出者清单对象上时使用的 addEntry()，以及当存取消息对象的发出者清单 条目中的一个条目时使用的 retrieveEntry()。

在图 15 中，地址类中的类装饰， 表明了它是抽象对象类型的。因此，这个对象必须作为扩展框架的一部分而被子类，以支持特定的电子邮件地址形式，作为其用户化的一部分。框架的实施者将需要以此方式来扩展地址对象。 框架允许任何数目的子类地址对象存在在同一框架中。认为各个扩展的地址类定义唯一的识别对象类型。对于什么类的地址 对象可以处于给定消息的发出者清单中，没有发出者清单对象要求。

在图 15 中显示了框架实施者可能进行的两种框架扩展。 一个地址子类或对象扩展包含电子邮件地址（在该图中被显示为 SMTP 地址）的互联网络（ Internet ）标准 SMTP（简单消息传送协议）形式。 另一地址子类保持电子邮件地址（在图中被显示为 SNADS 地址 ）的 IBM 公司 SNADS/OfficeVision （ Systems Network Architecture Distribution Services ）形式。面向对象的技术允许这些扩展的对象类进一步定义它们自己的具体方法或属性。这些具体子类不是框架定义的一部分，但可以是它们所支持的一部分，如框架实施者所要求的那样。

图 16 是类图，显示了构成框架的信封清单类的对象。该类与称为信封的类具有“具有”关系。信封类中的对象与信封清单类的关系，被认为是“核心”，这意味着在框架方面它是不可修正的。可以有任何数目的信封

清单对象，但每一个都由单个的消息 对象事例所使用或访问。一个信封清单对象包含特定电子邮件消息的信封对象清单。一个信封对象由框架定义，以保持有关电子邮件消息的信息，诸如优先权、标题、主题等等。该信息也称为电子邮件“标头”信息，因为它向消息服务描述了电子邮件消息的属性，但不是消息的内容。因此，有一个与电子邮件消息相联系的信封对象，且信封对象有助于确定当电子邮件消息被框架系统传送到消息接收者时它将受到的处理。

本发明的框架不定义信封的格式或它们的内容。其设计允许很多信封对象，它们是单个的电子邮件消息的信封清单的一部分。每一个都可以被产生，用以保持不同的信息或关于一段电子邮件的内容相同但格式不同的信息。用具体例子说明的信封清单对象的实际内容， 可作为消息 对象处理的一部分而被加上或改变。 信封清单 对象定义称为 add( )（它在将新的信封加到信封清单上时使用）和 retrieve()（它在存取消息对象的信封清单中的条目时使用）的方法。

称为信封的类在图 16 中被显示为抽象基本类。框架的设计假定，该类中的对象被再分类，作为扩展框架以支持特定的电子邮件信封格式（或多个可能的格式），作为其用户化的一部分。框架的实施者必须以此方式扩展信封对象。框架允许任何数目的子类信封对象存在于同一框架中。假定各个扩展的信封类定义了唯一的识别对象类型。没有如什么类的信封对象可能处于给定消息的信封清单中这样的信封清单对象要求。

图 16 显示了框架实施者可能会作出的三种框架扩展。 一个信封对象扩展设计成保持电子邮件消息属性的 SMTP 形式（在图中被示为 SMTP 信封）。另一个扩展设计成保持电子邮件消息属性的 IBM 公司 SNADS/OfficeVision 形式（在图中被显示为 SNADS 信封 ）。另一个扩展保持电子邮件消息属性的“通用”或非协议指定形式（在图中被显示为通用信封）。 这些扩展对象能够进一步定义具体的方法或属性，这些是框架实施者确定的其所要求的支持的一部分。信封对象定义了两个方法：当改变信封时使用的 change()，和用于获取信封的协议识别类型的 type()。

图 17 是类图，显示了构成框架的接收者清单类的对象。接收者对象与

接收者清单 类的关系被认为是“核心”或就框架方面不可修正的。框架定义了一个接收者对象，以保持如电子邮件地址所指定的关于电子邮件消息的目的地的信息。框架不定义电子邮件地址形式或它们的内容。该定义是框架实施者进行的框架扩展的一部分。 框架允许增加、改变或更换接收者，作为消息处理的一部分。

接收者清单类被设计成支持框架的关键功能：“分解”电子邮件地址。分解电子邮件地址意味着确定把电子邮件消息传送到其接收者所需的一组特定的电子邮件服务和支持。这些服务和支持是用对象类方法实施的。分解电子邮件地址，使框架能够具有灵活性，以支持多个类对象扩展，这些扩展的每一个都支持处理电子邮件消息的不同的要求。框架可以得到扩展，以支持子类方法中的多个特定的电子邮件功能组。这些组中的任何一个，或者也许它们全部，都可以作为单个的消息对象的处理的一部分而得到调用。 在分解电子邮件地址时，消息的接收者清单得到处理，以产生由它们的框架类扩展定义的单独具体的对象。例如，包括 SNADS 地址的电子邮件消息接收者清单，被分解成 SNADS 接收者清单对象，该对象因而定义了一个 SNADS 协议接收者。

当产生消息对象时， 假定一段电子邮件的所有地址都处于未分解状态。作为处理每一个消息 对象的框架的一部分而调用的方法之一，是 resolveAddresses()方法，它是针对消息对象而调用的。 resolveAddresses()方法又使用了为接收者清单对象类定义的 resolveAddresses( )。这种方法对未分解的接收者清单对象进行操作， 该对象属于接收者清单对象类的子类，并被预定义为框架的核心部分，如在其类云中的 C 所示。 框架从未分解的接收者清单中除去地址条目对象，并将它们加到已经作为框架实施者的扩展的一部分而定义的另一接收者清单类对象上。以此方式， 框架实质上把未分解的地址“引导”（或分解）到适当的接收者清单对象类中—该类具有定义的、用于处理具有该类接收者地址的方法。

当各个具体的地址对象类型的地址对象存在于消息的未分解的接收者清单对象中时，该类型必须受到 resolveAddresses()方法的支持。 resolveAddresses()方法将来自未分解清单的地址对象重新划分成适当接收

者类型的清单。因此，resolveAddresses()方法可以说是有助于地址借助该方法的功能来“分解自己”，并被从未分解的接收者清单中除去并被加到附在同一个消息事例上的另一个接收者清单对象中。图 17 所示的接收者清单类是一个抽象基本类。框架设计假定在此类中的对象作为框架扩展的一部分而得到再划分，以支持具体的电子邮件附件接收者清单作为框架用户的框架用户化的一部分。框架的用户以此方式扩展接收者清单类。

图 17 中显示了框架用户可能进行的两种示例性框架扩展。一个接收者清单对象扩展被设计成定义接收者的接收者清单对象类，它被标明为被分解成电子邮件方法的互联网络标准 SMTP 形式（在图中被显示为 SMTP 接收者清单）。另一接收者清单对象扩展被设计成定义接收者的接收者清单对象类，它被分解成电子邮件的 IBM 公司/OfficeVision 形式（在图中被显示为 SNADS 接收者清单）。图 17 显示了扩展的框架，从而使框架用户能够支持需要这两种处理的消息。由于单个的消息可以具有多个接收者，所以在扩展的框架处理每个消息时，地址可以被分解成这两个接收者清单中的一个或两个接收者清单。这些扩展的对象可以进一步定义具体的方法或属性，它们被实施者视为它们所需的支持的一部分。

图 17 显示出，接收者清单对象定义了以下方法：当把新的地址条目对象加到接收者清单上时使用的 addEntry() 方法，和 expandAddresses()方法—它在当把接收者清单 地址条目对象由代表一系列电子邮件地址的单个电子邮件地址（如在命名分布清单中那样）扩展成多个地址条目对象时使用。另一方法是 resolveAddresses()，它在当处理未分解的接收者清单对象的地址条目 对象时使用。当接收者清单对象存取消息对象的接收者清单类型对象条目中的地址条目对象时，使用一个 retrieveEntry()方法。当从消息对象的接收者清单中除去地址条目对象时，使用一个 removeEntry()方法。这是需要的，以实施“分解”功能，因为这意味着作为其分解的一部分从未分解的接收者清单除去了一个地址条目对象。最后，当取代消息对象的接收者清单中的一个地址条目对象时，使用了一个 replaceEntry()方法。这是需要的，以实施“取代”功能，因为这意味着从任何接收者清单中除去了一个地址条目对象并用另一地址条目 对象取代。应该注意的

是，调用 removeEntry() 方法并随后进行 addEntry()方法，可以代替 replaceEntry()方法，但与这两个方法的结合相比，在 rplaceEntry()方法如何工作上具体有所不同，且具有单独的方法使这些不同能够存在于框架内。

接收者清单类中的另一个对象是地址条目对象。 框架要求任何再划分的接收者清单对象必须访问地址条目 对象。地址条目对象访问称为接收者数据的类的具体框架对象以及接收者的地址对象。接收者数据对象为电子邮件地址特定的属性提供了一个开放区，这些属性诸如其与接收者清单中的地址对象的每一事例有关的状态（即本地、 远程或未分送）。地址对象的类与上述的发出者清单对象部分中描述的相同。

图 18 显示类图，显示了框架的附件访问清单的固有结构。附件访问对象与附件访问清单类的关系，被认为是“核心”的，或就框架来说是不可修正的。框架不定义电子邮件附件访问格式或它们的内容。框架的设计假定（但不要求）附件通常是对可能是消息内容的存储单元的访问。即，消息本体被作为附件。框架设计允许任何数目的附件处于消息对象的附件访问清单中，并允许附件被增加或改变，以作为消息处理的一部分。可以有任何数目的附件访问清单对象， 但每一个都被唯一的消息对象所使用或访问。 框架不定义也不限制一个电子邮件消息上的附件访问对象对作为另一消息的同一物理附件访问进行访问。一个附件清单包含消息的附件清单。由框架定义一个附件，用以保持关于电子邮件消息的内容信息，它由作为附件清单条目的一部分的访问确定。这也称为电子邮件“本体”或“内容”信息。

框架也不定义附件格式或它们的内容。其设计允许很多附件对象，它们是单个的电子邮件消息的附件访问清单的一部分。 各个附件对象可以被产生，以保持关于一段电子邮件的不同的信息或相同的信息，但是具有各种的格式。一个附件可以被加上或改变，作为消息处理的一部分。完全没有消息对象必须使用任何附件访问清单对象的限制， 象在其中电子邮件消息没有附件的情况下那样（象在分送报告中）。

如果框架实施者决定把电子邮件消息的消息内容或本体作为附件访

问对象处理，则具有消息本体的电子邮件消息的每一消息对象都将包括一个附件访问对象。或者，如果框架实施者决定仅为消息附件—诸如附在“回答”消息上的文本—使用附件访问对象，则只在电子邮件消息包括消息附件时消息对象才包括附件访问 对象。在任何情况下，由于附件访问清单 类是抽象基本类且相关的对象是核心对象，框架实施者必须定义至少一个这样的附件访问 对象和相关的处理方法。

附件访问清单类的对象定义了以下方法：当把新的附件加到附件访问清单上时使用的add()方法，以及当存取消息对象的附件访问清单条目中的条目时使用的retrieve()。附件访问对象是抽象对象类型的。框架设计假定附件访问对象必须被再划分，以此作为扩展框架的一部分以支持具体的电子邮件附件访问格式的（作为其用户化的一部分）。框架实施者必须以此方式扩展附件访问对象。框架允许任何数目的再划分的附件访问 对象存在。假定各个扩展的附件访问类定义了唯一的识别对象类型。至于在给定消息的附件访问清单中可能有什么种类的附件，没有对附件访问清单对象提出任何要求。

图18 中显示了框架可以进行的两种示例性框架扩展。一个附件访问对象扩展定义了对互联网络标准SMTP（或MIME）形式的电子邮件附件（在图中被显示为SMTP 附件访问 ）的访问。另一个对象扩展定义了对IBM 公司SNADS/OfficeVision 形式的电子邮件消息附件（在图中被显示为SNADS 附件访问）的访问。这些扩展对象可进一步定义具体的方法或属性，它们被框架用户视为其所需的支持的一部分。图18 显示，附件访问对象定义了称为change()（它在改变附件访问时使用） 和用于获取附件访问的识别类型的type()的方法。

图19 是类图，显示了构成框架的原始接收者清单的对象。原始接收者清单对象与原始接收者清单类的关系，被认为是“核心”的，或在框架方面是不可修正的。接收者对象由框架定义，以保持由电子邮件地址指定的电子邮件消息目的地的信息。当一个消息第一次被发出应用程序产生时，一个原始接收者清单对象包含该消息原始的接收者清单。当接收到电子邮件时，原始接收者清单条目允许接收者发现这一段电子邮件原来是由哪里

发出的。

这种处理不同于接收者清单处理之处在于，当邮递服务处理消息时，原始接收者清单类中的对象被假定为保持相同。当一段电子邮件的复本通过电子邮件网络从一个系统传送到另一系统时，接收者清单可被分成子组。假定原始接收者清单包含的接收者的原始清单不是被用于路由和转送邮件，而是报告邮件最先是从哪里被送到这一段电子邮件的所有接收者的。框架不定义电子邮件地址格式或它们的内容。这种定义必须是框架用户进行的框架扩展的一部分。

图 19 显示出，原始接收者清单对象定义了称为 addEntry()（它是在将新的原始接收者清单条目对象加到原始接收者清单类上时使用的）和 retrieveEntry()(它是在存取消息对象的原始接收者清单类中形成原始接收者清单条目时使用的）的方法。因此，原始接收者清单访问原始接收者清单条目对象，且原始接收者清单条目对象访问原始接收者的具体地址对象和分发类型对象。分发类型类中的对象使电子邮件消息的发出者能够识别所送到的原始地址是否被指定为“to：”、“cc：（复本）”或“bcc：”（盲复本）。框架假定消息处理将根据用于这些说明的规定进行，但不涉及这种处理包含的内容。即，这些条目被包括在消息中这一信息被传送，但不产生附加的、由框架系统进行的处理步骤。地址对象与在发出者清单对象部分中描述的相同。

图 20 是类图，显示了构成框架的回答对象清单类的对象。地址对象至回答对象清单类的关系被认为是“核心”关系，它是用户不可修正的。可以有任何数目的回答对象清单对象，但每一个都由唯一的消息对象所使用或访问。回答对象清单类包含电子邮件地址的清单，该地址应该被复制在接收者（电子邮件目的地）对消息的回答中。电子邮件消息回答被假定为只由“接收”应用程序产生。框架不定义电子邮件地址格式或它们的内容。这种定义必须是框架用户进行的框架扩展的一部分。

图 20 显示，回答对象清单对象定义了称为 addEntry()（它在将地址加到回答对象清单时使用）和 retrieveEntry() —它在存取消息对象的回答对象清单条目中的条目时使用—的方法。这些地址对象与上述发出者清单

对象部分中描述的相同。

图 21 是类图，显示了构成框架的回答对象清单类的对象。 地址对象与报告对象清单类的关系被认为是核心关系和不可修正的。可以有任何数目的报告对象清单对象，但每一个都由唯一的消息对象所使用或访问。报告对象清单类的对象包括由于电子邮件消息的分送、不分送或转送而需要向其报告的电子邮件地址的清单。报告被假定为是由于任何扩展的框架类方法或接收一段电子邮件的应用程序所产生的。框架不定义电子邮件地址格式或它们的内容。这种定义必须是框架用户进行的框架扩展的一部分。

图 21 显示，报告对象清单对象定义了称为 addEntry()（它是当把新的地址加到报告对象清单上时使用的）以及 retrieveEntry()（它是当存取消息对象的报告对象清单条目中的条目时使用的）的方法。地址对象与在上述发出者清单 对象部分描述的相同。

图 22 是类图，显示了组成框架的报告内容清单类的对象。在框架方面，地址对象与报告内容清单类的关系被认为是核心和不可修正的。可以有任何数目的报告内容清单对象，但每一个都由唯一的消息对象使用或访问。报告内容清单的对象包括在其上正在产生报告的电子邮件地址清单。在电子邮件消息是分送、不分送或转送的报告的情况下，消息表示为其产生这种报告的接收者。假定报告是用任何扩展的框架类方法产生的。框架不定义电子邮件地址格式或它们的内容。这种定义必须是框架用户进行的框架扩展的一部分。

图 22 显示，报告内容清单对象定义了称为 addEntry()（它在将新地址加到报告内容清单上时使用）以及 retrieveEntry() （它是当存取消息对象的报告内容清单条目中的条目时使用的）的方法。报告内容清单条目类中的对象访问报告内容清单条目对象。报告内容清单条目对象访问所报告的接收者的具体地址对象和报告指示器对象。报告指示器类中的对象包含关于所报告的内容的信息，例如消息是分送的报告还是错误，如果是错误则给出错误类型。 地址对象与在上述的发出者清单对象部分中描述的相同。

图 23 是类图， 显示了组成框架的地址产生器清单类的对象。每一个消息中心对象都有一个地址产生器清单 。 地址产生器类的对象与地址产

生器清单的对象的关系在框架方面被认为是核心和不可修正的。地址产生器清单被消息中心用于每一个地址对象的构成。注意地址被包含在很多其他的框架对象中。该框架不定义电子邮件地址格式或它们的内容。这种定义必须是框架用户进行的框架扩展的一部分。在此情况下，对框架的扩展是通过再划分地址产生器对象以产生新的、产生地址对象的具体类而进行的。

图 23 显示出，地址产生器清单对象定义了被称为 addEntry()（它在把新的地址产生器加到地址产生器清单上时使用）以及 removeEntry()（它在从地址产生器清单类中除去地址产生器对象时使用）的方法。地址产生器类的对象是抽象对象。框架设计假定，这种对象必须被再划分，作为扩展框架的一部分以支持作为其用户化的一部分的具体的电子邮件地址格式。框架用户将必须以此方式扩展地址产生器对象。在图 23 中显示了示例性的子类，其中地址产生器对象被再划分以产生 SMTP 地址和 SNADS 地址对象。图 23 显示，地址产生器对象定义了称为 createAddress()的方法，它被框架用于产生地址对象。

图 24 是类图，显示了组成框架的信封产生器清单类的对象。每一个消息中心对象都有一个信封产生器清单对象。信封产生器对象与信封产生器清单类的关系在框架方面被认为是核心和不可修正的。信封产生器清单被消息中心用于构成每一个信封对象。框架不定义电子邮件信封格式。这种定义是框架用户进行的框架扩展的一部分。在此情况下，通过再划分信封产生器对象来扩展框架，以形成产生信封对象的新的具体类。

图 24 显示出，信封产生器清单对象定义了称为 addEntry()（它在把新的信封产生器加到信封产生器清单上时使用）和 removeEntry()（它从信封产生器清单中除去信封产生器）的方法。信封产生器对象是抽象对象类型的。框架设计假定，这种对象作为框架用户用户化的一部分而得到再划分，这种用户化扩展了框架，以支持具体的电子邮件信封格式。即，框架用户必须以此方式扩展信封产生器对象。图 24 显示了三种示例性的再划分的信封产生器对象，它们产生称为 SMTP 信封、SNADS 信封和通用信封的类中的对象。在图 24 显示的示例性系统中，通用信封类方便地提供了一

般的说明类，说明了消息能够分成的类别。图 24 显示，信封产生器对象定义了称为 createEnvelope()的方法，它产生信封对象。

图 25 是类图，显示了组成框架的附件访问产生器清单类的对象。每一个消息中心对象有一个附件访问产生器清单。在框架方面，附件访问产生器对象与附件访问产生器清单类的关系被认为是核心和不可修正的。附件访问产生器清单被消息中心用于构成每一个附件访问对象。框架不定义电子邮件附件，也不定义附件是如何被消息访问的。这种定义是框架用户进行的框架扩展的一部分。在此情况下，对框架的扩展通过再划分附件访问产生器对象进行，以产生新的能造成附件访问对象的具体类。

图 25 显示，附件访问产生器清单对象定义了称为 addEntry()（它将新的附件访问产生器对象加到附件访问产生器清单上）和 removeEntry()（它从附件访问产生器清单除去附件访问产生器对象）的方法。图 25 显示出附件访问产生器为抽象对象类型的。框架设计假定，这种对象被再划分，作为框架用户的扩展的一部分，以支持具体的电子邮件附件访问格式。框架用户必须以此方式扩展附件访问产生器对象。

图 25 中显示了这种扩展的两个例子。再划分的附件访问产生器对象产生 SMTP 附件访问对象和 SNADS 附件访问对象。实际上可能不会需要一种以上的访问，因而文件名可被用于一种以上的电子邮件协议。附件访问在框架之内的抽象，使框架用户能够以此方式对其进行扩展。图 25 显示，附件访问产生器对象定义了称为 createAddressRef()的方法，它产生附件访问对象。

概要图

参见对象图，可以更好地理解由根据本发明构成的邮递服务器系统进行的操作步骤；本领域的技术人员明白，这些图显示了面向对象的程序设计框架实施的处理。对象图显示了对象类如何相互作用以产生框架功能。这些对象图随后假定了由框架用户对框架的某些扩展。该概要使用了上述类定义中使用的示例性类扩展。框架用户进行的实际扩展可以根据框架实施而变化。

图 26 是图 8 所示的计算机系统实施的框架的对象概要图。 图 26 是概要图，显示了消息中心对象（由消息中心对象云表示）的开始是如何响应以产生消息对象并开始对其进行处理，从而接收包括电子邮件消息的信息的，这由从消息中心对象云向回至其自身的、标为“1： handleIncoming Message()”的环线表示。

更具体地， handleIncomingMessage()方法得到调用，以处理进入的信息并产生消息对象。框架假定， 该方法传送一或多个参数—它用这些参数构成消息对象。 handleIncomingMessage()方法假定，这些参数中的一个是一个消息文本串。即，框架处理认为该消息是分解成组成部分的文本串，而这些组成部分被映象到在上述类图中描述的对象类上。消息文本串在图 26 中被表示为文本串。文本串的内容被用作至该概要的其他方法的输入。

应该注意的是，框架不规定用作输入的文本串的句法，但作为框架实施的一部分而加上的方法将句法规则加到输入的文本串上，因为它们将用它作为建立框架所处理的一段电子邮件的地址、信封和地址访问的输入。框架用户的一部分任务，是考虑电子邮件消息的来源和要由邮递服务器系统处理的电子邮件消息中所要接收的信息的形式（它可以在其他的对象中）。在此概要中所列的构成方法，是能够识别具有这些信息形式的信息的框架的扩展。

图 26 所示的下一个步骤，是对消息对象构成器的调用， 如从消息中心对象云至消息对象云的连线上的符号“2：消息（文本串）”所表示的。因此，消息中心对象产生消息对象。消息对象构成器以所述顺序调用该概要中的所有其他方法。然后，必须建立消息处理所需的各种对象。下一个步骤是调用 buildRecipientList()方法，以用消息文本串作为输入来产生消息对象的接收者清单对象，它由消息 对象云上的“3： buildRecipientList() ”表示。如上所述， 信封处理消息对象都具有接收者清单。

虽然消息对象可以具有不只一个接收者清单对象，但通常的情况是这样的，即在开始时所有的电子邮件接收者将被包含在一个未分解的接收者清单对象中。 接收者清单对象包含各地址条目对象。各个地址条目 对象由地址 对象和接收者数据对象组成。 createAddress()方法随后得到调

用，这由在从消息对象云至地址产生器清单对象云的连线上的符号“4：createAddress()”表示，以建立地址对象。createAddress()方法得到调用，直到不再有要加到消息的接收者清单 对象上的接收者地址对象。该方法将完成的接收者清单对象送回消息构成器。

从 buildOriginatorList（）、buildReplyList（）、buildReportToList（）、buildReportOnList（）、和 buildOriginalRecipientList()调用 createAddress()方法，因为所有这些对象都包含地址对象。从上述的相应类图可以看清楚这点。对 createAddress()的相应方法的调用没有在图中显示，但它们是该概要的一部分。createAddress() 方法处理将在下面得到更详细的描述。

随后的处理步骤是用文本串作为输入，调用 buildEnvelopeList()方法，以产生消息对象的信封清单对象。该步骤由消息对象云处的符号“5：buildEnvelopeList() ”表示。信封清单包含信封对象。createEnvelope() 方法得到调用，以建立各个信封对象，如从消息对象云至信封产生器清单对象云的连线上的符号“6： createEnvelope()”所表示的。该方法将在下面得到更详细的描述。createEnvelope( )方法得到调用，直到不再有要加到消息的信封清单对象上的信封。createEnvelope()方法将完成的信封清单对象送回到消息构成器。

图 26 的消息处理中的下一个步骤，是调用 buildOriginatorList()方法，以利用文本串作为输入产生消息对象的发出者清单对象，如消息对象云处的“7： buildOriginatorList()”所表示的。每一个消息 都具有发出者清单对象，它包含地址对象。作为该步骤的处理的一部分，createAddress()方法得到调用，以建立各个地址对象。createAddress( ) 方法得到调用，直到不再有地址要被加到消息的发出者清单对象上。该方法将完成的发出者清单对象送回到消息构造器。下一个步骤由消息对象云处的“8： buildAttachmentRefList( )”表示，它表示至 buildAttachmentRefList()方法的调用，以利用文本串作为输入，产生消息对象的附件访问清单对象。附件访问清单是消息对象的可选部分，且该方法可能不为给定的文本串产生一个，如果框架用户这样选择的话。附件访

问清单对象包含附件访问对象。下一个步骤是调用 createAddressRef()方法，以建立各个附件访问对象，如在从消息对象云至附件访问产生器清单对象云的连线上的符号“9： createAttachmentRef()”表示的。该方法的处理将在下面得到更详细的描述。createAttachmentRef()方法得到调用，直到不再有附件访问对象要加到消息的附件访问清单对象上。该方法将完成的附件访问清单对象送回到消息构成器。

