[19]中华人民共和国专利局

[11] 公开号 CN 1088722A

[12] 发明专利申请公开说明书

[21]申请号 93118225.5

[51]Int.Cl[5]

[43]公开日 1994 年 6 月 29 日

H04L 29/02

[22]申请日 93.8.27

[30]优先权

[32]92.8.28 [33]SE[31]9202488

[32]93.2.5 [33]SE[31]9300363

[71]申请人 艾利森电话股份有限公司

地址 瑞典斯德哥尔摩

[72]发明人 P·-A·凯尔布兰 J·范滕堡

M·帕尔森 P·塞斯泰特

J·施韦德堡 B·塔达尔

A·吉兰达 S·施特龙堡

[74]专利代理机构 中国专利代理(香港)有限公司

代理人 马铁良 程天正

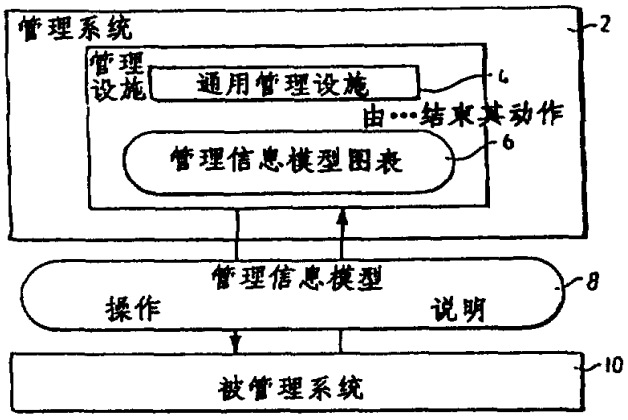
H04L 12/48 H04Q 3/42

说明书页数: 附图页数:

[54]发明名称 远程通讯及开放系统中的管理设施

[57]摘要

本发明是关于一具有至少一管理系统（2）和至少一被管理系统（10）的远程通讯或开放系统的管理网络。所述被管理系统包含有多个为管理系统作为资源的数据映象形式的被管理时象进行管理的物理的和/或逻辑的资源。管理系统为其针对被管理系统的操作采用一包括有所有适配于该管理系统的操作模式的被管理对象的说明的被管理系统的管理信息模型（8）。此管理系统除被管理系统（10）外包括有一通用管理设施（4）和该管理信息模型的表达图表（6）。该通用管理设施在操作期间的性能即由这模型图表决定。



权利要求书

1．一具有至少一个管理系统和至少一个被管理系统的远程通讯或开放系统的管理网络，其中所述被管理系统包含有多个为管理系统以资源的数据映象形式作为被管理对象进行管理的物理和／或逻辑资源，以及其中的管理系统为其针对被管理系统的操作而利用包括有全部与该管理系统的操作模式相适应的被管理对象的说明的一个被管理系统的信息模型，该管理网络的特征在于：

除被管理系统外，还包括有一通用管理设施和一该管理信息模型的表达图表，该通用管理设施在操作期间的性能即由此模型图表来决定。

2．按照权利要求1的管理网络，其特征是该管理信息模型的图表被引入到被管理系统中并能为该通用管理设施所利用。

3．按照权利要求1或2的管理网络，其特征是管理信息模型的技术规范作成该管理信息模型的可判断图表的形式，所述模型定义：

——从管理观点出发被管理系统所能具有的有意义的状态；

——能为被管理系统所接收的操作；

——能针对处于某一特定状态中的被管理系统的操作；以及最后

——在经受某一特定操作对被管理系统所达到的状态，

其中所说的可判断的图表模型是指上述定义能以机器可解释的语言来表达，以使上述特性由该技术规范确定。

4．按照权利要求3的管理网络，其特征是一组在所述被管理系统处于某种特定状态下才允许的特殊操作，此所述特定状态是由构成该类被管理对象，或一组这些类别的一逻辑部分的先决条件和／或结

束条件所指定的，其中，先决条件状态为被管理系统要接收一操作所必须呈现的状态，而结束条件状态则为在一更新事务之后被管理系统应呈现的状态。

5．按照权利要求4的管理网络，其特征是结束条件被规定为它们由下述两种对策之一来满足，即：

——管理系统对被管理系统的更新要使得能保持此结束条件，在此情况下管理设施负责满足此条件，而如果此管理设施忽略了这一职责，被管理系统就拒绝此更新事务，或者

——被管理系统借助自动进行为满足该结束条件所必须的再次更新来维持所设定的限制。

6．按照权利要求5的管理网络，其特征是结束条件的约束可被保持到调整和生成操作。

7．按照权利要求4－6中任一个的管理网络，其特征是结束条件在一被管理系统的管理信息模型中加以规定，此结束条件说明被管理系统中必须不得违背的固定的一致性限制，此结束条件可应用于被管理系统中的数据库，与存贮在该数据库中的对象实例及它们的属性值相关连。