消息处理的下一个步骤是调用 buildReplyToList()方法，以利用文本串作为输入，产生消息对象的回答对象清单对象。该处理由消息对象云处的符号 “10： buildReplyToList() ”表示。回答对象清单是消息对象的可选部分，且该方法可能不为给定的文本串产生对象。回答对象清单包含地址对象。作为该步骤的处理的一部分， createAddress( ) 方法得到调用，以建立各个地址对象。createAddress( )方法得到调用，直到不再有地址要加到消息的回答对象清单对象上。该方法将完成的回答对象清单对象送回到消息构成器。

图 26 随后的处理步骤由消息 对象云处的“11 ： buildReportTo List()”表示，它是一个方法调用，以利用文本串作为输入部分消息 对象的报告对象清单对象。该报告对象清单对象是消息对象的可选部分，且该方法对于给定的文本串可能不产生一个对象。报告对象清单包含地址目标。因此，该步骤的处理的一部分，是调用 createAddress() 方法，以建立各个地址对象。该 createAddress()方法得到调用，直到不再有地址被加到消息的报告对象清单对象上。该方法将完成的报告对象清单对象送回到消息构成器。

在产生了报告对象清单 对象之后， 下一个步骤由“12 ： buildReportOnList() ”表示，它是一个方法调用，用于利用文本串作为输入产生消息对象的报告内容清单对象。该报告内容清单是消息对象的可选部分，且该方法对于给定的文本串可能不产生一个对象。 报告内容清单包含报告内容清单条目对象，它由地址对象和报告指示器对象组成。该步骤的处理包括调用 createAddress()方法，以建立地址对象。 createAddress() 方法得到调用，直到不再有地址被加到消息的报告内容清单对象上。 该方

法将完成的报告内容清单对象送回到消息构成器。

下一个处理步骤由消息对象云处的“13：buildOriginalRecipientList()”表示，它是一方法调用，用于利用文本串作为输入，产生消息对象的原始接收者清单对象。原始接收者清单包含原始接收者清单条目对象，该对象由地址对象和分发类型对象组成。该步骤的处理包括调用 createAddress()方法，以建立地址对象。createAddress( )方法得到调用，直到不再有地址被加到消息的原始接收者清单条目对象上。该方法将完成的原始接收者清单对象送回到消息构成器。

现在已经构成了消息 对象。下一个步骤是将构成好的消息对象加到消息中心的消息队列对象上，在那里它将被保持到被 processMessage()方法除去。将消息 对象加上的步骤，在图 26 中由从消息对象云至消息队列对象云的连线上的“14： add()”表示。processMessage() 处理将在下面得到更详细的描述。最后，所示的最后的步骤，是用从消息队列对象云至队列对象云的连线上的符号“15： add()”表示的 add( ) 调用，将现在已经构成的消息对象加到消息队列的队列对象上。

图 27 显示了用于 buildRecipientList()的 createAddress() 方法处理，但它也类似地适用于 buildOriginatorList()、buildReplyToList()、buildReportToList()、buildReportOnList()、和 buildOriginal Recipient List()，因为所有这些方法都涉及地址对象。createAddress()方法处理以所完成的地址对象被送回到正在建立的清单对象（接收者清单对象、发出者清单 对象等等中的消息构成器结束。必须可以将每一个有效的消息（由文本串指定）分解成至少一个发出者清单条目和一个接收者清单条目。因此，有效的消息应该包括至少两个这种地址对象。

图 27 的处理从对 createAddress()方法的 buildRecipientList()方法调用开始，以建立与消息有关的所有接收者清单对象，由消息对象云处的“1：buildRecipientList()”表示。在此情况下，假定有至少两个地址产生器，一个是 SNADS 地址的，另一是 SMTP 地址的。下一个步骤由从消息对象至地址产生器清单云的“2： createAddress()”表示，它是以正在作为地址对象而构成的文本串 调用地址产生器清单的方法调用，其中所述地址对象

应被加到正在为消息建立的接收者清单上。

随后， createAddress( ) 方法必须调用它所包含的各个地址产生器对象， 直到它们中的一个送回了将被送回并加到接收者清单对象的地址对象。这由从地址产生器清单目标云至 SMTP 地址产生器对象云的连线上的符号“3： createAddress()”表示。下一个步骤是调用 SMTP 地址对象构造者—由从 SMTP 地址产生器云至 SMTP 地址对象云的连线上的符号“4： SMTP 地址（文本串）”表示，将用作输入的文本串传送给它，以产生 SMTP 地址。图 27 没有显示能由构造器使用的其他的信息来产生 SMTP 地址对象。

当 SMTP 地址产生器对象不识别文本串 因而不产生 SMTP 地址对象时，作为这种情况的一部分， createAddress()方法再次得到调用。 由于在此例中有地址产生器清单 也知道的 SNADS 地址产生器对象，该对象将被调用，以进行其自身的努力来识别文本串。该处理由至 SNADS 地址产生器对象云的、带有符号“5： createAddress()”和至 SNADS 地址对象云的、带有符号“6： SNADS 地址（文本串）”的连线表示， 并将用作输入的文本串传送给它，以产生 SNADS 地址对象。图 27 没有显示能由构造器用来产生 SNADS 地址对象的其他信息。 createEnvelope()方法得到调用，以建立各个信封对象。 createEnvelope()方法得到调用，直到不再有信封被加到消息的信封清单对象上。这种情况以完成的信封对象被送回到正在建立的信封清单对象内的消息构成器结束。图 28 显示了 createEnvelope()处理。

首先， buildEnvelopeList()方法将调用 createEnvelope()方法，以建立与消息有关的所有信封清单对象。在此概要中，假定有至少两个信封产生器。该步骤由消息 对象云处的环上的“1： buildEnvelopeList()”表示。随后以将要被构造为信封对象的文本串，对信封产生器清单调用 createEnvelope( )方法，而该信封对象将被加到正在为消息建立的信封清单上。这种处理由从消息云至信封产生器清单云的连线上的“2： create Envelope()”表示。

createEnvelope()方法必须调用它包含的各个信封产生器对象，直到它

们中的一个送回了将要送回并加到信封清单对象上的一个信封对象。这种处理由至 SMTP 信封产生器云的连线上带有符号“3 :createEnvelope()”的连线表示。然后， SMTPEnvelope()对象构造器得到调用，将被用作输入的文本串传送给它,以产生 SMTP 信封 。这由至 SMTP 信封 云的连线上的符号“4： SMTP 信封（文本串）”表示。图 28 的概要没有显示出会有其他的信息可以被构造器用来产生 SMTP 信封对象。

当 SMTP 信封产生器不识别文本串因而不产生 SMTP 信封对象时，作为该概要的一部分， createEnvelope()方法再次被调用。在此例中，由于还有信封产生器清单知道的 SNADS 信封产生器，所以它将被调用，以进行其自己的努力来识别文本串 ，如从信封产生器清单云至 SNADS 信封产生器云的连线上的符号“5： createEnvelope() ”所表示的。然后， SNADS 信封对象构造器方法得到调用，并将被用作输入以产生 SNADS 信封的文本串传送给它，如符号“6： SNADS 信封（文本串）”所表示的。图 28 的概要没有显示构造者可以使用其他的信息来产生 SNADS 信封对象。

作为消息处理的一部分，必须建立附件访问对象。为此，createAttachmentRef( )方法得到调用， 直到不再有附件访问对象被加到消息的附件访问清单对象上。该概要以完成的附件访问对象被送回到正在建立的附件访问清单对象内的消息构成器结束。

如图 29 所示， AttachmentRef()处理以对 buildAttachmentRefList()的调用为起点开始处理。 buildAttachmentRefList()方法随后调用 createAttachmentRef()方法，以建立与消息有关的所有附件访问清单对象。在此概要中，假定有至少两个附件访问产生器。这由消息类云处的环上的符号“1： buildAttachmentRefList()”表示。然后，对附件访问产生器清单对象调用 createAttachmentRef() 方法。该调用是对将被构成为附件访问对象的文本串 进行的， 而该附件访问 对象将被加到为消息建立的附件访问清单上。

createAttachmentRef( )方法必须调用它包含的每一个附件访问产生器对象，直到它们中的一个送回了被加到附件访问清单对象上的附件访问对象。图 29 显示了一个示例性概要，它具有两个附件访问产生器对象，其中

调用的第一个附件访问产生器是SMTP 地址，由从附件访问产生器清单 对象云至带有符号“ 3： SMTP 附件访问产生器对象云的SMTP 附件访问产生器目标云的连线表示。

随后 SMTPAttachmentRef()对象构成器得到调用，将用作输入以产生SMTP 附件访问的文本串传送给它。 该构成器调用由从 SMTP 附件访问产生器对象云至 SMTP 附件访问 对象云的连线上的符号“4： SMTP 附件访问（文本串）”表示。图 29 的概要没有显示会有其他的信息可由构成器来产生 SMTP 附件访问对象。

当 SMTPAttachmentRefCreator()方法没有识别文本串因而不产生SMTP 附件访问对象时，作为该概要的一部分， createAttachmentRef()方法再次被调用。由于在此例中 SNADS 附件访问产生器也为附件访问产生器清单对象所知道，它将得到调用，以进行其自己的努力来识别文本串。该处理由从地址产生器清单云至 SNADS 附件访问产生器对象云的连线上形成符号“5： createAttachmentRef()”表示。然后， SNADS 附件访问对象构成器被产生者对象所调用，将用作输入以产生 SNADS 附件访问的文本串传送给构可由构成器。 该概要没有显示会有其他的信息可由构成器来产生 SNADS 附件访问对象。这最后的对象产生处理步骤由从 SNADS 附件访问产生器云至 SNADS 附件访问云的连线上的符号“6： SNADS 附件访问（文本串）”表示。

每一个消息对象的处理都是利用 processMessage() 方法实现的。图 30 显示了框架对象进行相互作用以控制处理的处理步骤，并显示出处理是以processMessage()方法调用开始的。 processMessage()方法得到调用，以命令消息中心对象处理各个电子邮件消息，对各个消息对象执行框架的步骤组，并随后从消息队列中删除该对象。在开始时调用 processMessage()的步骤由从消息中心对象向回至其自己的环上的符号“1： processMessage()”表示。该处理以对 removeMessage() 方法的呼叫继续进行，该方法经过框架获得消息对象以进行处理。 该处理由从消息中心至消息队列的连线上的符号“2： removeMessage()”表示。该方法将消息对象送回到消息中心。其余的方法随后依次在消息对象上进行，而该消息对

象被从消息队列除去，如下面描述。

随后的除去消息对象的处理步骤，由符号“3： expandList()”表示，它是消息对象上的 expandList() 方法调用。该方法将对消息的接收者清单进行操作， 以对它们进行可能的扩展。expandList()方法将在下面得到详细描述。下一个处理步骤是调用消息对象上的 resolveAddresses()方法。该方法将对消息的对象接收者清单进行操作，以对未分解的接收者清单中的每一个条目进行“分解”， 从而当该方法返回时，清单中不再有对象。 该处理由从消息中心对象云至消息对象云的连线上的符号“4：resolveAddresses()”表示。该方法的处理将在下文中更详细描述。

在地址被分解之后，处理以对消息对象调用 processEnvelopes()方法继续进行。该方法将支持对框架的任何具体扩展，该扩展在信封清单中的信封对象上进行特定的功能（或使用）。该方法调用由符号“5：processEnvelopes() ”表示，将在下面得到更详细的描述。然后，在消息对象上调用 attachmentProcessing()方法。该方法将支持对框架的所有具体扩展，以对（或使用）附件访问清单中的附件访问对象完成特定的功能。应该注意的是，对任何给定的消息对象事例，可能没有附件访问清单对象。这种处理由“6：attachmentProcessing()表示，并将在下面得到更详细的描述。

图 30 的处理的下一个步骤，是调用 securityAuthority() 方法。该 securityAuthority()方法支持对框架的所有特定扩展， 该扩展执行关于安全或鉴别的特定功能。框架设计者必须决定这可以包括的功能以及什么消息对象的哪些部分可以为该方法所使用。该方法将在下面得到更详细的描述，并在图 30 中由符号“7： securityAuthority()”表示。下一个处理步骤是调用消息对象上的 deliverLocal()方法。 deliverLocal()方法将支持对框架的所有具体扩展，以执行关于电子邮件消息内容的本地分送的特定功能。同样， 框架设计者必须确定它可以包括什么功能以及消息对象的哪些部分可以为该方法所使用。 这些功能不是框架的一部分。该方法的目的是复制消息对象的全部或部分信息。 deliverLocal()方法处理将在下面得到更详细的描述。

在完成了本地分送处理之后，下一个步骤是调用 forwardMessage()方法，该方法支持对框架的特定扩展，以执行关于向网络的其他部分转送电子邮件消息内容的功能。框架设计者必须决定这可能包括的功能，以及消息对象的哪些部分可以为该方法所使用。其目的是复制消息对象的部分或全部信息。该步骤由符号“9： forwardMessage()”表示，并将在下面得到更详细的描述。

在任何电子邮件消息转送之后，下一个步骤是对消息目标调用 handleNonDelivery()方法。该方法支持框架的特定扩展以执行关于电子邮件消息内容的送回或报告功能。同样。框架设计者必须决定这可能包括什么功能，以及消息对象的哪些部分能够为该方法所使用。且其目的是复制消息对象的部分或全部信息（可能是通过产生一个新的“报告”消息 对象）。handleNonDelivery()步骤由符号“10： handleNonDelivery()”表示。

下一个步骤是对消息对象调用 manageAttachments()方法。该方法支持框架的特定扩展，以对（或使用）附件访问清单中的附件访问对象执行特定功能。注意对于任何给定的消息对象，可能没有附件访问清单对象。该处理由“11： manageAttachments()”表示。最后，最后的步骤是对消息对象调用 accounting()方法。会计步骤支持对框架的特定扩展—该扩展执行关于框架处理的电子邮件消息的会计的特定功能。框架设计者必须决定这可能包括什么功能，以及消息对象的什么部分可被该方法所使用。其目的是复制消息对象的必须得到会计处理的部分或全部信息。

如上所述，expandList()方法扩展了消息的接收者清单。图 31 显示了这种处理。如图 31 所示，expandList() 方法对消息的未分解的接收者清单地址条目对象进行操作，以在需要时扩展它们。这种处理的开始由从消息云至未分解的接收者清单云的连线上的“1： expandList()”表示，这表示对象调用未分解的接收者清单对象的 expandList() 方法。然后，expandList() 方法为包含在未分解的接收者清单对象中的每一个地址条目调用 expandAddress()方法，如从未分解的接收者清单云至地址条目云的连线上的“2： expandAddress()”所表示的。

地址条目对象的每一个将对于它们的地址调用该扩展方法。在此例的概要中，该由从地址条目对象至SMTP地址 对象（框架实施者对框架进行的一个扩展）连线上的“3： expand() ”表示。如果SMTP地址对象确定它应该被一个地址清单取代， 它对于未分解的接收者清单对象调用replaceEntry()方法。该处理由从SMTP地址至未分解的接收者清单 的连线上的“4： replaceEntry()”表示。

当expandList()方法已经完成时，所有的未分解的接收者清单地址条目对象都有机会扩展附于消息对象的电子邮件地址。注意框架实施者（或再划分地址对象的任何人）能够决定没有能力扩展这种地址对象，且在此情况下能够实施一个送回给者的空方法。

应该注意的是，在该概要图中没有显示的是，由于expandAddresses()会导致未分解的接收者清单地址条目对象被取代，如果出现了这种情况，expandAddresses() 方法把一个标志送回到消息，以表示有新的未分解的接收者清单地址条目对象。这向消息目标表明它必须再次调用在上述概要中描述的expandList()方法。只要新的条目被expandAddresses()方法加到未分解的接收者清单上， 就继续重复这些方法。

如上所述， resolveAddresses()方法对消息进行操作，以分解消息的未分解的接收者清单中的各个地址条目对象。图32详细描述了这种处理。对于框架，地址条目对象的“分解”意味着地址条目对象被从未分解的接收者清单类中除去， 并被放置在通过框架扩展实施的另一个接收者清单对象类中。该分解过程涉及由消息的resolveAddresses()方法调用未分解的接收者清单对象的resolveAddresses()方法。这在图32中由从消息对象至未分解的接收者清单对象云的连线“1： resolveAddresses()”表示。

随后，未分解的接收者清单对象的resolveAddresses()方法将为未分解的接收者清单对象中包含的每一个地址条目对象调用resolveAddresses()方法。该处理由“2： resolveAddresses( ) ”表示。然后，每一个地址条目对象将就其自己的地址调用resolve()方法。在此例的概要中，该处理由SMTP地址对象（该对象是框架实施者对其框架的扩展）和从地址条目对象云至SMTP地址 对象云的连线上的“3： resolve()”表示。各个地址 对

象确定对于“分解”它应该做什么。用什么标准来进行分解判定，是 resolve() 方法的用户实施的一部分，不由框架定义。如果 SMTP 地址 通过变成 SMTP 接收者清单对象（该对象是框架用户对框架的一个扩展）的一部分而得到分解，它就未分解的接收者清单对象呼叫 removeEntry()方法。或者，地址对象的分解可能导致其地址条目被一个不同的地址所取代。提供这种选择以使电子邮件地址在切换接收者地址时能够被“交换”或“切换”，作为实施电子邮件协议之间的网关或建立可能的电子邮件地址别名的一部分。在图 32 中，这种替换处理由从 SMTP 地址至未分解的接收者清单的连线上的“4： removeEntry()或 replaceEntry()”表示。

在电子邮件地址分解之后， SMTP 地址对象将调用 addEntry()方法，以将其从未分解的接收者清单除去的地址清单条目加到 SMTP 接收者清单对象上。这完成了在此概要中所示的“分解”，并由从 SMTP 地址至 SMTP 接收者清单的连线上的“5： addEntry()”表示。

当完成 resolveAddresses()时，在未分解的接收者清单种类中启动的所有地址条目对象都具有机会被“分解”成附于消息的其他接收者清单对象。注意框架用户（或者再划分地址对象的任何人）都能够确定如果在 resolveAddresses()方法返回到消息之前在未分解的接收者清单对象中包含有任何剩余的地址清单对象时，应该未取什么行动。对消息对象的其他改变（诸如增加或改变信封清单对象）能够成为“分解”未分解的接收者清单地址条目的一部分。这取决于 resolve()方法的框架用户。这在该概要图中没有显示。

应该注意的是，在概要中没有示出的一个这样的事实，即由于 resolveAddresses()方法能够导致未分解的接收者清单地址条目被取代，在发生了这种情况时 resolveAddresses()方法将把一个标志送回消息，表明有新的未分解的接收者清单地址条目对象。这向消息 表明它必须再次调用在上述概要中描述的 expandAddresses() 方法并随后必须再次调用在此概要中所示的 resolveAddresses()。通过这些方法进行的再循环将继续，只要新的地址条目对象通过 expandAddresses()或 resolveAddresses()方法被加到未分解的接收者清单上。

每个消息包括一个信封对象。这些对象用在消息目标上调用的processEnvelopes()方法进行处理，如图33所示。该方法支持对框架的任何特定扩展—这些扩展执行（或使用）在信封清单中的信封对象上的功能。应该注意的是，框架假定任何所希望的信封对象处理都是围绕出现在电子邮件消息上的接收者电子邮件地址的类而设计的。这意味着processEnvelopes()方法是作为接收者清单框架对象的扩展的一部分而实施的。当考虑localDelivery()或messegeForwarding()方法时，该更容易理解。对于具有SMTP接收者的电子邮件消息和具有SNADS接收者的电子邮件消息（在SNADS接收者清单对象中有至少一个地址条目）来说，在消息对象的信息上进行的功能可能是完全不同的。这些方法由框架用户进行的接收者清单对象的扩展定义和实施。事实上，前述的所有概要都反映了框架的这种主要的设计假定，当框架用户设计和扩展框架时他必须考虑这点。

首先，对于附于消息的每一个接收者清单对象，processEnvelopes()方法将调用processEnvelopes()方法。在此概要中，假定有两个，即SMTP接收者清单和SNADS接收者清单。先对SMTP接收者清单调用该方法，如从消息对象云至SMTP接收者清单对象云的连线上的“1：processEnvelopes()”所表示的。然后，processEnvelopes()方法将对消息对象的信封清单对象调用retrieve()或add()。这种处理由至信封清单的连线上的“2：retrieve()或add()”表示。该概要假定寻找电子邮件消息的一个信封并可能将其作为执行调用的一部分而加到processEnvelopes()方法上。processEnvelopes()方法的实施者将设计processEnvelopes()在电子邮件消息上实际执行的功能。

随后，消息的processEnvelopes()方法将调用附于该消息的下一个接收者清单对象的processEnvelopes()方法。图33的处理显示，对SNADS接收者清单，从消息对象云作出了调用“3：processEnvelopes()”。象前面一样，processEnvelopes()方法将对消息对象的信封清单调用retrieve()或add()方法，由“4：retrieve()或add()”表示。该概要再次假定，作为对processEnvelopes()执行调用的一部分，电子邮件消息的信封将被查看并可能被加上。

除了消息对象和信封对象处理，框架还实施附件处理。图 34 显示了这种处理。首先，对消息对象调用 attachmentProcessing()方法。该方法支持对框架的任何特定扩展—该扩展在附件访问清单中的附件访问对象上执行（或使用）特定的功能。注意对于任何给定的消息对象，可能没有附件访问清单。

在附件处理中，首先消息的 attachmentProcessing()方法将调用附在消息上的每一个接收者清单对象的 attachmentProcessing()方法。在此概要中，假定有两个接收者清单，一个 SMTP 接收者清单和一个 SNADS 接收者清单。首先对 SMTP 接收者清单调用 attachmentProcessing()方法，如从消息云至 SMTP 接收者清单云的连线上的“1： attachmentProcessing()”所表示的。然后， processEnvelopes()方法将对消息 对象的附件访问清单调用 retrieve()或 add()方法。该概要假定，电子邮件消息被访问的附件将被查看或可能作为对 attachmentProcessing()执行调用的一部分而被加上。attachmentProcessing()方法的实施者将设计 attachmentProcessing()在电子邮件消息上将实际执行的功能。该处理在图 34 中由至附件访问清单的线上的“2： retrieve()或 add()”表示。

下一个附件处理步骤，是由消息的 attachmentProcessing()方法调用附在消息上的下一个接收者清单对象的 attachmentProcessing()方法。图 34 显示了对 SNADS 接收者清单对象云正在进行的调用“3： attachmentProcessing()”。然后， attachmentProcessing()将对消息对象的附件访问清单呼叫 retrieve()或 add()方法，如“4： retrieve()或 add()”所示。 和前面一样，该概要假定，电子邮件消息的被访问的附件将被查看，或可能作为对 attachmentProcessing()执行调用的一部分而被加上。

在最佳实施例中，框架支持安全处理。这由图 35 表示，它描述了 securityAuthority()方法。该方法是在消息对象上调用的，并支持对框架的任何特定扩展—该扩展执行有关安全或鉴别的特定功能。框架设计者必须决定这可能包括的功能和该方法所可能使用的消息对象部分。

在安全处理中，首先消息的 securityAuthority()方法调用附在消息上的每一个接收者清单对象的 securityAuthority()方法。在此概要中，有两个接

收者清单对象， 一个SMTP接收者清单对象和SNADS接收者清单对象。首先对SMTP接收者清单调用securityAuthority()方法，这在图35中由从消息至SMTP接收者清单的线上的“1： securityAuthority()”表示。securityAuthority()方法的实施者必须设计出该方法提供什么类的安全或鉴别功能。在该概要中没有显示的还有可以获取消息的任何对象（诸如发出者清单、信封清单或附件访问清单）且可以在执行安全或鉴别功能的过程中使用该对象。然后，消息的securityAuthority()方法将调用附在消息上的下一个接收者清单对象的securityAuthority()方法。图35显示了对SNADS接收者清单进行的调用，由从消息至SNADS接收者清单的线上 的“2： securityAuthority() ”表示。

某些消息将具有本地目的地，这意味着它们是送给与操作系统处于同一网络上的接收者的。在此情况下，采用deliverLocal()方法。图36显示了这种方法的处理。deliverLocal()是在消息 对象上被调用的，并支持对框架的任何特定扩展—该扩展执行有关电子邮件消息内容的本地分送功能。框架设计者必须决定这可以包括什么功能，以及消息对象的哪些部分可以被该方法 使用。

图36显示出，在该处理中，首先将deliverLocal()应用到消息对象上。该方法对附在消息上的每一个接收者清单调用deliverLocal()方法。在图36的概要中，处理的第一个地址清单是SMTP接收者清单，如从消息至SMTP接收者清单 的线上的“1： deliverLocal()”所示。 该deliverLocal( ) 方法被应用到SMTP接收者清单对象上，以控制电子邮件消息至本地接收者的分送。该方法调用附件访问清单上的retrieve()方法， 以获得与消息有关的附件访问对象的清单，如“2： retrieve()”所表示的。

一旦获取了与消息有关的附件访问对象，对信封清单类调用retrieve()方法，以获取信封清单对象。这由至信封清单的线上的“3： retrieve()”表示。 一旦获取到了附件访问对象和信封对象， deliverLocal() 方法准备通过SMTP接收者清单来执行基本的分送功能。 这可以通过获取SMTP接收者清单中的每一个地址条目并检查接收者数据以确定电子邮件消息接收者是本地的还是远程的来实现。如果电子邮件接收者是本地的，则执行

分送所需的处理，如框架用户所确定的。

随后，deliverLocal()方法被应用到SNADS接收者清单上。该方法是要控制至本地接收者的电子邮件消息分送，如从消息至SNADS接收者清单的线上的“4： deliverLocal()”所表示的。该方法又调用附件访问清单上的retrieve()方法，以获得与消息有关的附件访问对象清单。该处理由从SNADS接收者清单至附件访问清单的线上的“5： retrieve()”表示。一旦获得了与消息有关的附件访问对象，retrieve()方法对信封清单调用retrieve()方法，以获取信封对象。这由至信封清单的线上的“6： retrieve()”表示。一旦获取了附件访问和信封对象，deliverLocal()方法准备通过SNADS接收者清单来执行基本的分送功能。这可以通过获取SNADS接收者清单中的各个地址条目并检查接收者数据以判定电子邮件接收者是本地还是远程来进行。如果电子邮件接收者是本地的，则如框架用户所确定的那样进行分送消息所需的处理。

如果消息不是给本地电子邮件接收者的，则它必须被转送到网络的另一部分。在此情况下，不是框架组成部分的另一个处理程序将电子邮件消息内容传送到另一个系统。这种处理涉及图36中所示的forwardMessage()方法。该forwardMessage()方法在消息对象上得到调用，并支持任何特定扩展—该扩展执行与向网络的另一部分转送电子邮件消息的内容有关的功能。框架设计者必须决定这可能包括的功能和该方法可以使用的消息对象部分。它的目的是复制消息对象的部分或全部消息。

图37显示，forwardMessage()方法被应用到消息对象上，并对附在消息上的每一个接收者清单对象而得到调用。在此概要中，处理的第一个地址清单是SMTP接收者清单，如从消息至SMTP接收者清单的连线上的“1： forwardMessage()”所表示的。该forwardMessage()方法被应用到SMTP接收者清单上，以控制电子邮件消息至远程接收者的分送。该方法随后在附件访问清单上调用retrieve()方法，以获得与消息有关的附件清单。该处理由“2： retrieve()”表示。

一旦获取了与消息有关的附件访问对象，对信封清单调用retrieve()方法，以获取信封清单。这由从SMTP接收者清单至信封清单的连线上的

“3： retrieve()”表示。在获取了消息附件访问和信封对象之后，forwardMessage()方法准备通过SMTP接收者清单以执行基本的分送功能。这是通过获取SMTP接收者清单中的各个地址条目并检查接收者数据以判定电子邮件接收者是本地还是远程来进行的。如果电子邮件接收者是程的，则如框架用户确定的那样进行分送消息所需的处理。

随后，forwardMessage()被应用到SNADS接收者清单上，如从消息至SNADS接收者清单的线上的“4： forwardMessage()”所示。该方法控制至远程电子邮件接收者的电子邮件消息分送，并在附件访问清单上调用retrieve()方法，以获得与消息有关的附件访问对象清单。该处理由至附件访问清单的连线上的“5： retrieve()”表示。一旦获取了与消息有关的附件，对信封清单调用retrieve()方法，以获取信封清单对象，如至信封清单的连线上的“6： retrieve()”所示。一旦获取了附件访问和信封对象，forwardMessage()方法准备通过SNADS接收者清单，以执行基本的分送功能。这可通过获取SNADS接收者清单中的各个地址条目并检查接收者数据以判定电子邮件接收者是本地还是远程来进行。如果电子邮件接收者是远程，则如框架用户确定的那样进行转送消息所示的处理。

当电子邮件消息被送回或如果希望对电子邮件消息的报告时，发生一个未分送事件。这种事件涉及图38所示的handleNonDelivery()方法。该方法在消息对象上得到调用，并支持任何框架的特定扩展—该扩展执行有关电子邮件消息内容的送回或报告的功能。框架设计者必须决定这可能包括的功能和该方法可以使用的消息对象部分。其目的是复制必须报告的消息对象信息的部分或全部（可能通过产生新的“报告”消息对象）。

图38显示，处理从handleNonDelivery()方法开始—该方法调用附在消息上的各个接收者清单对象的handleNonDelivery()方法。在此概要中，有两个接收者清单，SMTP接收者清单和SNADS接收者清单。在图38中，首先对SMTP接收者清单对象调用该方法，如从消息对象云至SMTP接收者清单对象云的连线上的“1： handleNonDelivery()”所示。handleNonDelivery()方法的实施者必须决定该方法提供什么类的未分送功能。该方法随后调用SNADS接收者清单，如从消息至SNADS接收者清

单的连线上的“2： handleNonDelivery()”所示。handleNonDelivery() 方法的实施者必须决定该方法提供什么类的未分送功能。

在某些情况下，消息对象可以包括必须得到处理的附件访问清单。这由图 39 所示的 manageAttachments()方法处理。该方法在消息对象上得到调用， 并支持对框架的任何特定扩展—该扩展执行（或使用）附件访问清单中的附件访问对象上的特定功能。注意对于给定的消息对象，可能没有附件访问清单。

图 39 的处理以消息的 manageAttachments()方法开始，该方法调用附在消息上的每一个接收者清单对象的 manageAttachments()方法。在图 39 的概要中，假定有两个接收者清单， SMTP 接收者清单和 SNADS 接收者清单。 先对 SMTP 接收者清单调用该方法， 如从消息至 SMTP 接收者清单的连线上的“1： manageAttachments() ”所示。然后，manageAttachments()方法将对消息对象的附件访问清单调用 retrieve()方法。该概要假定，电子邮件消息的被访问的附件将被查看，并可能被作为执行 manageAttachments()调用的一部分。 manageAttachments()方法的实施者将设计 manageAttachments()实际对电子邮件消息将执行什么功能。该处理步骤在图 39 中用至附件访问清单的线上的符号“2： retrieve()”表示。

下一个处理，是由消息的 manageAttachments()方法调用附在消息上的下一个接收者清单对象上的 manageAttachments() 方法。图 39 显示了对 SNADS 接收者清单进行该调用，如“3：manageAttachments()”所表示的。然后，接收者清单的 manageAttachments()方法将对附件访问清单调用 retrieve()方法，如“4:retrieve()”表示的。象前面一样，该概要假定，作为对 manageAttachments()的执行调用的一部分，电子邮件消息被访问的附件将被查看。

最后，框架提供了会计功能，以跟踪对消息处理等的收费。图 40 所示的 accounting()方法在消息对象上得到调用，并支持对框架的特定扩展—该扩展执行与对框架处理的电子邮件消息的会计处理有关的特定功能。框架设计者必须决定这可能包括什么功能和消息对象的哪些部分可以被该方法

首先，消息的accounting()方法调用附在消息上的每一个接收者清单对象的accounting()方法。在此概要中，假定有两个接收者清单，SMTP接收者清单和SNADS接收者清单。在图40中，首先对SMTP接收者清单调用该方法，如从消息对象云至SMTP接收者清单对象云的连线上的“1：accounting()”所示。然后accounting()方法对消息对象的发出者清单呼叫retrieve()方法，如至发出者清单云的线上的“2： retrieve()”所示。该概要假定，该会计处理方法寻找一段电子邮件的发出者地址，作为执行对会计处理的调用的一部分。会计处理的实施者必须设计该方法提供什么类的会计处理功能。该概要没有显示消息的任何其他对象能够被获取（诸如接收者清单、信封清单或附件访问清单）以及对象可被用于执行会计处理功能。

在下一个处理步骤中，消息的accounting()方法在附于消息的下一个接收者清单对象上调用accounting()方法。图40显示了正在对SNADS接收者清单进行的调用，如从消息 目标云至SNADS对象云的连线上的“3：accounting()”所示。最后， accounting()方法处理对消息对象的发出者清单调用retrieve()方法，如从SNADS接收者清单至发出者清单的连线上的“4： retrieve()”所示。同样， 该概要假定会计处理方法寻找一段电子邮件的发出者地址，作为执行会计处理调用执行的一部分。

这里给出的实施例和例子，是为了最好地说明本发明及其实际应用，从而使本领域的技术人员能够作出和使用本发明。然而，本领域的技术人员应该理解，前述的描述和例子只是为了说明和举例的。所给出的描述不是排他的，也不是要将本发明限制在公布的具体形式。在不脱离本发明的精神和范围的前体下，借助以上的描述，可以进行很多修正和变形。

记号

在交流面向对象的程序设计的思想方面，还没有一致接受的记号。在本说明书中所用的记号，与程序设计领域中被称为Booch 记号（以Grady Booch 命名）的非常类似。Booch 先生是可从TheBenjamin/Cummings Publishing Company, Inc.获得的Object-Oriented Analysis and Design

With Applications, 2d ed.(1994)一书的作者。在本说明书中使用 Booch 记号，不应该被理解为本专利申请的发明者和/或受让人与 Booch 先生或 Booch 先生的雇主之间有什么联系。Booch 先生所用的记号体系，在上述书的第 5 章（171—228 页）进行了更为详细的描述。下面将大体上描述这里采用的记号体系。这里采用的其他记号约定，将根据需要得到描述。

由面向对象的框架模型化的系统，可以高度抽象地用称为顶级类图的图来表示。图 1 是顶类图的一个例子，它包含代表模型化的系统的框。这些框以层级结构排列，从而使代表与系统的物理部件接近的抽象的框处于图中的较低级，而代表更抽象的功能部件的框更接近图的顶部。在图 1 中，这些框被标有“机构”，以表示这些抽象包括用于实施模型化的系统部件的装置。这些框（机构）可被认为是类别—它们包括按照面向对象的程序设计的概念定义的相似类组。图 1 代表了动物园管理模型，因而较低的层级结构框包括被称为动物机构的框—它代表在动物园模型中的动物，以及一个被称为圈养单元机构的框—它代表动物围栏和笼子。在图 1 的最高级，被称为动物园管理的框代表包含由人执行的各种管理任务的功能抽象。

在顶级类图中的框，代表了提供系统行为的系统抽象。该系统抽象包括类和对象。各系统类的细节在类图中提供，而该类图被用来显示类的类别并表示各类的关系和责任。类由形状不规则的、虚线图象表示，这些图象通常被称为云。例如，图 2 显示了被表示为云的几个类。各个类由对于有关的类类别来说是唯一的名来标明，且该名还表示了各个类与图 1 所示的机构之一的关系。在类图象中，类名被列在属性、操作或方法名的上方。随后跟着的是圆括号和包括在括号中的限制。图 3 更详细地显示了动物园管理类。图 3 表示，动物园管理类包括多个操作，包括被称为 5_minute_timer()、add_animal( )和 add_containment_unit()的操作。操作名（和类属性名）中的词被分隔开，以便于阅读。在图 5 中所示的类 Animals 中的被称为 feedfreq 和 temp range 的属性，显示了类属性 List 的一个事例。

机构（图 1）和类（图 2）之间的连线，表示了这些抽象之间的关系的性质。因此，图 1 中的框之间的联接，代表了各种机构之间的关系。例

如，直的连线表示共享信息的简单联系。“使用”关系是简单联系的一个明确表达，在这种联系中被称为服务器或提供者的抽象向被称为用户的另一个抽象提供服务。这种关系由在简单联系线的一端的开放圆圈表示，该开放圆圈的端部指示“使用”有关的服务器的用户。

两个类之间的简单联系的另一个明确表达，是被称为继承关系的类型。继承是类之间的一种关系，其中一个类分享与一个或多个其他类有关的结构和/或行为。 继承联系也被称为“是一个”关系。因此，给定两个类A和B，如果类A是B的一个例子，则A与类B有继承关系；A被称为是B的子类，且B被称为是A的母类。即，A“是一个”B。继承关系用一端带有箭头的连线表示，以表示一个子类，该子类从位于连线另一端的母类导出其特性。

类关系的另一个明确表达，被称为聚集关系，它表示全部与其部分或属性类之间的联系。在记号上，全部类与属性类用联系线相联，且它们之间的聚集关系用在全部类端部的实圆圈表示，而属性处于另一端。

类图表示的另一种关系，是例示关系。例示关系代表类的一个事例，诸如由程序设计语言支持的类的一种具体实施。例如，称为“动物”的类可以具有多个例示，包括狮子、老虎和熊。类的例示用带有箭头的虚联系线表示，箭头从类的事例指向一般的类。

最后，被称为亚类的类关系，表示这样的关系，即其中类自身被当作一个可被操纵的对象。即，亚类是这样的类，即其事例本身也是类。某些计算机语言，诸如Small Talk，支持亚类概念。这种关系用带有箭头的阴影线表示，其中箭头从亚类的事例指向总的亚类。

类可以被参数化，它表示其结构和行为被与其形式类参数相独立地得到定义的类组。参数化的类由云形的类图象表示，其中矩形框被置于云部分的上方。参数清单在矩形框之内命名。例示的类包括参数框—称为装饰，与用于一般类的虚线框形成对照。参数化类与其例示的类之间的例示关系，由指向参数化的类的虚线表示。通常，例示的类要求与另一实际类的“使用”关系，以被用作实际的参数。

类的性质可用包含在类云图象中的类装饰表示。具体地，抽象类用位

类的性质可用包含在类云图象中的类装饰表示。具体地，抽象类用位于云中的三角内的大写“A”表示。抽象类是这样的类，即对于它不能产生事例。即，它是类组成的类。其他的类装饰是OO实施语言的功能。例如，C++ 语言允许特定的类资格，该资格将被给予特定的装饰。静态类由装饰三角内的大写“S”表示，朋友类由装饰三角内的大写“F”表示，且虚拟类由装饰三角中的大写“V”表示。

除了定义类，面向对象的程序设计系统的设计者必须定义对象（见Booch 书的第 136 页）。对象用实线云表示，在该云中对象属性上方有对象名。对象是实际的实体，它呈现确定的行为。对象被用来表示实际系统的某些部分，这些部分由面向对象的程序设计表示。对象的特征在于状态、行为和身份。对象可被认为是类的一个事例。对象的行为是对对象在其状态改变和其消息传送方面如何行动和反应的表示。

对象和它们之间的关系被表示在对象图中，该对象图包括具有联接线的对象图象，这些联接线表示了对象之间的同步。联接线被依次编号，以表示操作流程。在两个对象之间存在有联接线，表明了它们对应的类之间的联系，并表示了它们之间的通信路径。因此，两个对象之间的联接线，表明一个对象可以向另一个传送消息。消息传送的方向用在简单的连线上加箭头来表示，该箭头从调用操作的对象（被称为用户）指向提供该操作的对象（称为提供者）。对简单的同步关系的这种代表，表示了最简单形式的消息传送。这种联系可以表示例如操作的调用。操作参数可被表示在联接线附近。

某些对象可以是主动的，这意味着它们实现它们自己的控制线。即这种对象不只是顺序性的。主动对象可以具有各种当前特性。如果对象具有多个控制线，则同步必须得到说明。消息同步化可以是同步的，这意味着用户将等候，直到提供者接受消息。同步的同步化，用带有箭头的X表示。同步可包括阻止消息传送，这意味着如果提供者不能立即给消息提供服务，用户将放弃消息。阻止作用由指回到自身的箭头表示。同步可以包括定时同步，这意味着如果提供者不能在指定的时间里给消息提供服务，用户将放弃消息。定时同步用联接线箭头附近的时钟来表示。最后，同步可

将该消息排队，且用户随后不等候提供者而继续进行操作。本领域的技术人员应该理解的是，异步的消息同步与中断处理类似。异步的消息同步用半箭头表示。

应该指出，Booch 记号包括跟踪对象和类的执行的作用图。作用图基本上是重新组成的对象图。即，作用图与对象图相比并没有给出更多的信息，而只是以不同的形式提供了相同的信息。本说明书采用了对象图而不是作用图，但本领域的技术人员应该理解它们是等价的，且应该理解如何从一个转换到另一个，而不用再作进一步的说明。

例如，在图 7 中，被称为 Zelda 706 的对象，通过调用来自被称为 Zoo keeper Register 的对象的、被称为 List Zoo keepers 的操作，而获得了当前动物园工作人员的清单。在图 7 中， 第二个处理步骤，用 Zoo keeper Register 对象表示—该对象通过将一个消息传送到表示该动物园工作人员清单的 Zelda 对象而响应该操作调用。该动物园工作人员对象包括被称为 Tina 、 Vince 和 Fred 的 Zoo keeper 类的成员。该对象图中表示的第三个步骤，是由对象 Zelda 将一个消息传送给各个动物园工作人员，通过调用各个动物园工作人员对象的相应 Check Animals 操作，命令它们检查动物。

图 1

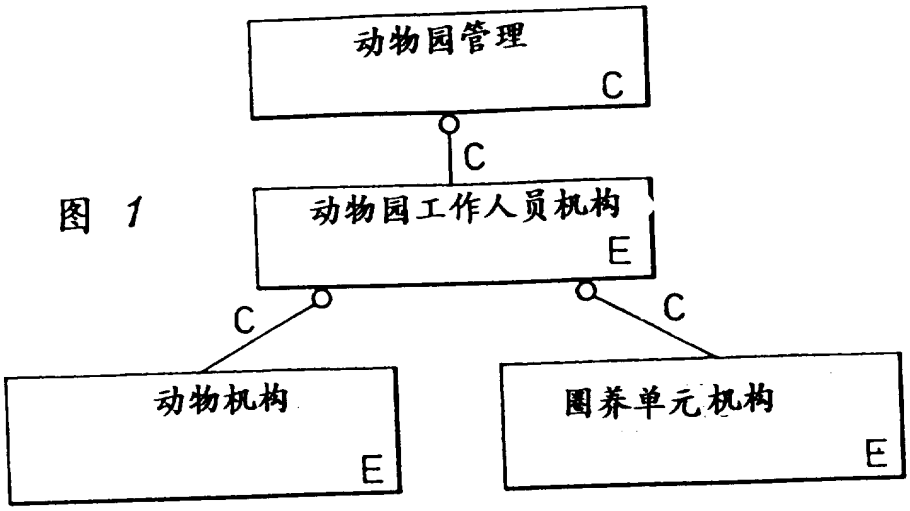

动物园管理
C
C
动物园工作人员机构
E
C
C
动物机构
E
圈养单元机构
E

图 3

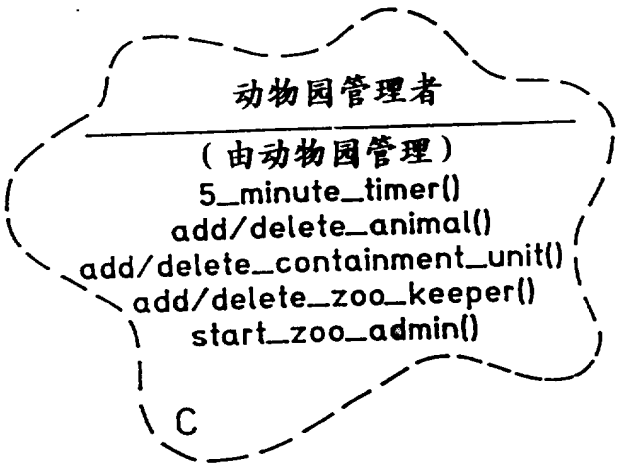
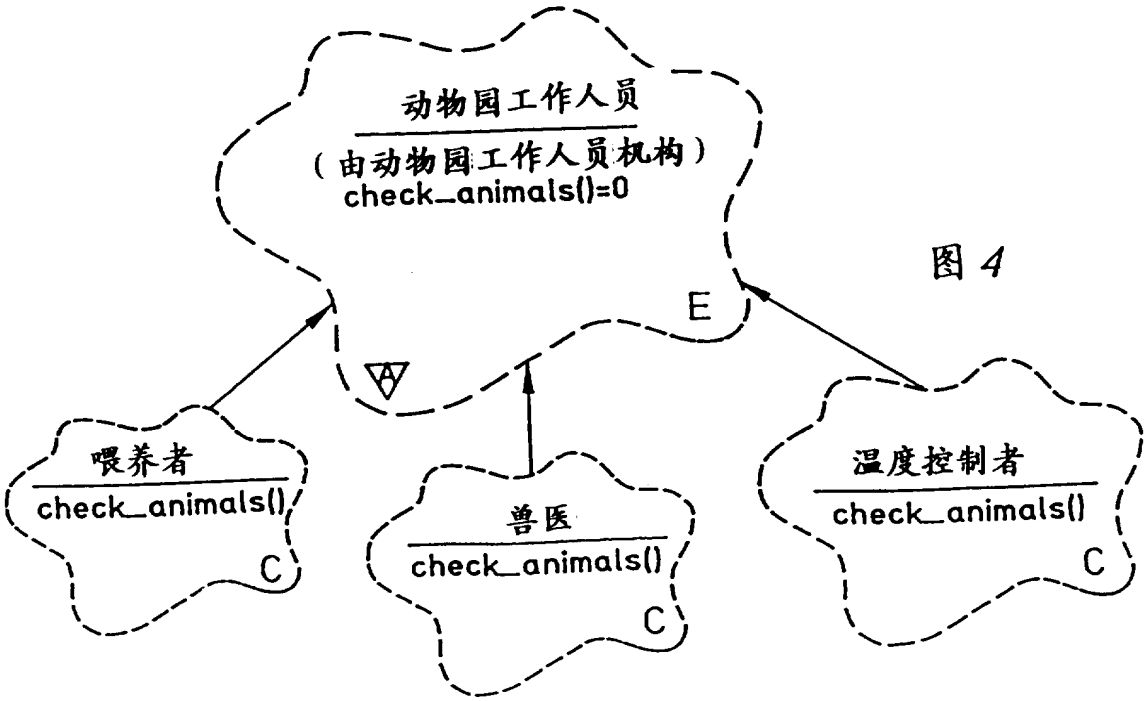

动物园管理者
(由动物园管理)
5_minute_timer()
add/delete_animal()
add/delete_containment_unit()
add/delete_zoo_keeper()
start_zoo_admin()
C

图 4


动物园工作人员
(由动物园工作人员机构)
check_animals()=0
E
喂养者
check_animals()
C
兽医
check_animals()
C
温度控制者
check_animals()
C

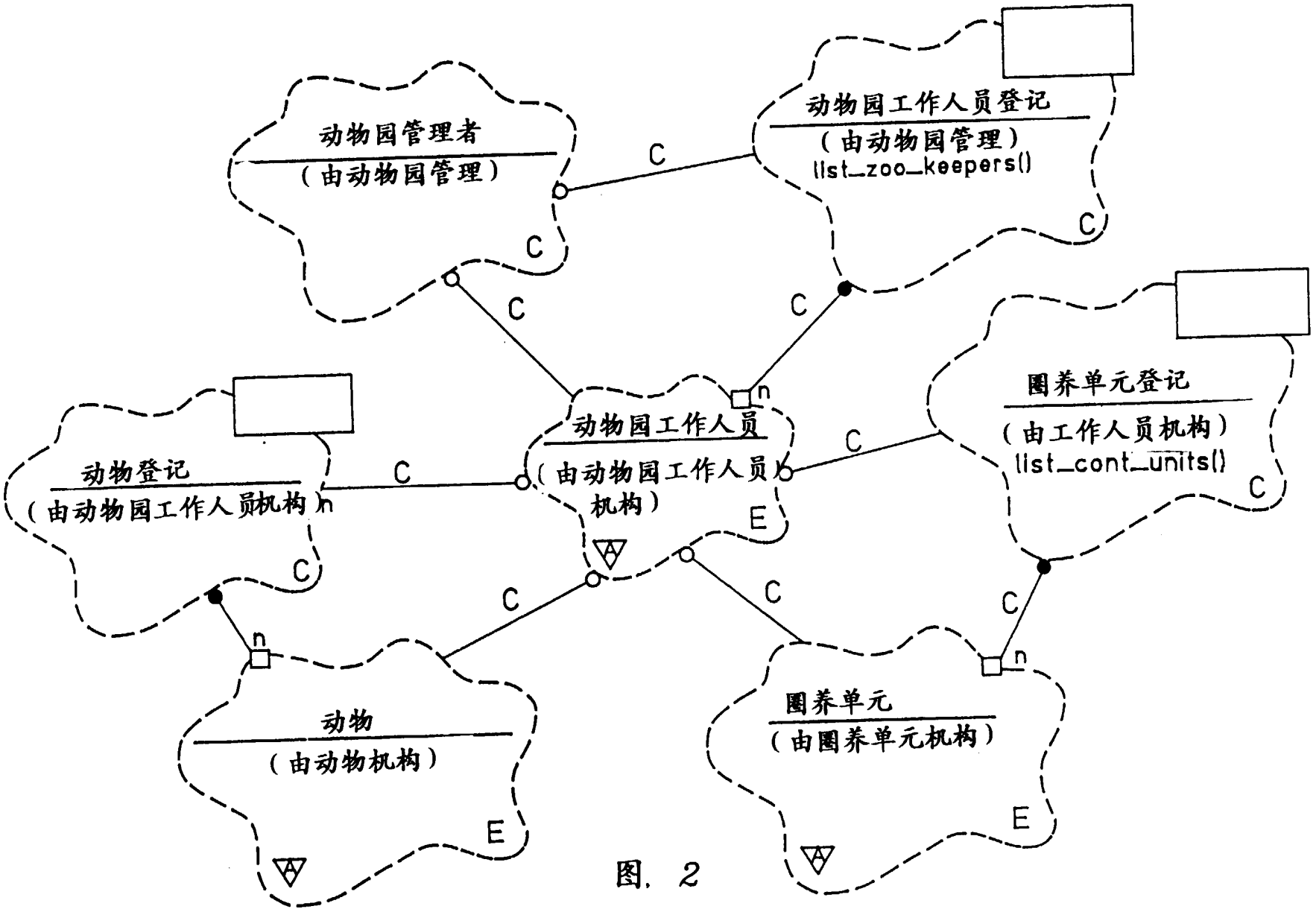

动物园管理者
（由动物园管理）
动物园工作人员登记
（由动物园管理）
list_zoo_keepers()
动物登记
（由动物园工作人员机构）
动物园工作人员
（由动物园工作人员机构）
圈养单元登记
（由工作人员机构）
list_cont_units()
动物
（由动物机构）
圈养单元
（由圈养单元机构）
C
n
E

图. 2

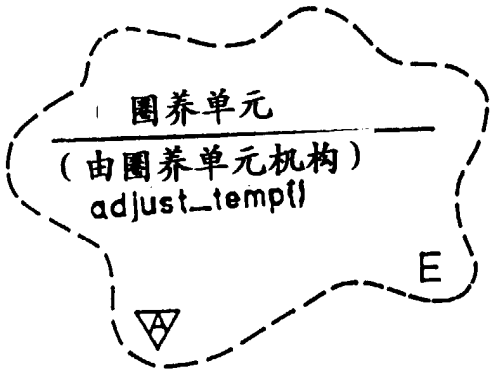

圈养单元
(由圈养单元机构)
adjust_temp()
E

图 6

动物
(由动物园机构)
Feed_freq
Location
Temp_range
Vet_visit_freq
get_temp_range()
feed()=0
needs_food()=0
needs_vet_visit()=0
vet_visit()=0
E
哺乳动物
E
爬行动物
Temp_range
get_temp_range()
E
食肉动物
feed()
needs_food()
E
食草动物
feed()
needs_food()
E

图 5

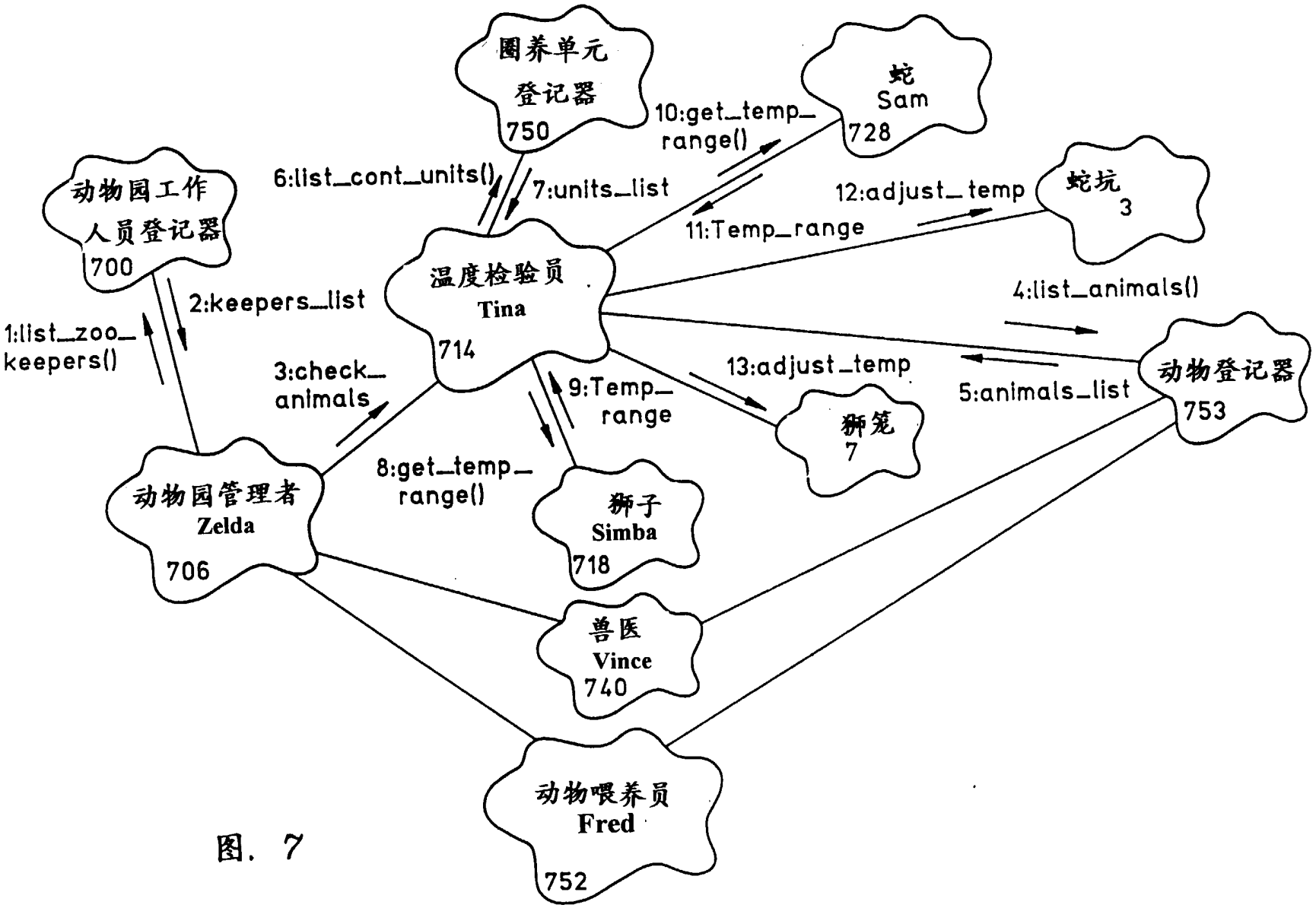

圈养单元
登记器
750
蛇
Sam
728
10:get_temp_
range()
6:list_cont_units()
7:units_list
动物园工作
人员登记器
700
12:adjust_temp
蛇坑
3
11:Temp_range
温度检验员
Tina
714
4:list_animals()
2:keepers_list
1:list_zoo_
keepers()
3:check_
animals
13:adjust_temp
5:animals_list
动物登记器
753
9:Temp_
range
狮笼
7
8:get_temp_
range()
动物园管理者
Zelda
706
狮子
Simba
718
兽医
Vince
740
动物喂养员
Fred
752

图. 7

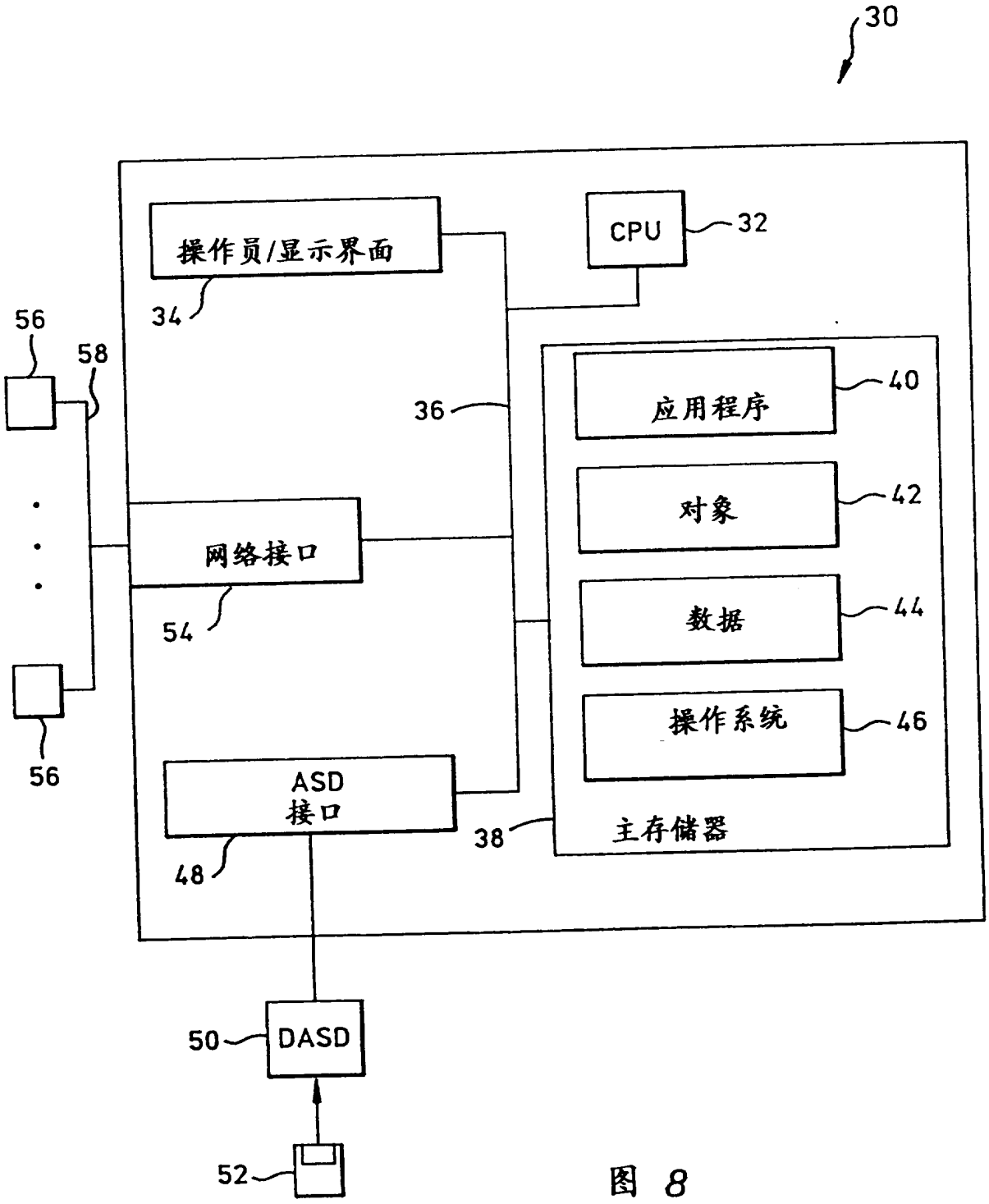

30
操作员/显示界面
34
CPU
32
56
58
36
应用程序
40
对象
42
网络接口
54
数据
44
操作系统
46
56
ASD
接口
48
38
主存储器
50
DASD
52

图 8

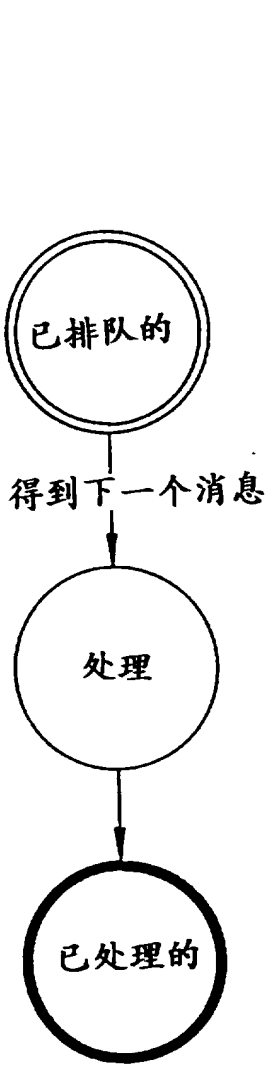


图 9

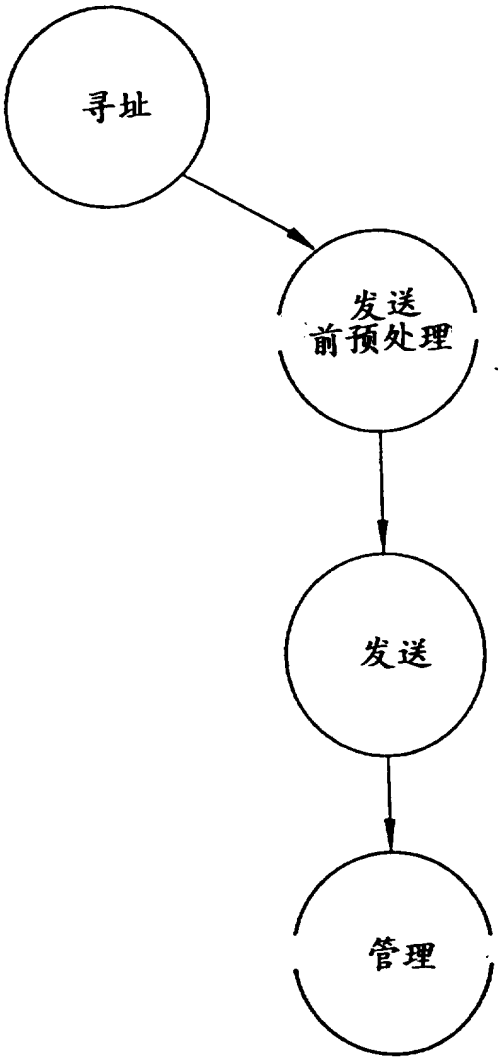


图 10

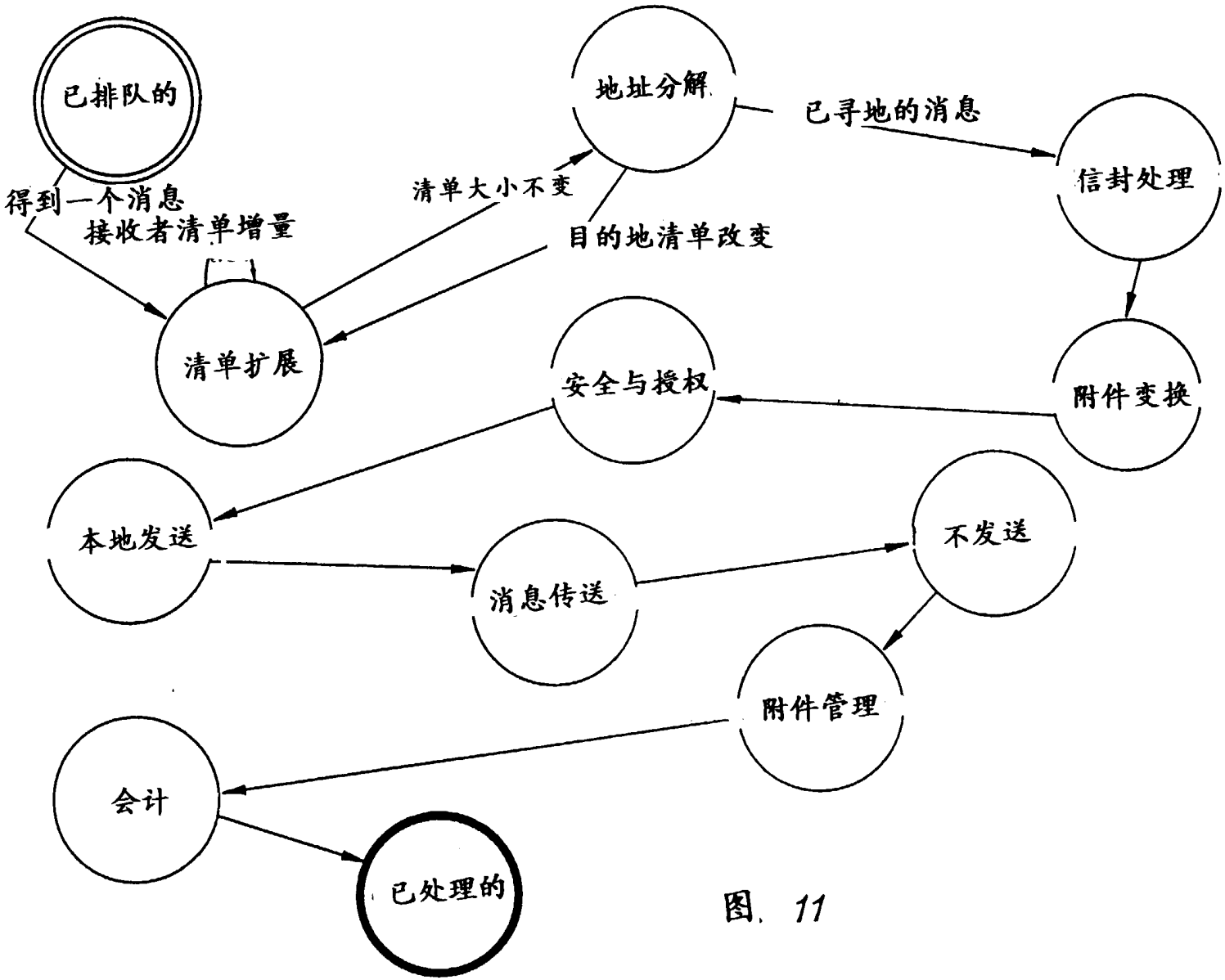

已排队的
得到一个消息
接收者清单增量
清单扩展
清单大小不变
地址分解
目的地清单改变
已寻地的消息
信封处理
附件变换
安全与授权
本地发送
消息传送
不发送
附件管理
会计
已处理的

图. 11

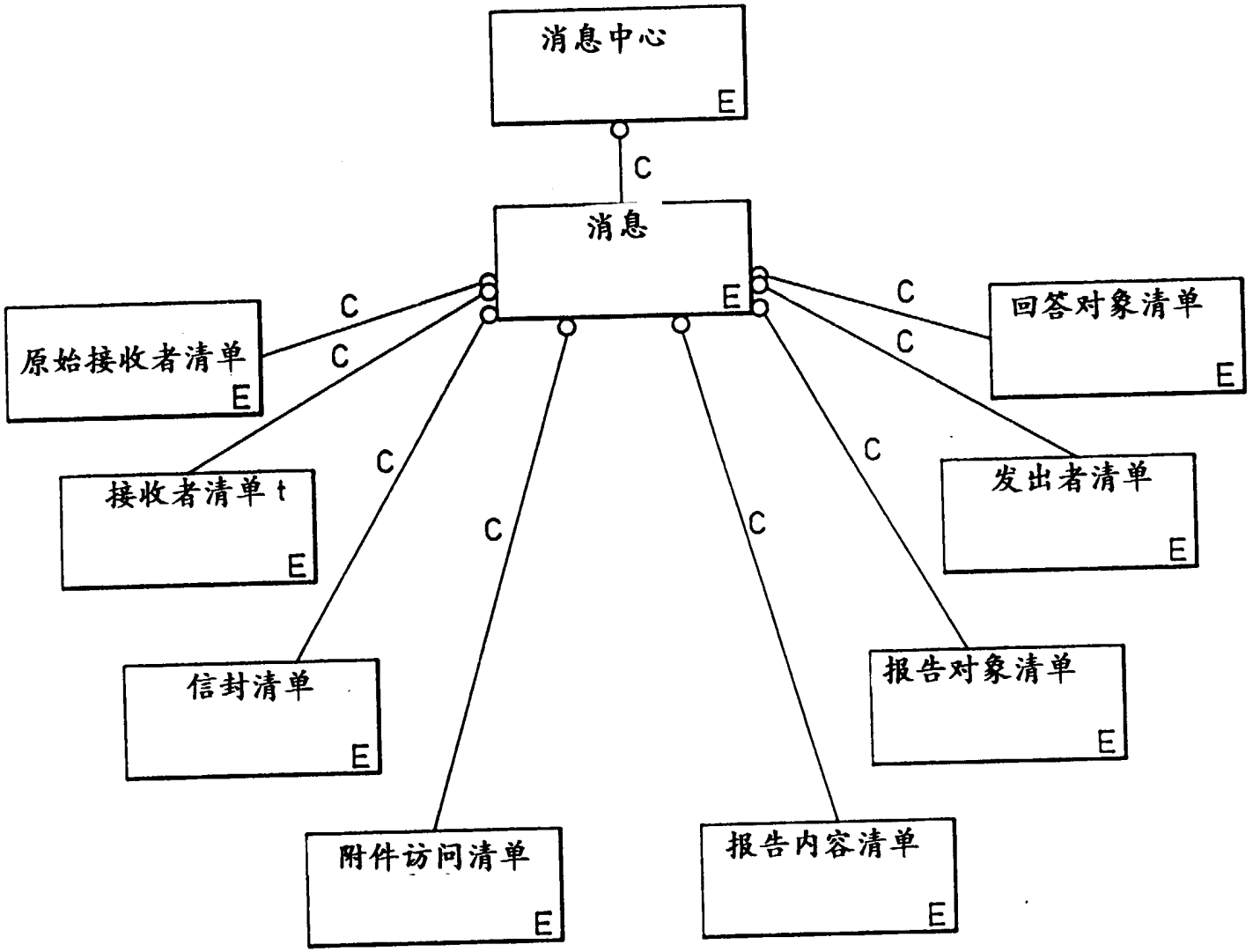

消息中心
E
C
消息
E
C
原始接收者清单
E
C
接收者清单 t
E
C
信封清单
E
C
附件访问清单
E
C
报告内容清单
E
C
发出者清单
E
C
报告对象清单
E
C
回答对象清单
E

图 12

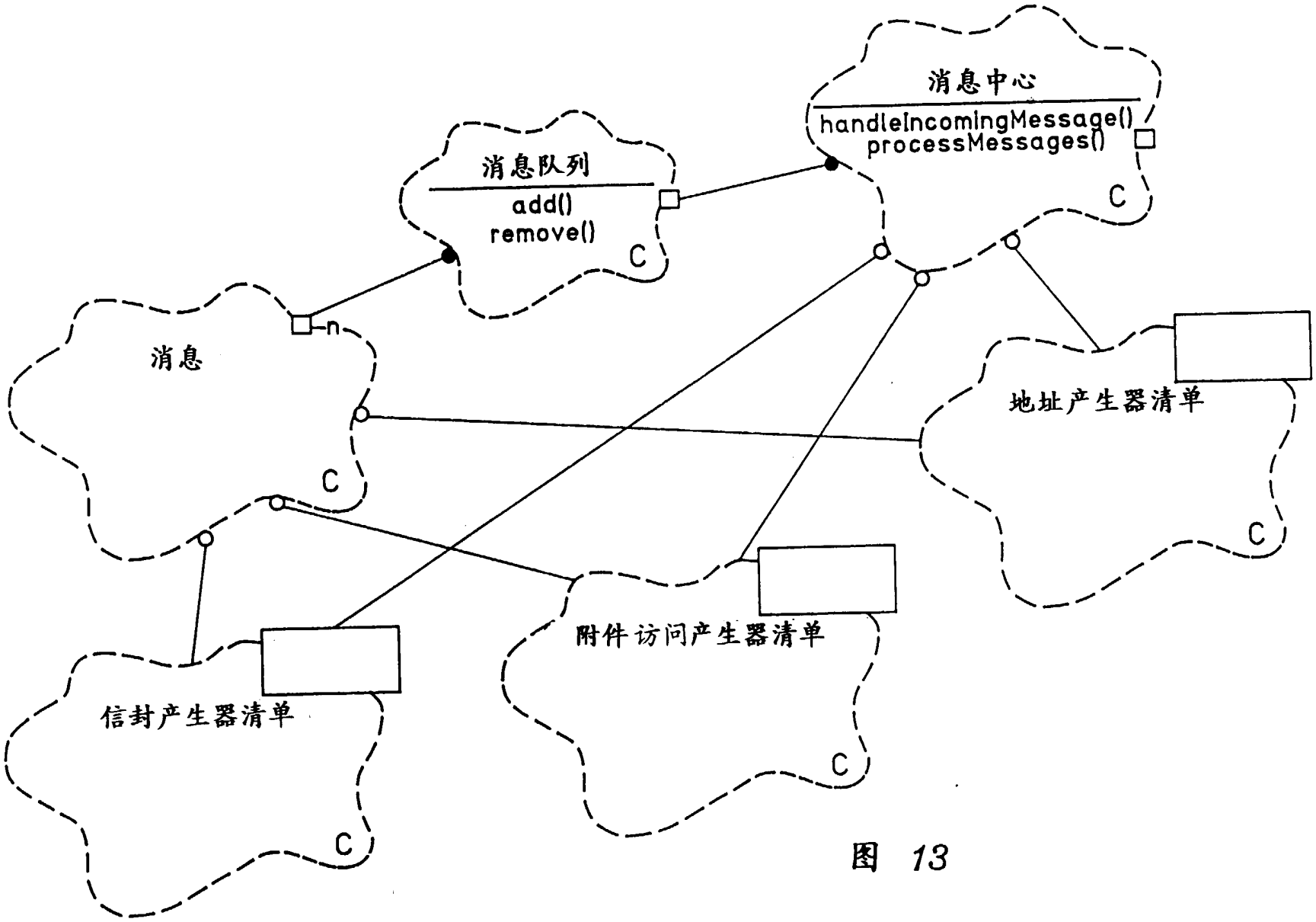

消息中心
handleIncomingMessage()
processMessages()
C
消息队列
add()
remove()
C
消息
n
C
地址产生器清单
C
附件访问产生器清单
C
信封产生器清单
C

图 13

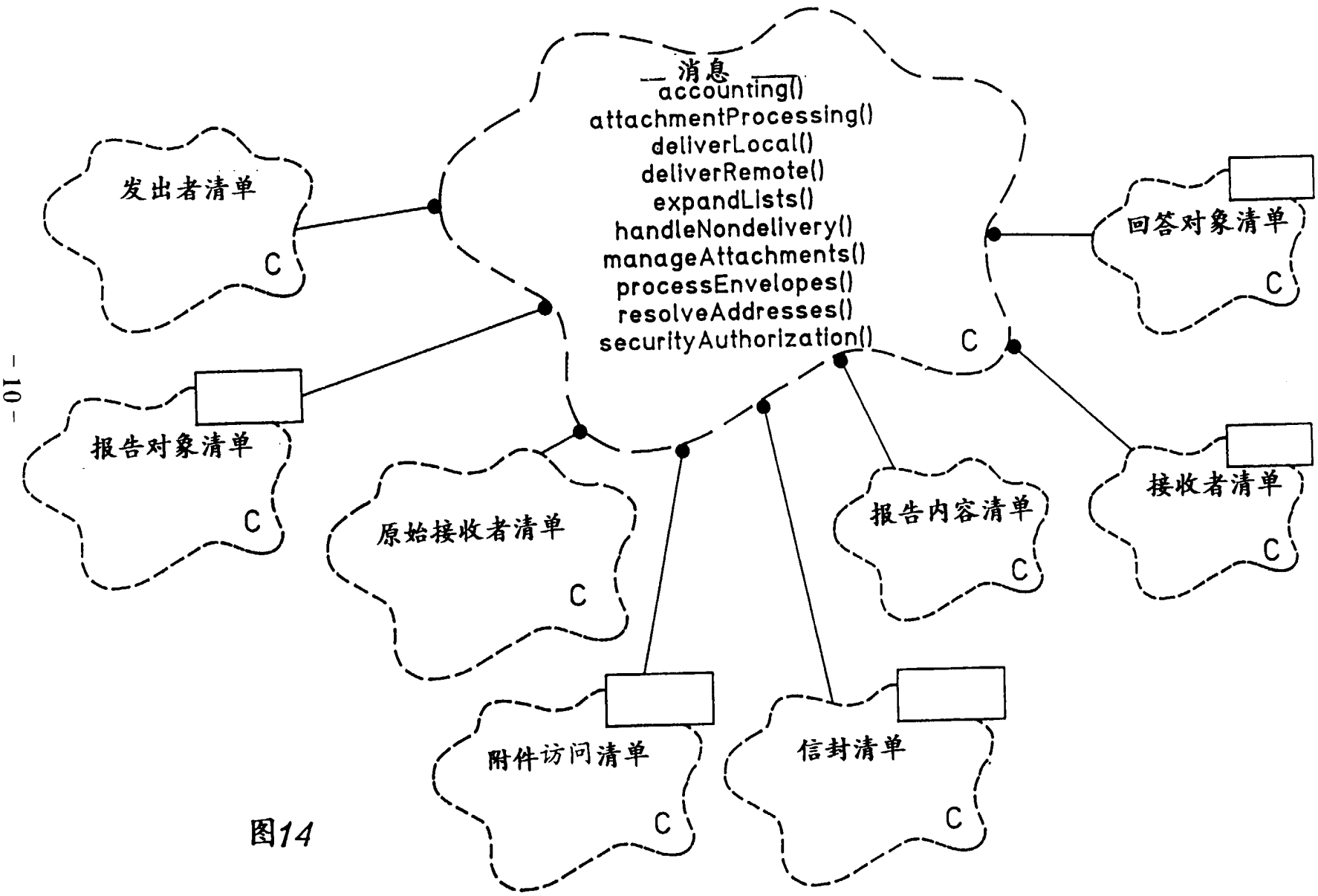

消息
accounting()
attachmentProcessing()
deliverLocal()
deliverRemote()
expandLists()
handleNondelivery()
manageAttachments()
processEnvelopes()
resolveAddresses()
securityAuthorization()
C
发出者清单
C
回答对象清单
C
报告对象清单
C
原始接收者清单
C
报告内容清单
C
接收者清单
C
附件访问清单
C
信封清单
C

图14

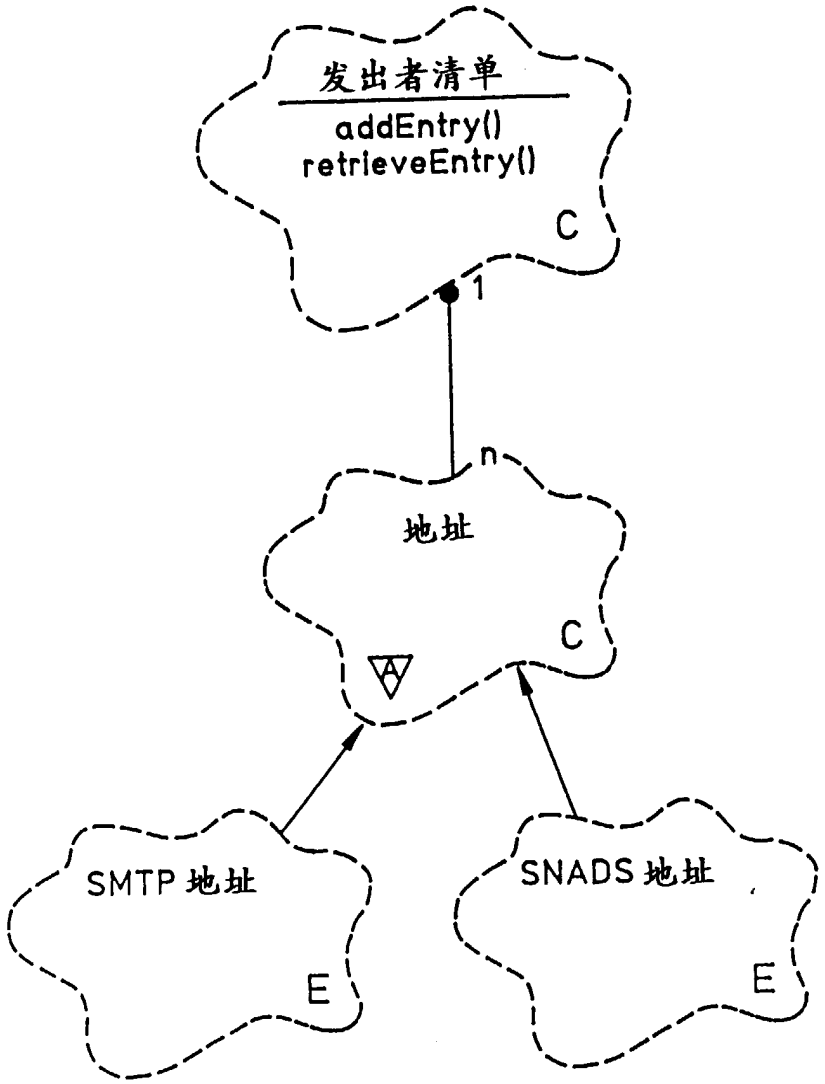

发出者清单
addEntry()
retrieveEntry()
C
1
n
地址
C
A
SMTP 地址
E
SNADS 地址
E

图 15

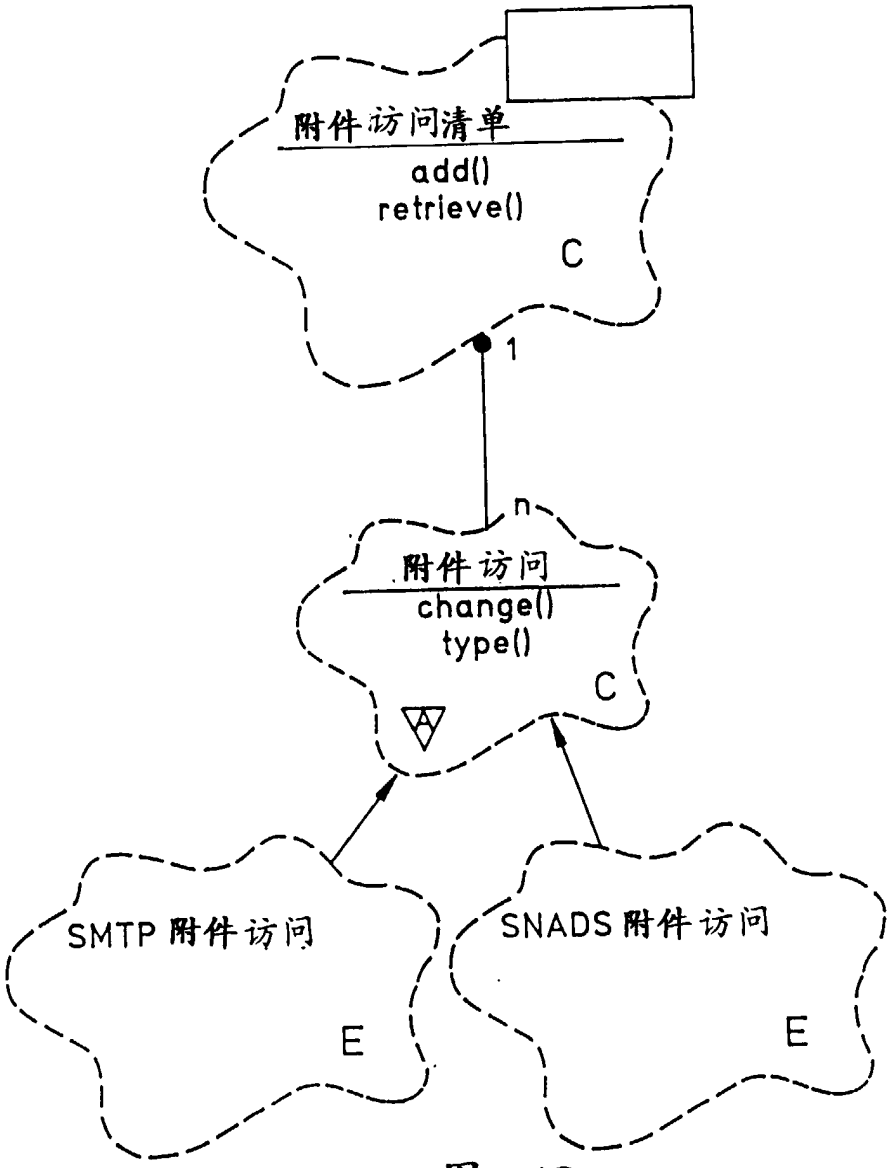

附件访问清单
add()
retrieve()
C
1
n
附件访问
change()
type()
C
A
SMTP 附件访问
E
SNADS 附件访问
E

图. 18

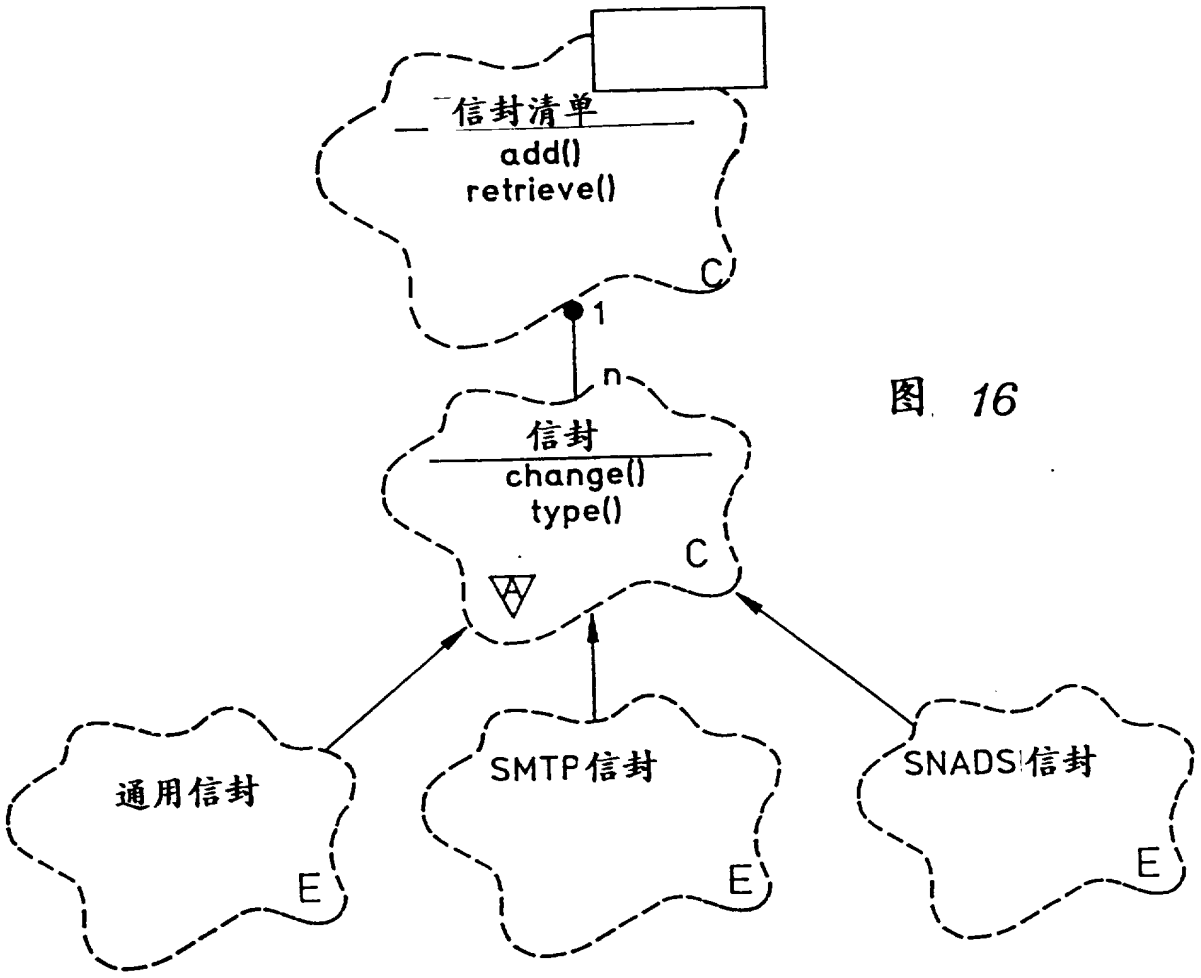

信封清单
add()
retrieve()
C
1
n
信封
change()
type()
C
A
通用信封
E
SMTP信封
E
SNADS信封
E

图 16

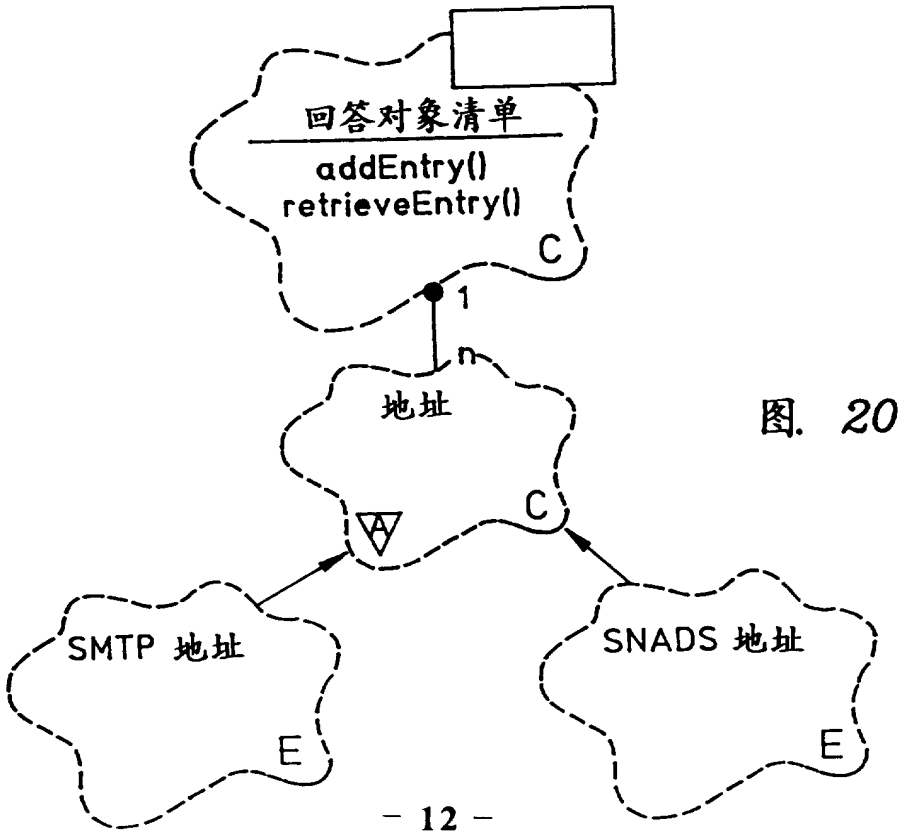

回答对象清单
addEntry()
retrieveEntry()
C
1
n
地址
C
A
SMTP 地址
E
SNADS 地址
E

图. 20

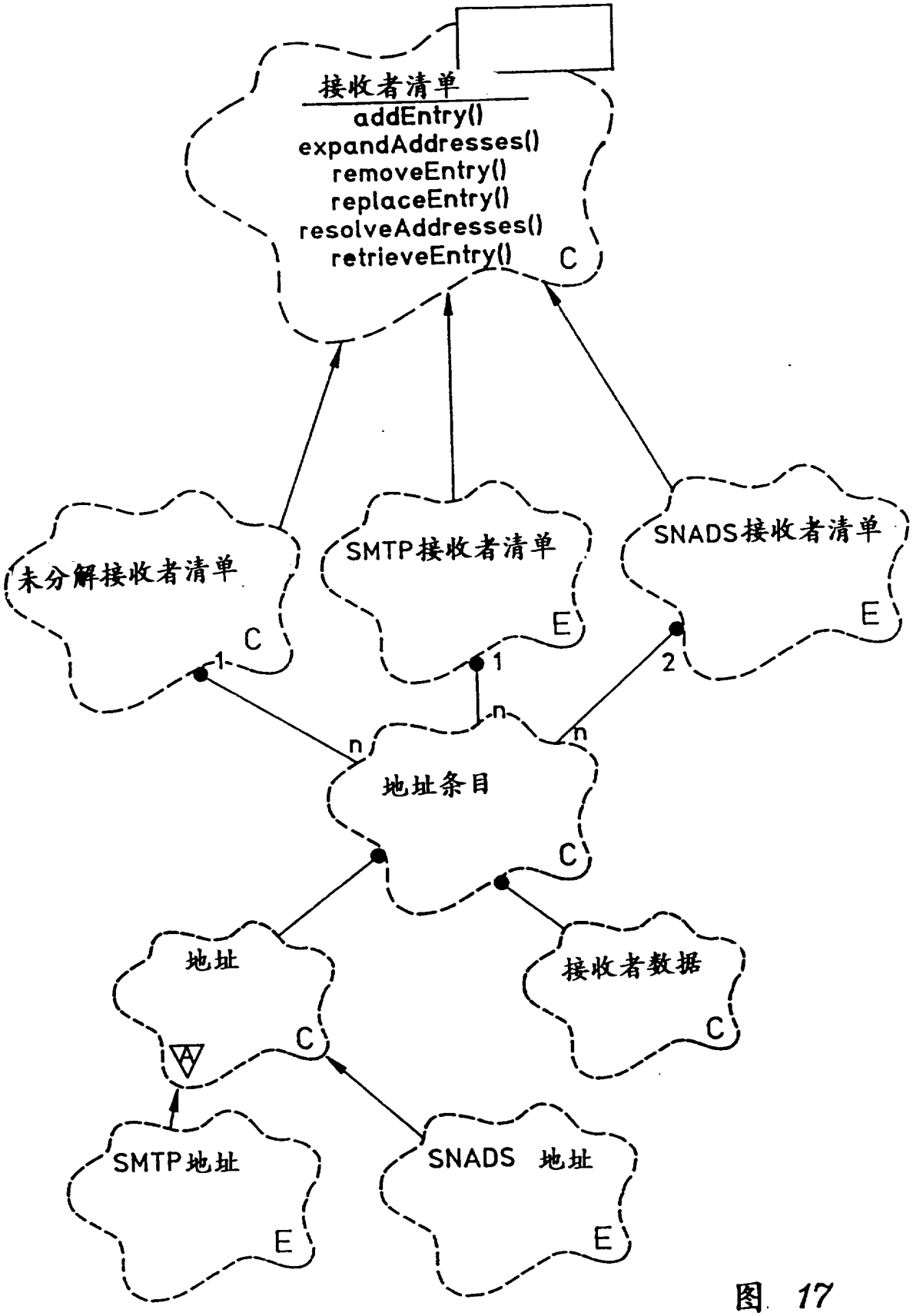

接收者清单
addEntry()
expandAddresses()
removeEntry()
replaceEntry()
resolveAddresses()
retrieveEntry()
C
未分解接收者清单
C
SMTP接收者清单
E
SNADS接收者清单
E
1
1
2
n
n
n
地址条目
C
地址
C
接收者数据
C
SMTP地址
E
SNADS 地址
E

图 17

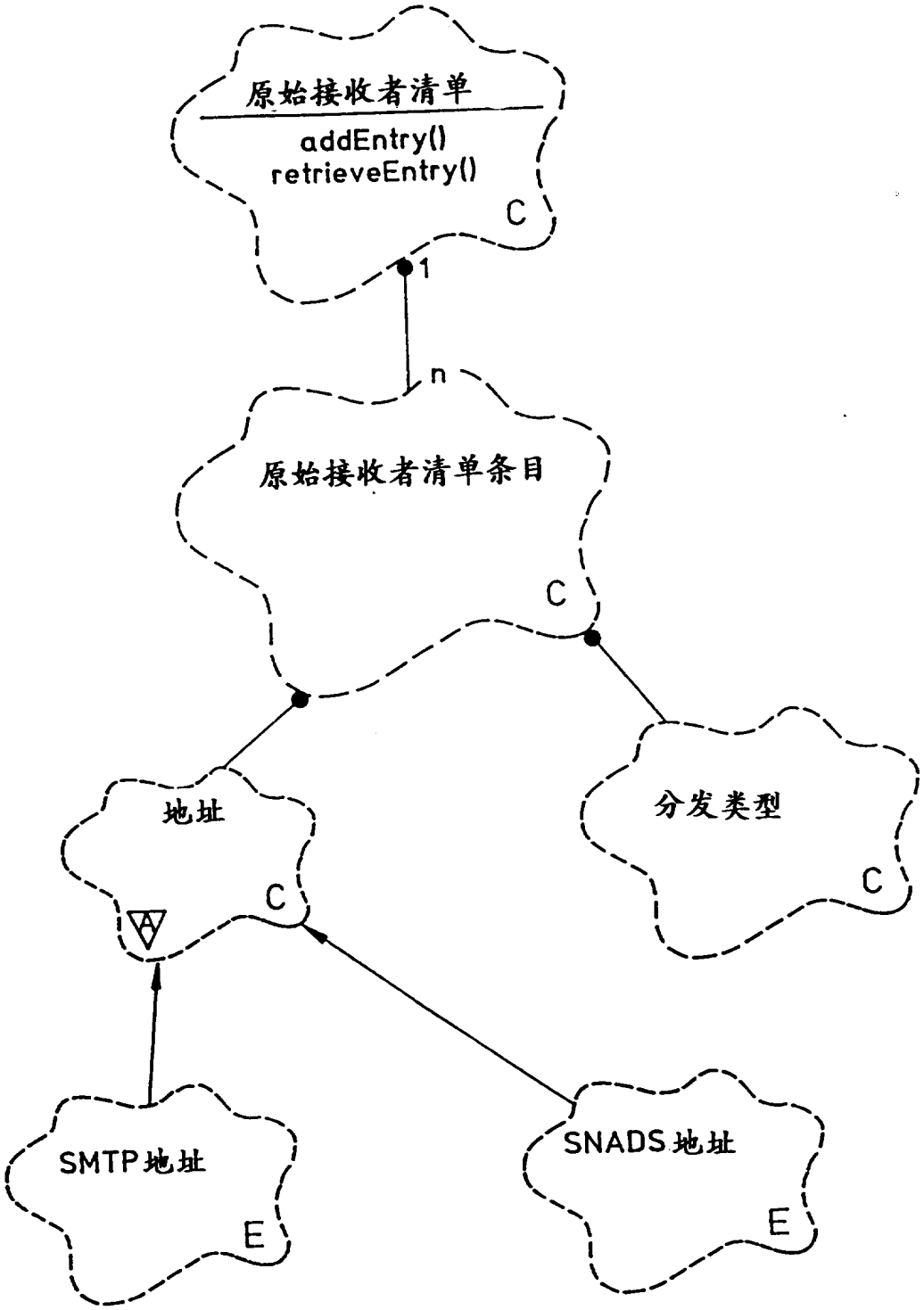

原始接收者清单
addEntry()
retrieveEntry()
C
1
n
原始接收者清单条目
C
地址
C
分发类型
C
SMTP地址
E
SNADS地址
E

图. 19

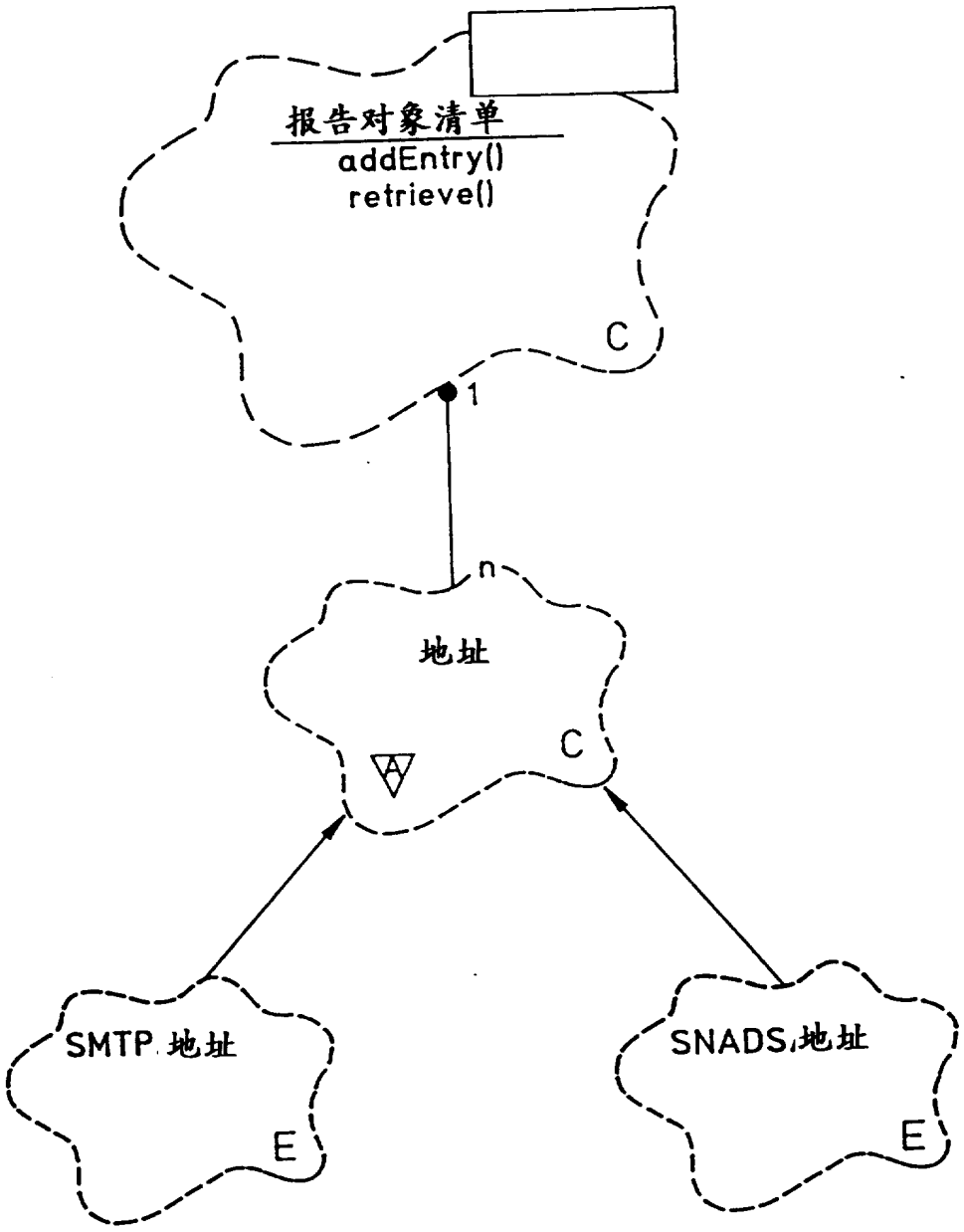


图 21

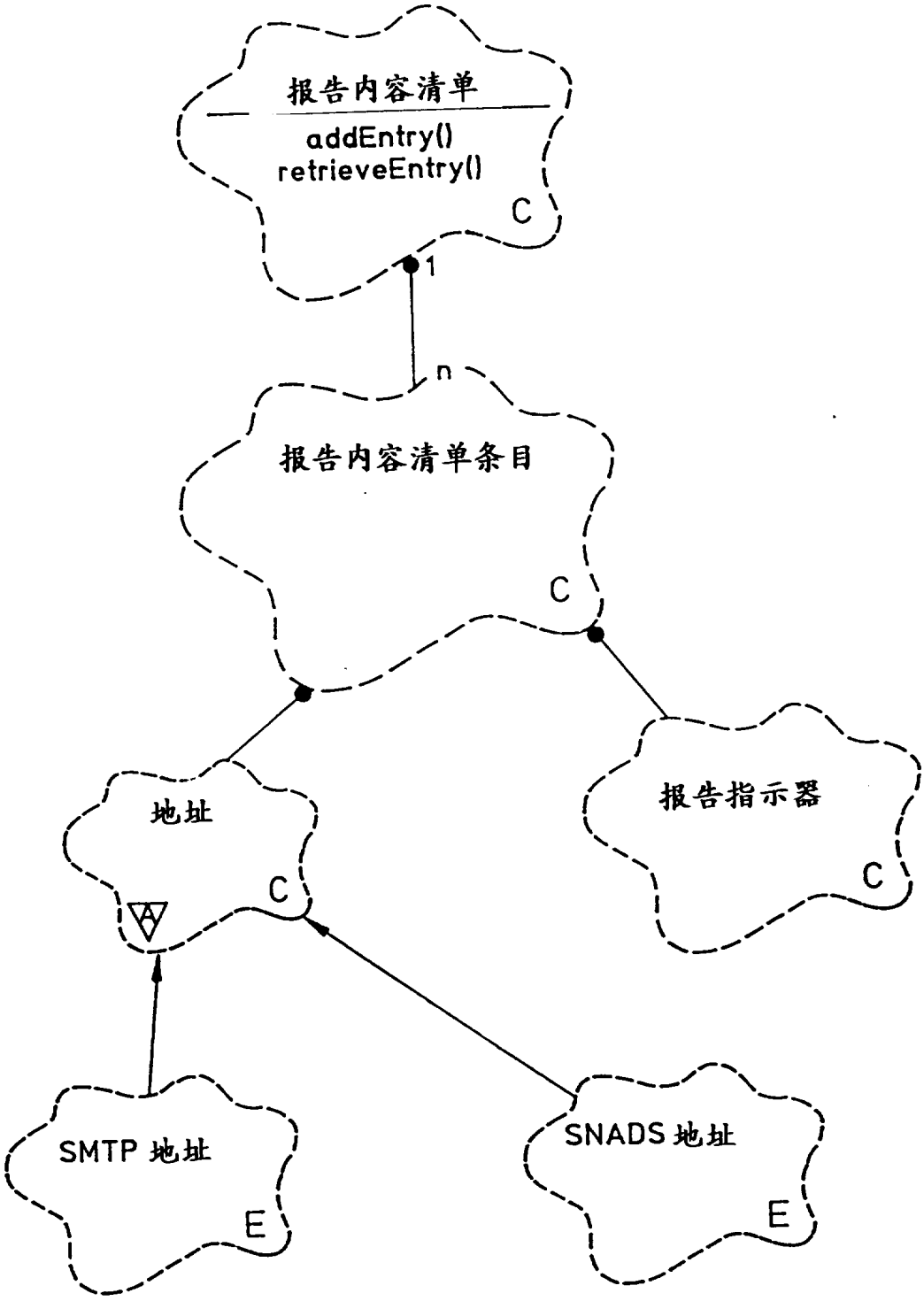

报告内容清单
addEntry()
retrieveEntry()
C
1
n
报告内容清单条目
C
地址
C
报告指示器
C
SMTP 地址
E
SNADS 地址
E

图 22

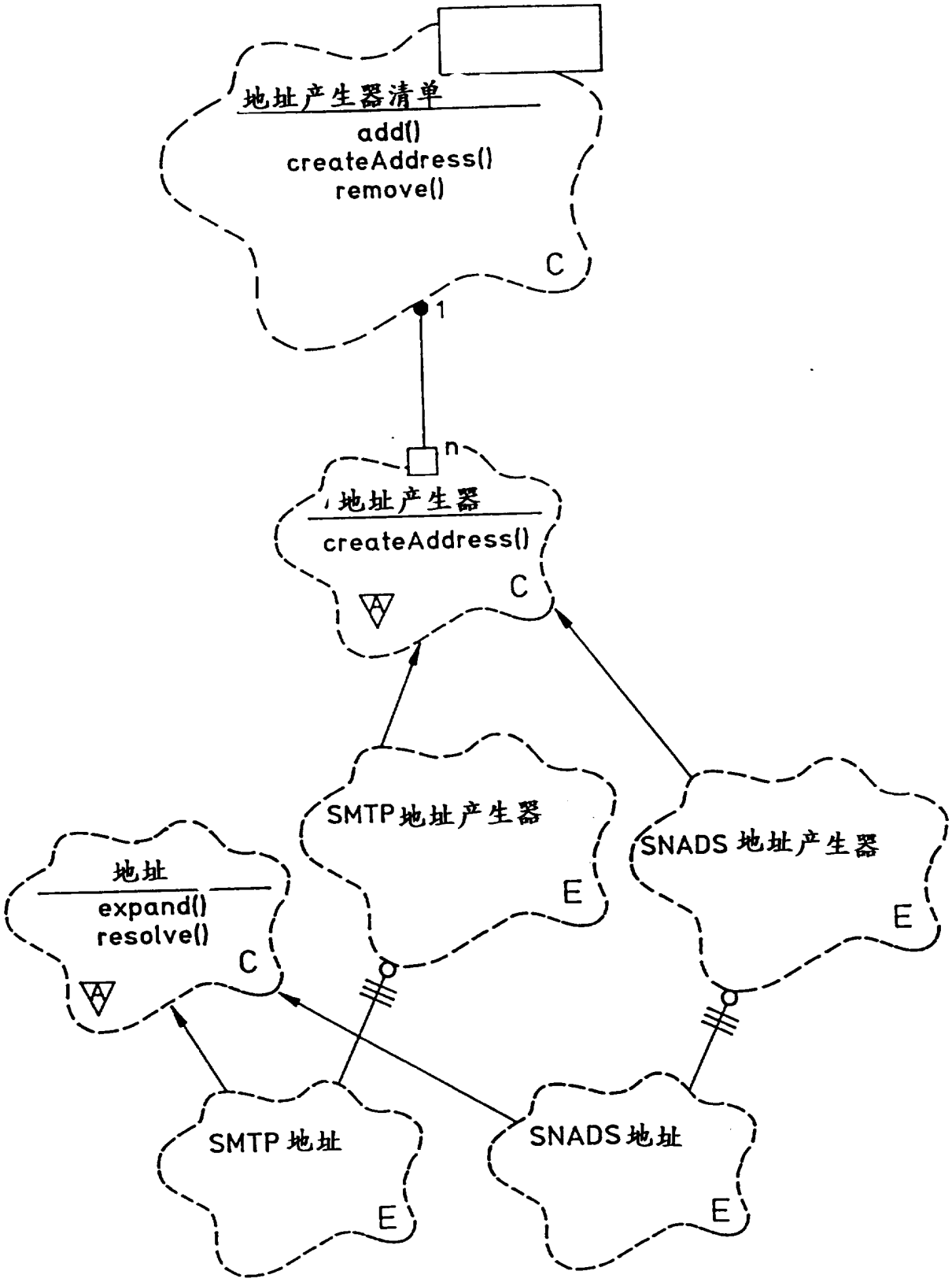

地址产生器清单
add()
createAddress()
remove()
C
1
n
地址产生器
createAddress()
C
SMTP地址产生器
E
SNADS 地址产生器
E
地址
expand()
resolve()
C
SMTP 地址
E
SNADS地址
E

图. 23

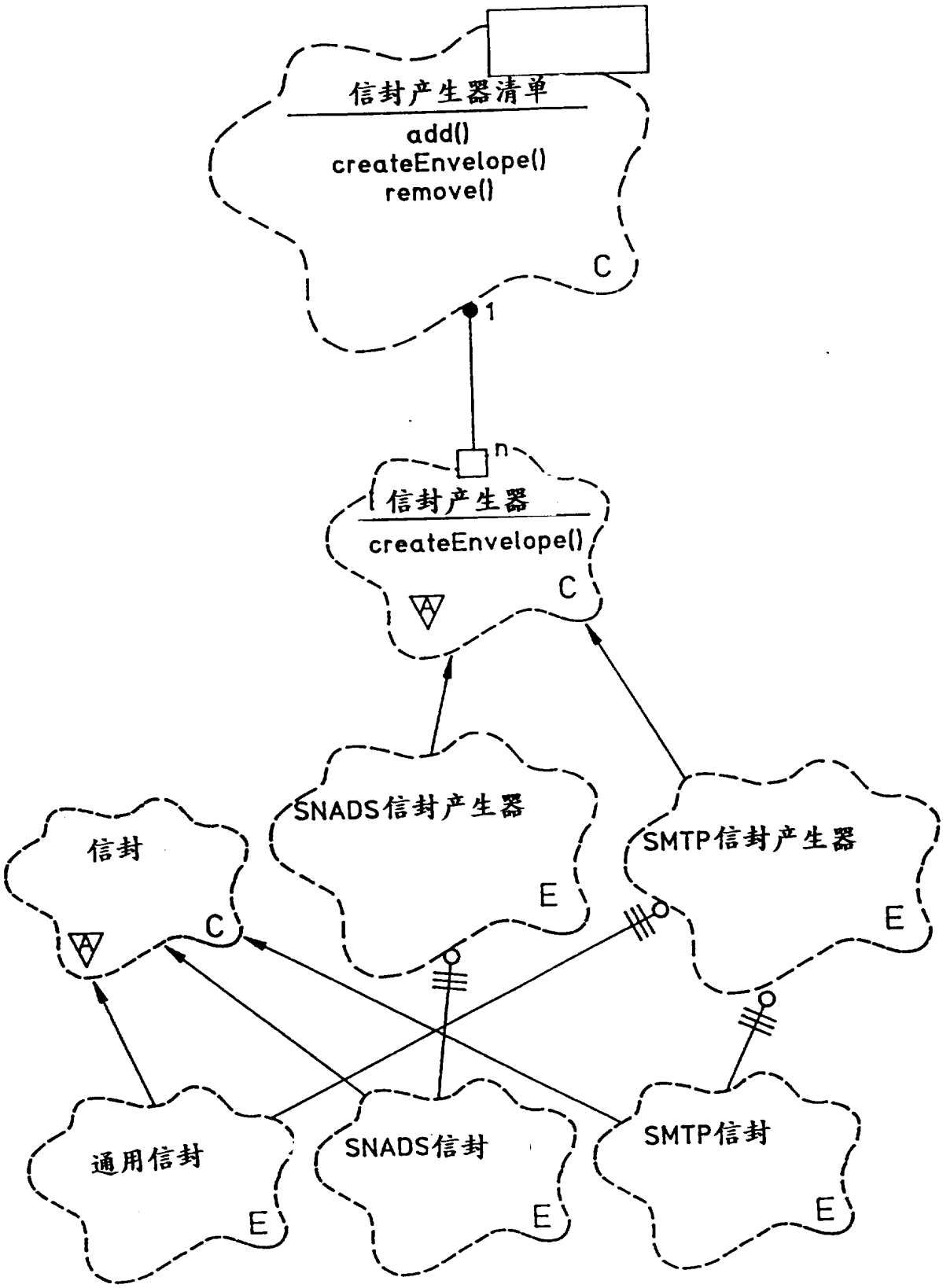

信封产生器清单
add()
createEnvelope()
remove()
C
1
n
信封产生器
createEnvelope()
C
SNADS信封产生器
E
SMTP信封产生器
E
信封
C
通用信封
E
SNADS信封
E
SMTP信封
E

图 24

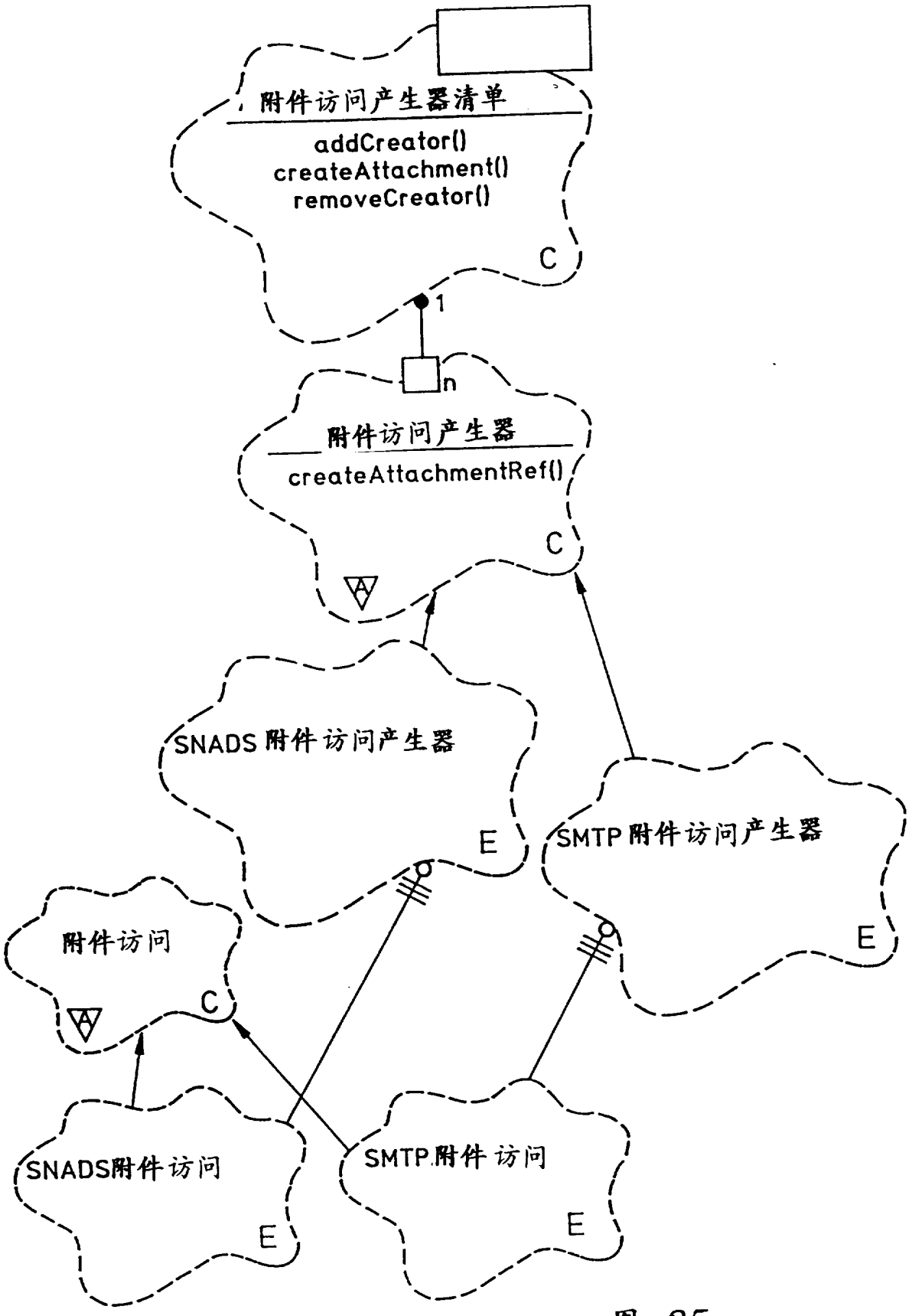

附件访问产生器清单
addCreator()
createAttachment()
removeCreator()
C
1
n
附件访问产生器
createAttachmentRef()
C
SNADS 附件访问产生器
E
SMTP 附件访问产生器
E
附件访问
C
SNADS附件访问
E
SMTP 附件访问
E

图 25

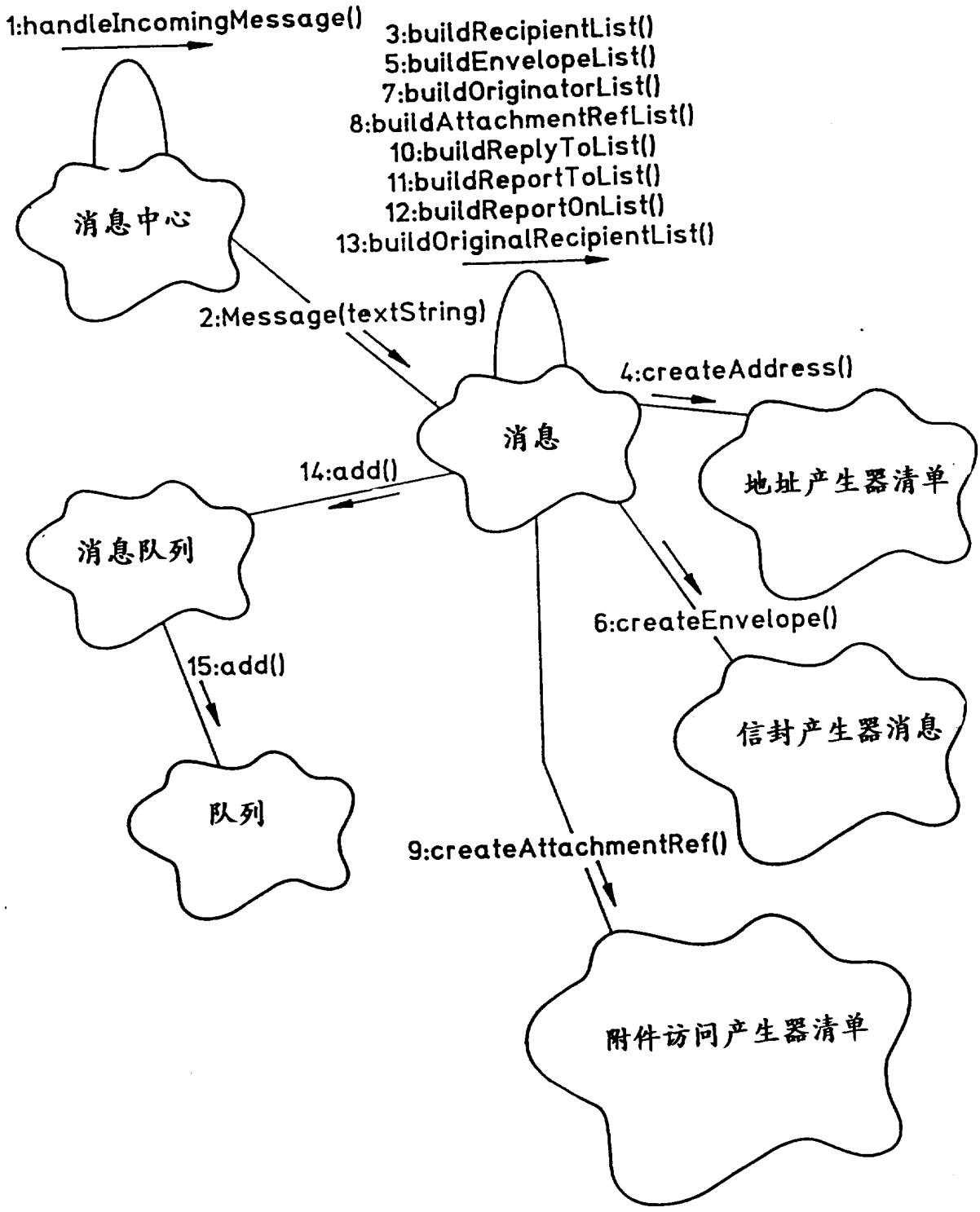

1:handleIncomingMessage()
3:buildRecipientList()
5:buildEnvelopeList()
7:buildOriginatorList()
8:buildAttachmentRefList()
10:buildReplyToList()
11:buildReportToList()
12:buildReportOnList()
13:buildOriginalRecipientList()
消息中心
2:Message(textString)
消息
4:createAddress()
地址产生器清单
14:add()
消息队列
6:createEnvelope()
15:add()
信封产生器消息
队列
9:createAttachmentRef()
附件访问产生器清单

图 26

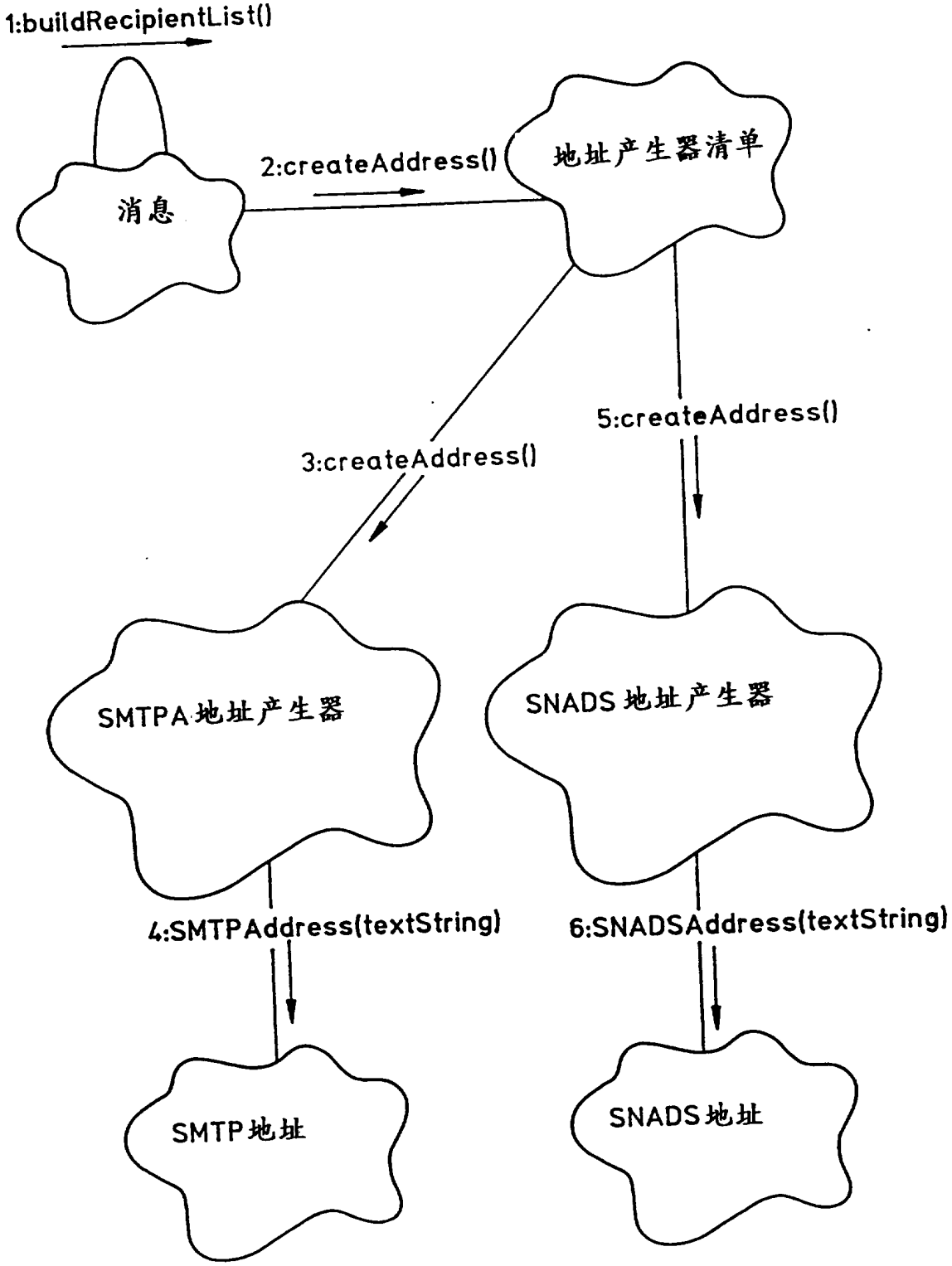

1:buildRecipientList()
消息
2:createAddress()
地址产生器清单
3:createAddress()
5:createAddress()
SMTPA地址产生器
SNADS地址产生器
4:SMTPAddress(textString)
6:SNADSAddress(textString)
SMTP地址
SNADS地址

图 27

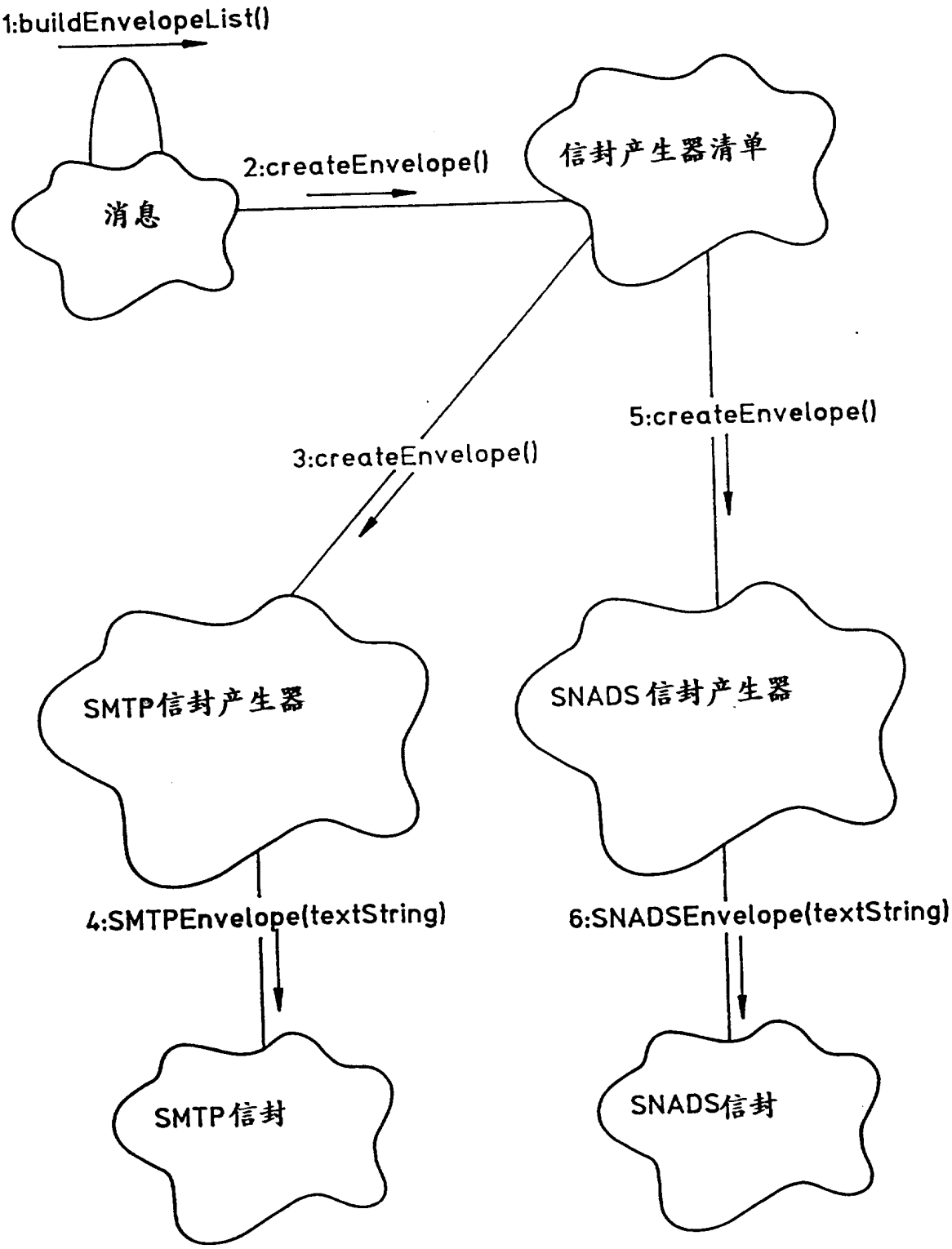

1:buildEnvelopeList()
消息
2:createEnvelope()
信封产生器清单
3:createEnvelope()
5:createEnvelope()
SMTP信封产生器
SNADS信封产生器
4:SMTPEnvelope(textString)
6:SNADSEnvelope(textString)
SMTP信封
SNADS信封

图 28

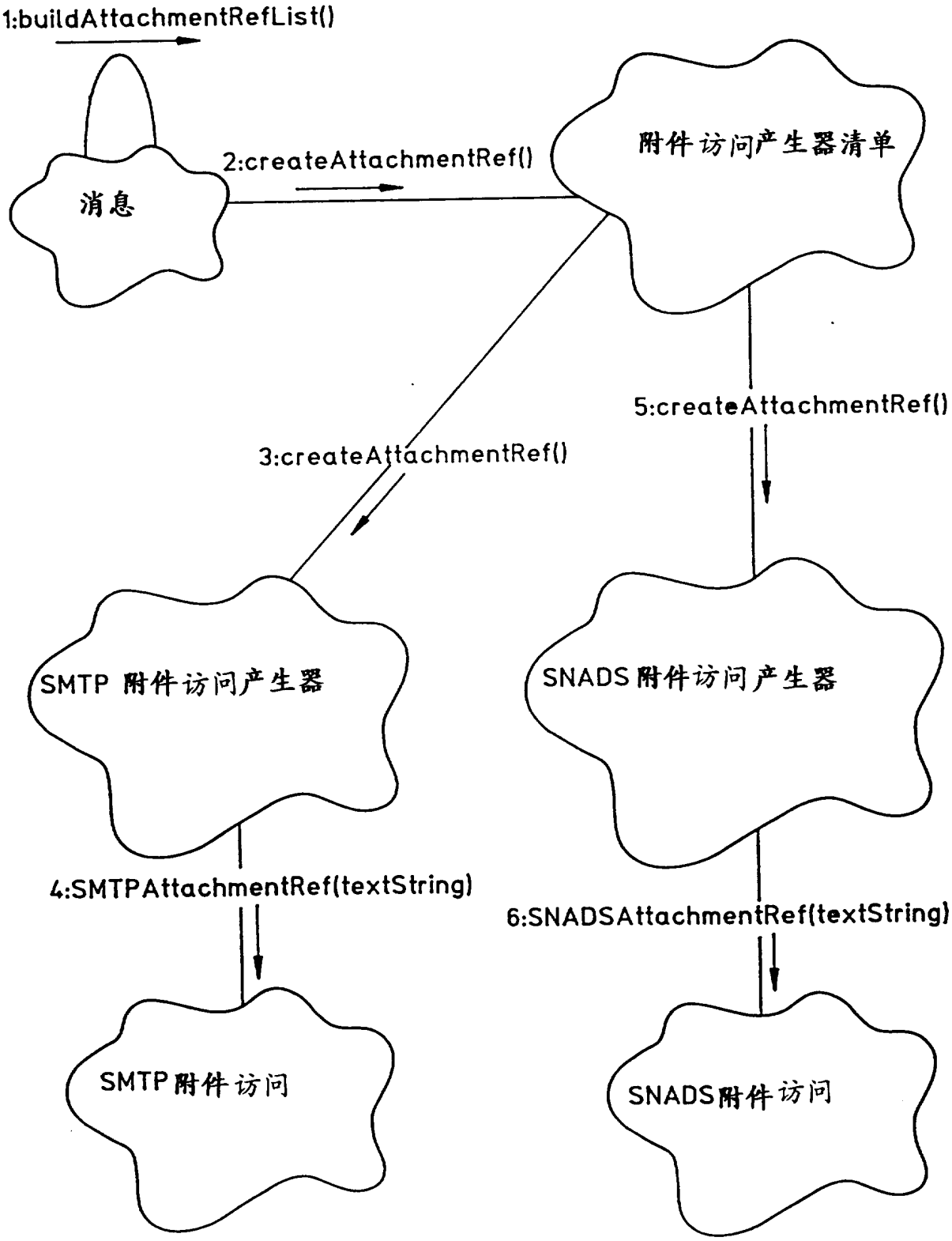

1:buildAttachmentRefList()
消息
2:createAttachmentRef()
附件访问产生器清单
3:createAttachmentRef()
5:createAttachmentRef()
SMTP 附件访问产生器
SNADS 附件访问产生器
4:SMTPAttachmentRef(textString)
6:SNADSAttachmentRef(textString)
SMTP附件访问
SNADS附件访问

图 29

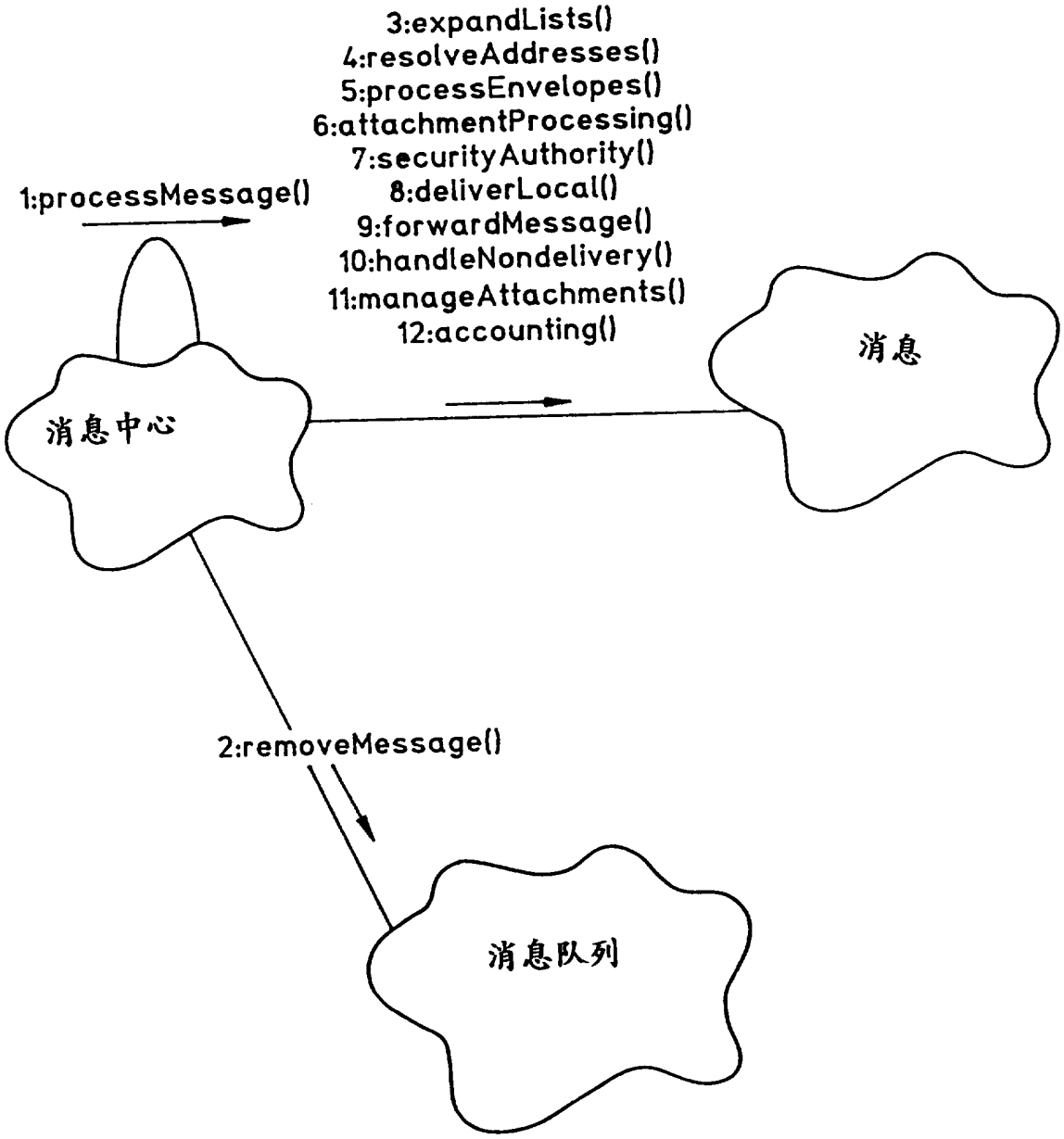

3:expandLists()
4:resolveAddresses()
5:processEnvelopes()
6:attachmentProcessing()
7:securityAuthority()
8:deliverLocal()
9:forwardMessage()
10:handleNondelivery()
11:manageAttachments()
12:accounting()
1:processMessage()
消息中心
消息
2:removeMessage()
消息队列

图 30

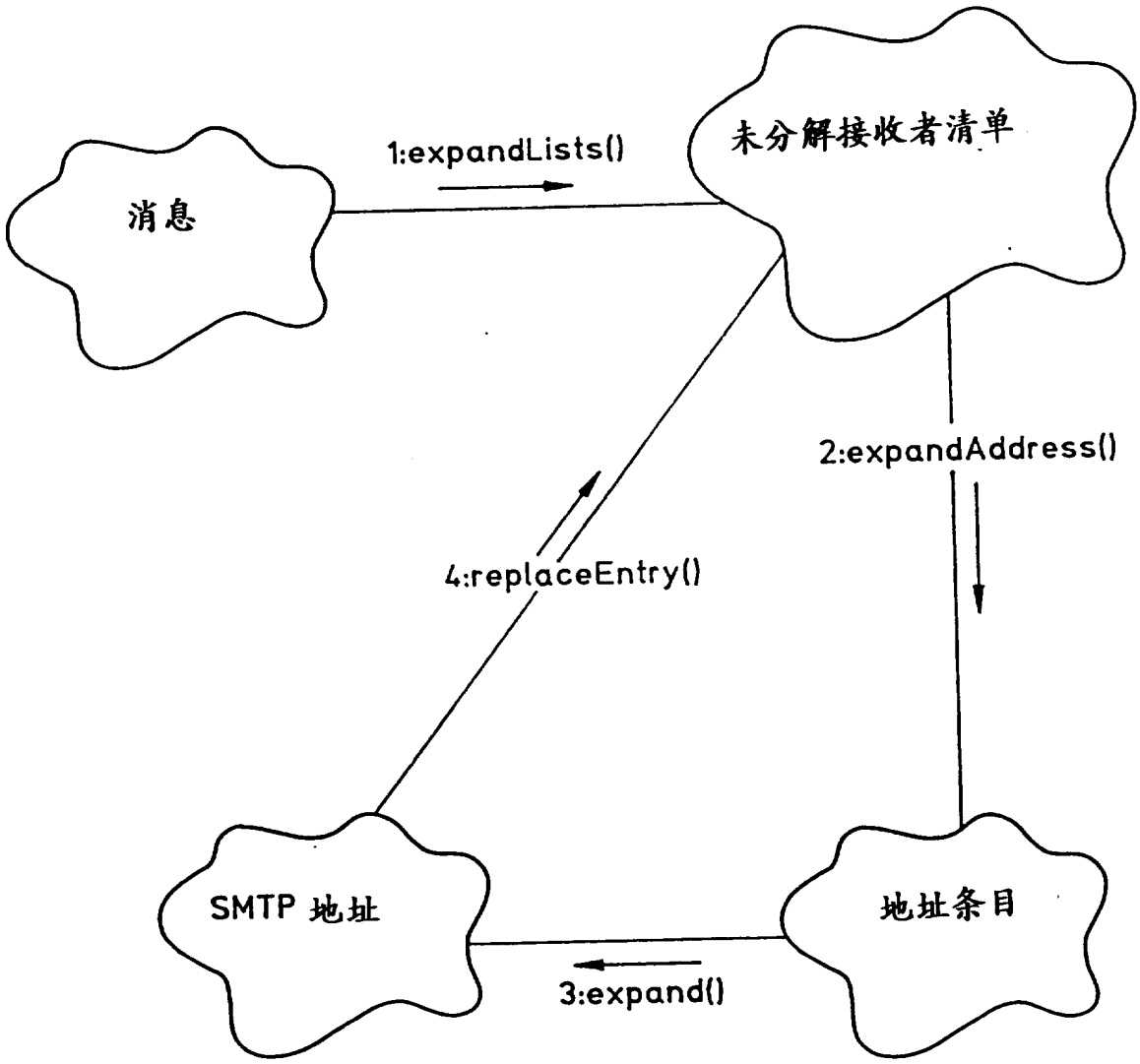

消息
1:expandLists()
未分解接收者清单
2:expandAddress()
4:replaceEntry()
SMTP 地址
3:expand()
地址条目

图 31

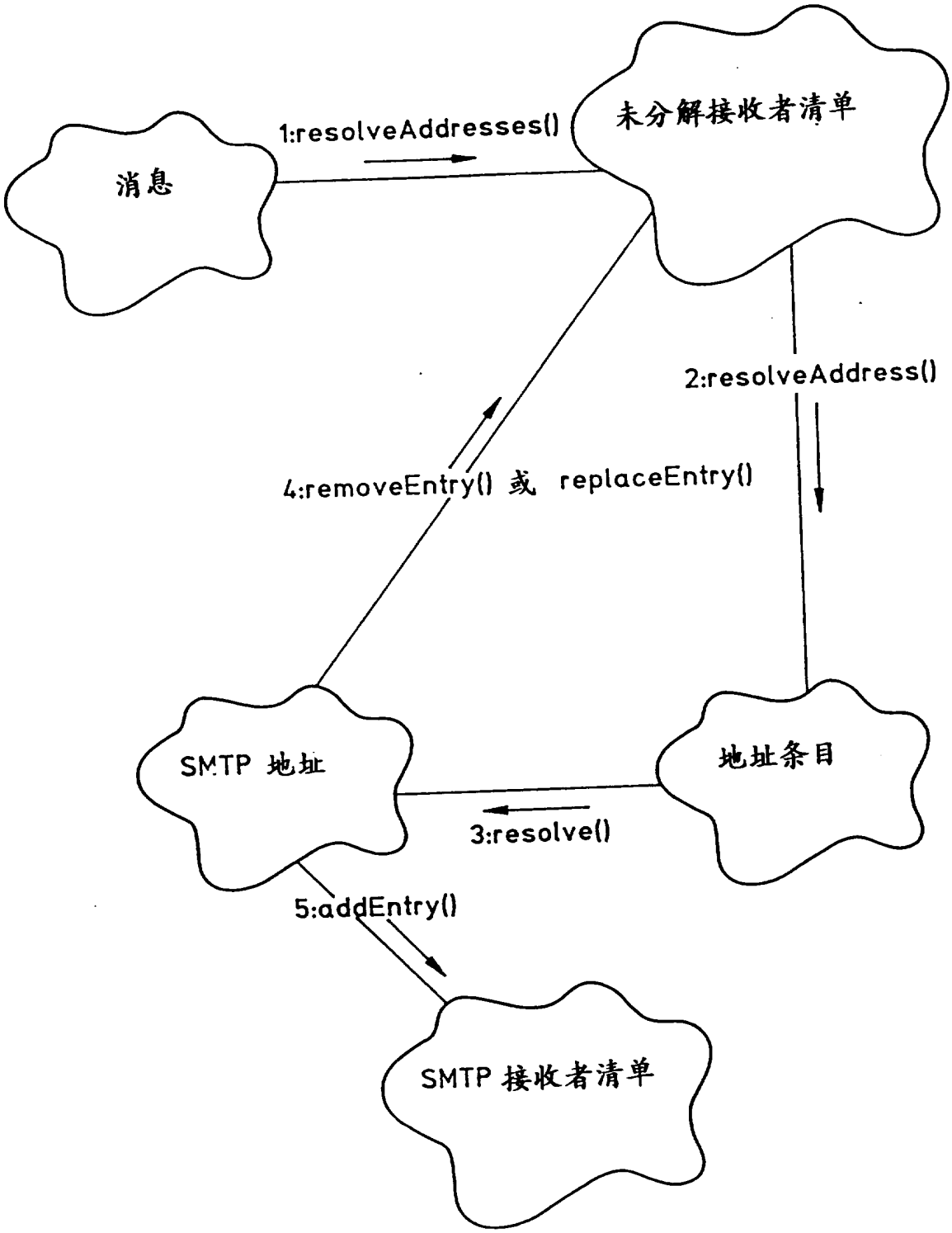

消息
1:resolveAddresses()
未分解接收者清单
2:resolveAddress()
4:removeEntry() 或 replaceEntry()
SMTP 地址
地址条目
3:resolve()
5:addEntry()
SMTP 接收者清单

图32

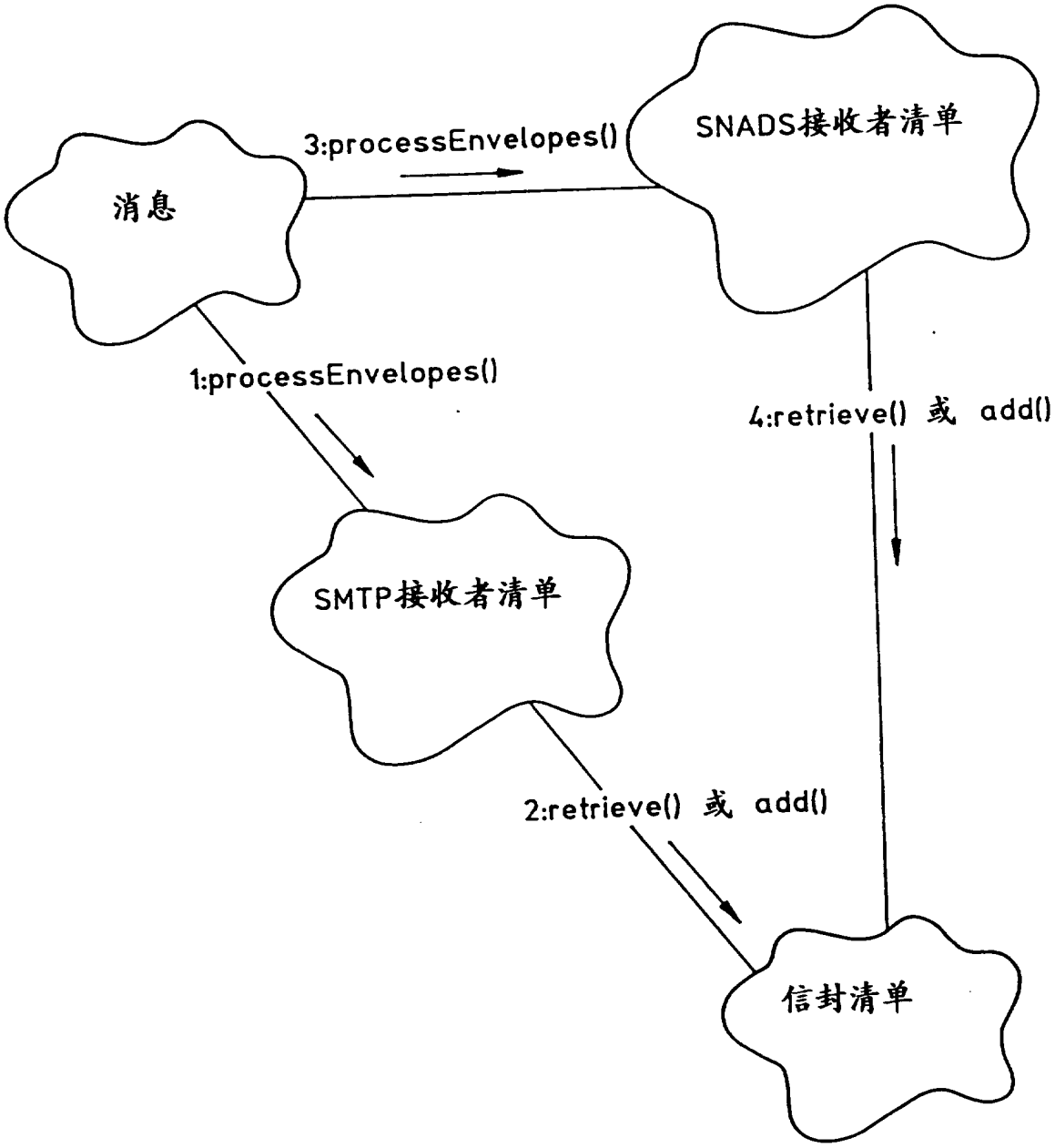

SNADS接收者清单
3:processEnvelopes()
消息
1:processEnvelopes()
4:retrieve() 或 add()
SMTP接收者清单
2:retrieve() 或 add()
信封清单

图 33

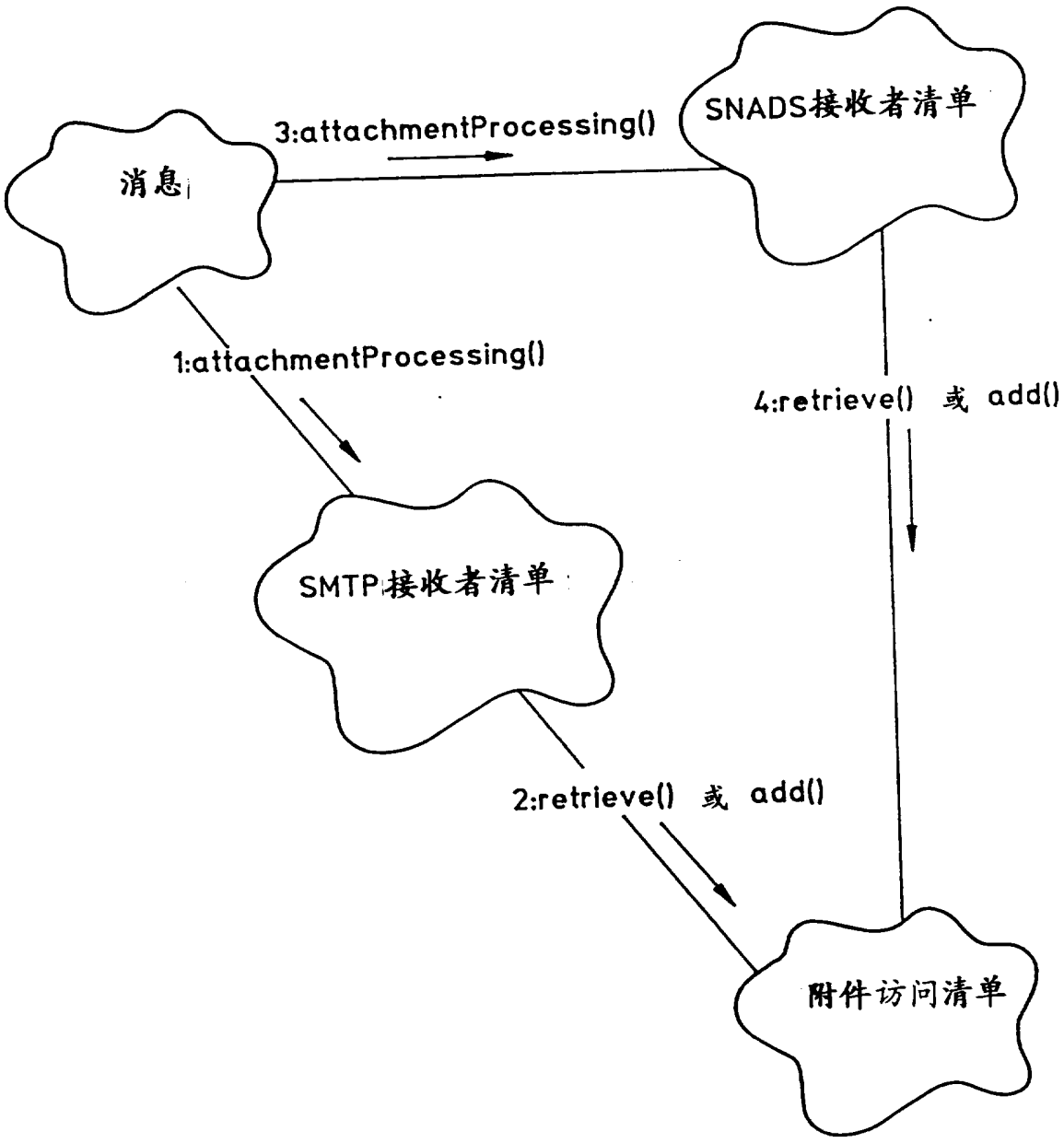

SNADS接收者清单
3:attachmentProcessing()
消息
1:attachmentProcessing()
4:retrieve() 或 add()
SMTP接收者清单
2:retrieve() 或 add()
附件访问清单

图 34

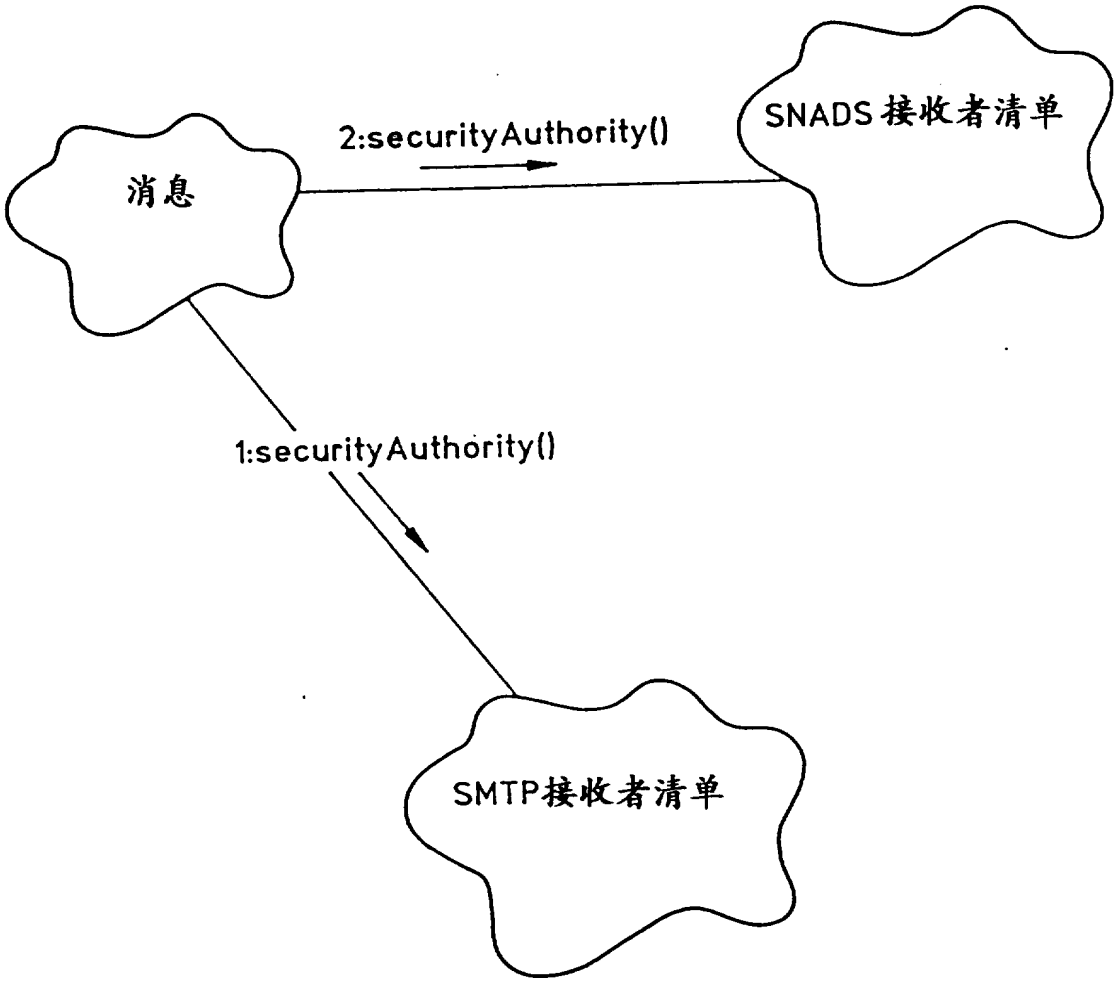

SNADS 接收者清单
2:securityAuthority()
消息
1:securityAuthority()
SMTP接收者清单

图 35

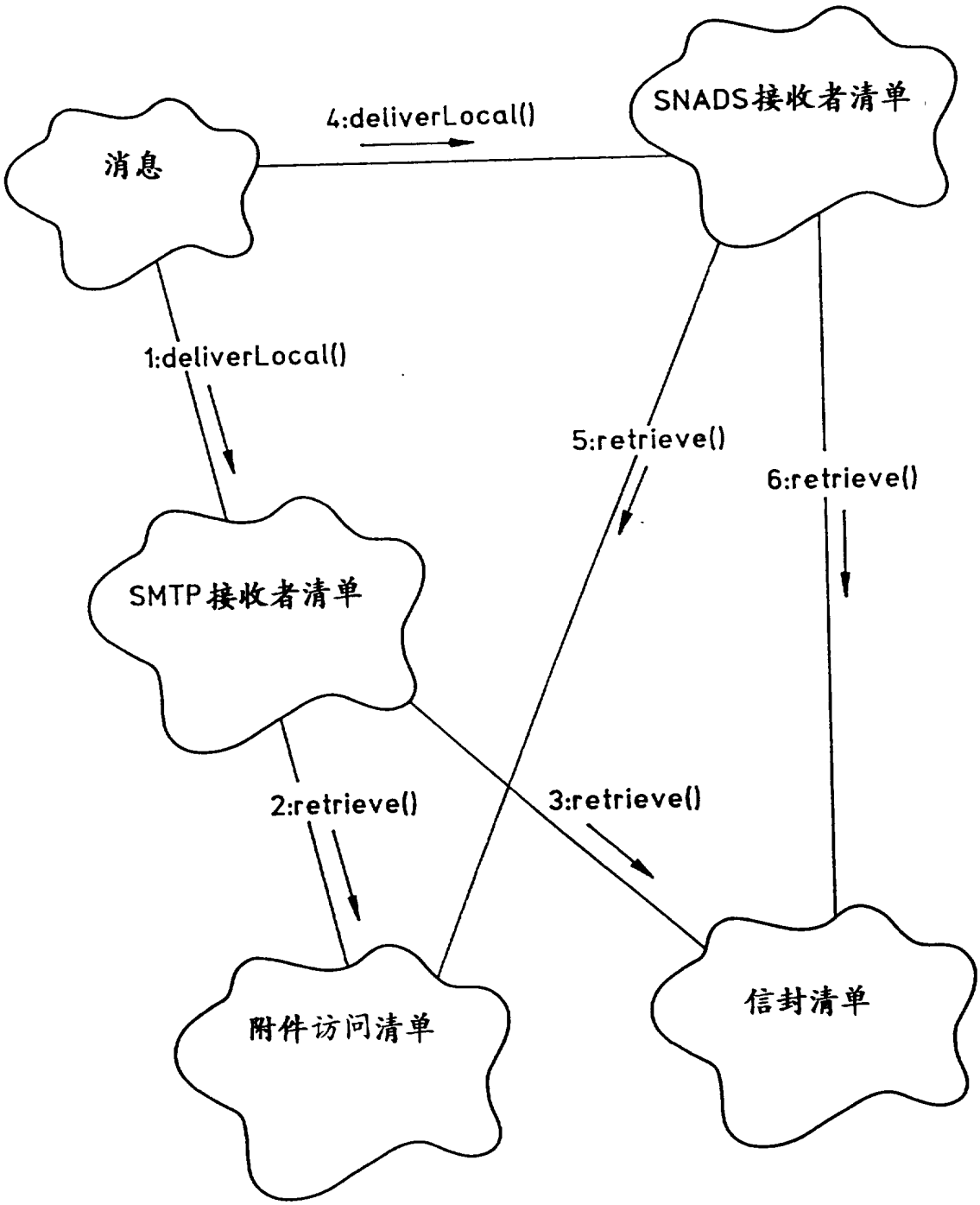

消息
4:deliverLocal()
SNADS接收者清单
1:deliverLocal()
5:retrieve()
6:retrieve()
SMTP接收者清单
2:retrieve()
3:retrieve()
附件访问清单
信封清单

图 36

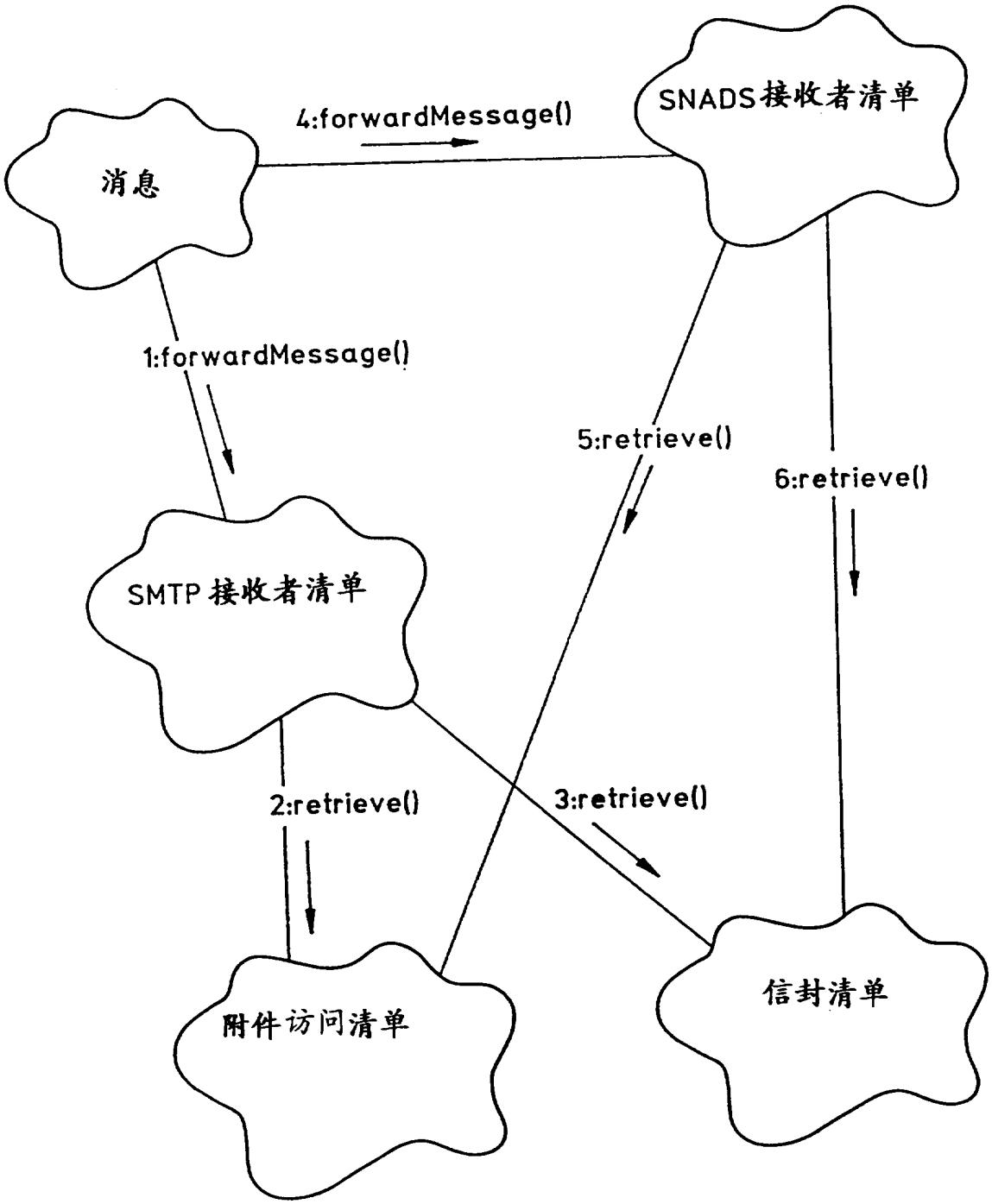

SNADS接收者清单
4:forwardMessage()
消息
1:forwardMessage()
5:retrieve()
6:retrieve()
SMTP 接收者清单
2:retrieve()
3:retrieve()
附件访问清单
信封清单

图 37

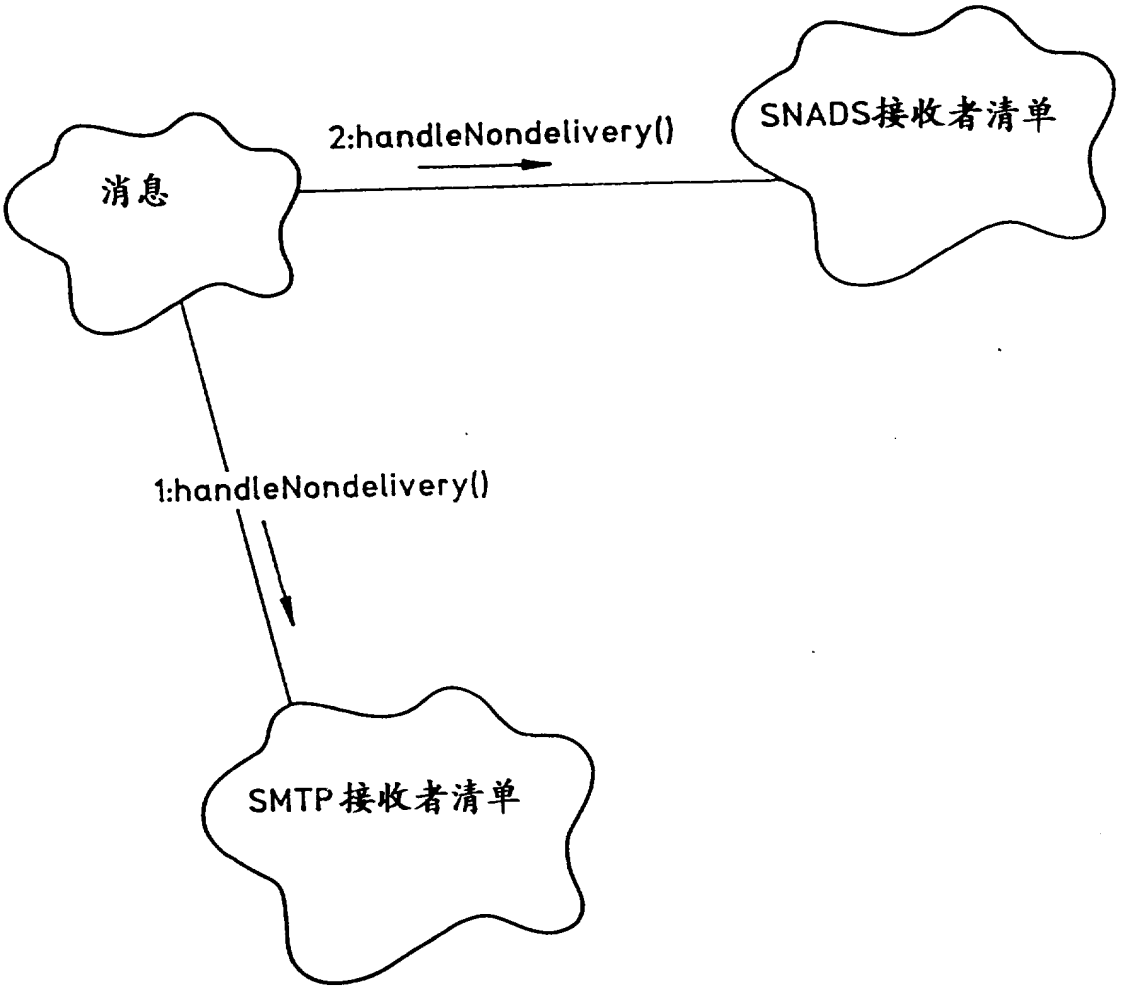

SNADS接收者清单
2:handleNondelivery()
消息
1:handleNondelivery()
SMTP接收者清单

图 38

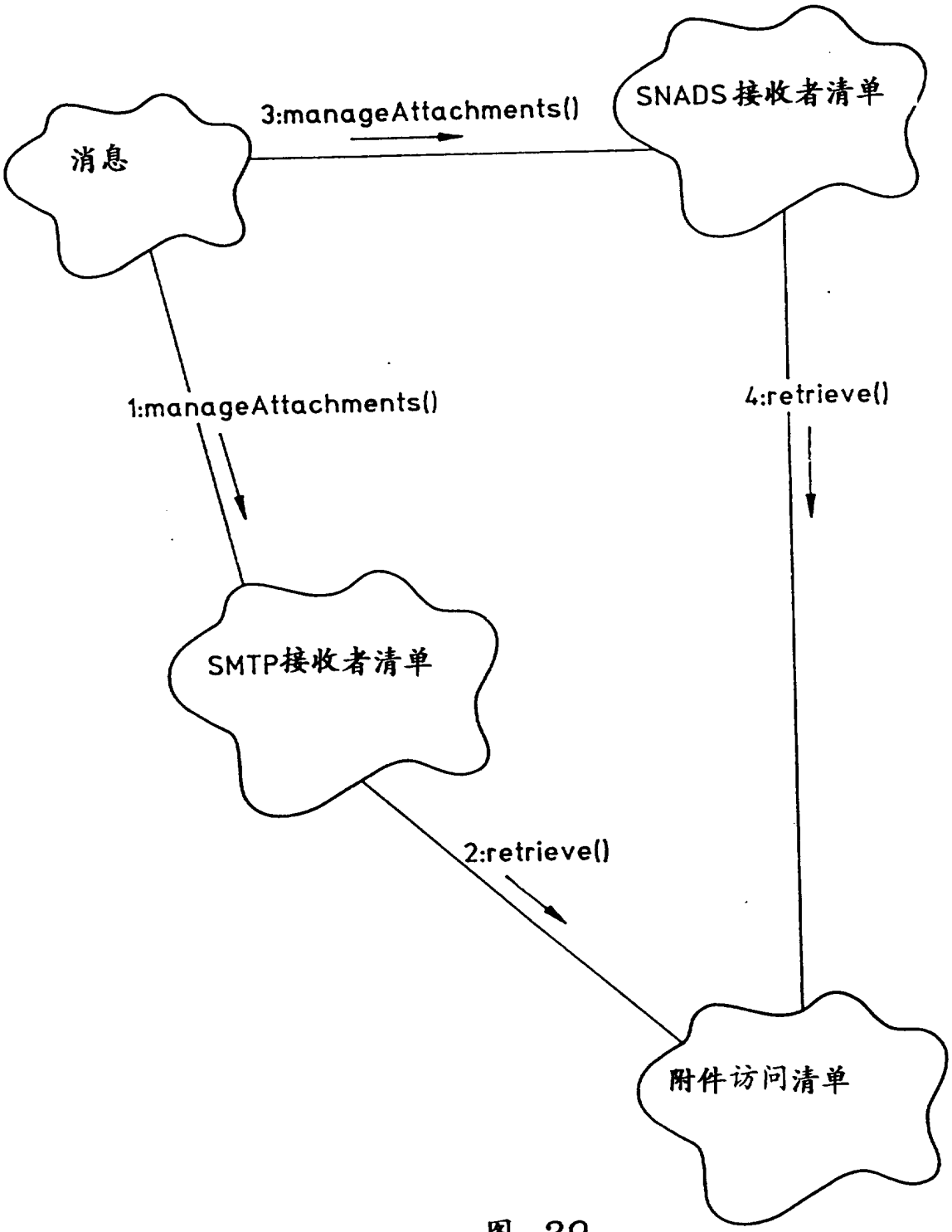

消息
3:manageAttachments()
SNADS接收者清单
1:manageAttachments()
4:retrieve()
SMTP接收者清单
2:retrieve()
附件访问清单

图 39

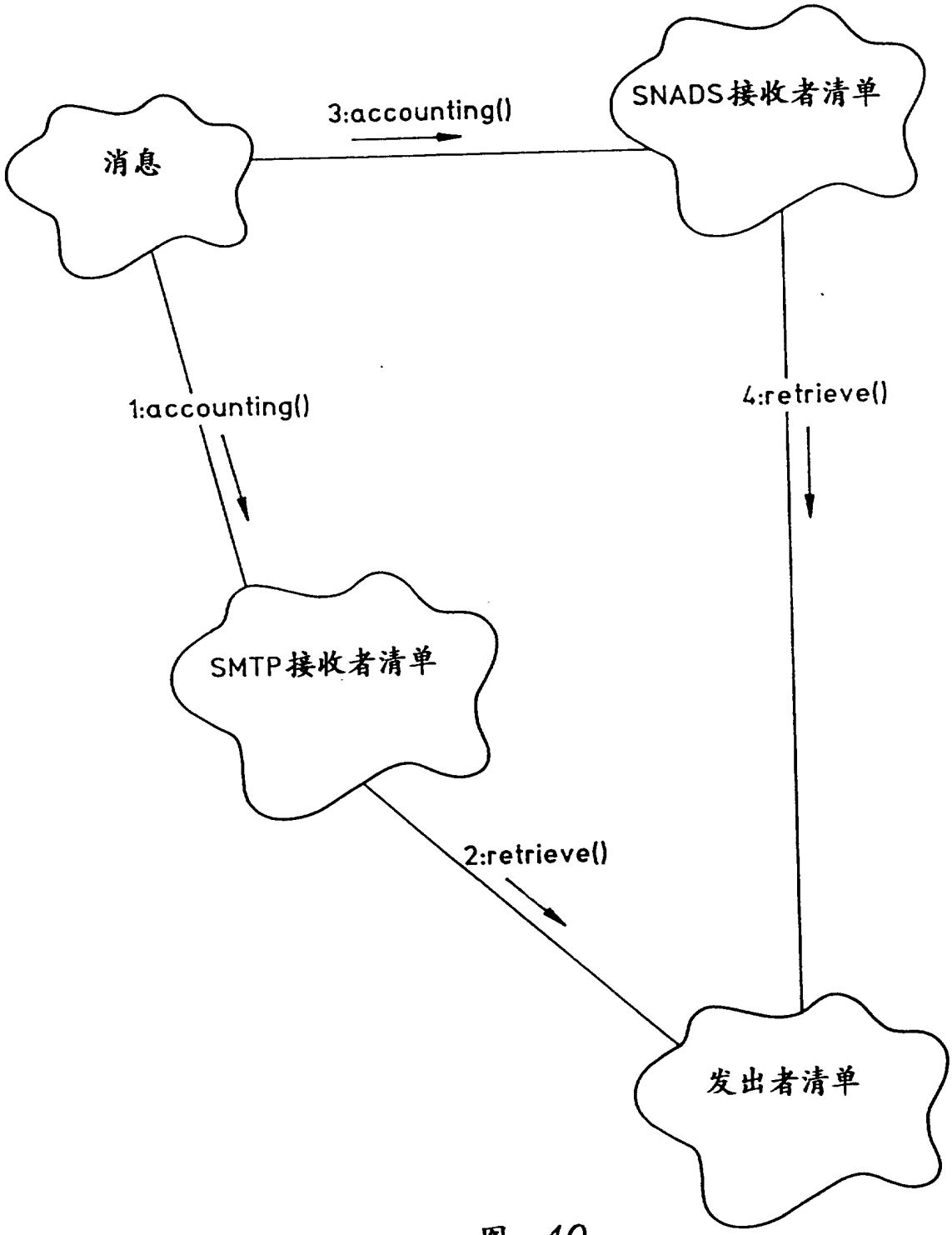

消息
3:accounting()
SNADS接收者清单
1:accounting()
4:retrieve()
SMTP接收者清单
2:retrieve()
发出者清单

图 40