8．按照权利要求4－7中任一个的管理网络，其特征是结束条件可以表明

——属性值之间的依从关系；

——属性和数据库关系的组成数量，即对一属性的取值数量的限定；

——对一对象型式的实例数量的限制。

9．按照权利要求4－8中任一个的管理网络，其特征是先决条件说明被管理系统的一数据库状态的限制，此限制在该数据库的始端

中的一特殊操作的事务之前必须予以完成。

１０．按照权利要求５－９中任一个的管理网络，其特征是

——在第一种对策时，事务中的一致性控制在该事务被提交之前实现，只有在不违反任何结束条件时该事务才进行到底，否则即重新运行；

——在第二种对策时，在该事务被提交之前其中要采取自动矫正措施，例如在一特定属性被更新时。

１１．按照权利要求１－１０中任一个的管理网络，其特征是为了设置能够包含在一被管理系统的被管理对象中的可重复使用的功能性部件，可以设置具有特定功能性的部件以生成所述部件不知道的一些部件能接收的事件。

１２．按照权利要求１１的管理网络，其特征是为了这种设定，包含多种属性的部件，在这些属性值改变时产生诸多事件。

１３．按照前述权利要求中任一的管理网络，为了在被管理系统的一子系统中实现一被管理对象，这里说的子系统是指一被管理系统的每一包含有一个或多个被管理对象以及被管理对象之间的数据库关系的部分，其特征是在相对于其它子系统未经协调的子系统中这样来实现被管理对象，即它可以连接并传送信息到其他子系统中的其他对象，而无需知道该其他子系统中的对象的形式。

１４．按照权利要求１３的管理网络，其特征是第一对象　被设置为与一抽象对象相协作，此抽象对象定义一由未实施的方法所构成的接口并能为该第一对象所访问，而在以后设置一所述第一对象不知道的第二对象时，此第二对象并企图能与第一对象协作，该第二对象即承袭该抽象对象并实现所承袭的方法，以便使得第一对象在与第二

对象相协作时将其看作为所述抽象对象。

15．按照权利要求14的管理网络，其特征是在调用该抽象对象的接口中定义的方法时，此调用将依靠随后的连接交由实际对象来实现。

16．按照权利要求4和15的管理网络，其特征是结束条件是针对一组类别指定的，这些类别可以属于不同的子系统。

17．按照权利要求14～16中任一个的管理网络，其特征是依靠各相关子系统间的动态链接可以避免在装载所述第二对象时重新装载被管理系统中所述第一对象。

18．按照权利要求14－17中的任一个的管理网络，其特征是所述第一和第二对象分别处于第一和第二子系统中。

19．按照权利要求18的管理网络，其特征是借助动态链接的应用使得装载所述第二子系统的被管理对象时可能无须再装载所述第一子系统。

20．按照权利要求1－10中任一个的管理网络，其特征是通用管理设施能根据模型图表确定针对一被管理系统可以执行哪些操作，并能确定为达到所希望的状态需要哪些操作。

21．按照权利要求1、2、3或20中的任一个的管理网络，其特征是管理信息模型的图表被用来分别解释一被管理系统的数据结构和将数据结构装入组装入一被管理系统。

22．按照权利要求1、2、3、20或21中任一个的管理网络，其特征是管理信息模型及系统中被管理对象的实现是由采用机器可解释的语言的同一技术规范得到的，其中采用这一语言的多个被管理对象的技术规范共同组成管理信息模型的规范。

23．按照权利要求22的管理网络，其特征是一被管理对象的技术规范被传送到一汇编程序，由此而产生该被管理对象的管理信息模型的规范的实现码和任意的中间格式。

24．按照权利要求23的管理网络，其特征是该实现连同该中间格式一齐组装成一装载程序包被装载到被管理系统中，而在这一装载过程中该中间格式被用来将该被管理对象的管理信息模型加到系统管理信息模型。

25．按照权利要求24的管理网络，其特征是管理信息模型的图表在被管理系统中被实现作为一特定类别被管理对象的实例，并且为每一类被管理对象生成一个这种特定类别的实例。

26．按照权利要求25的管理网络，其特征是为在被管理系统中装配以所说特定类别的对象采用一种装配方法，它在以被管理对象的实现码执行时，即使一所述特定类别的实例被装配。

27．按照权利要求20～26中任一个的管理网络，其特征是通用管理设施具有一用户接口，它是在操作期间依据信息模型的图表产生的，而且其中被管理系统的所有被管理对象均可加以检验和更改。

28．按照权利要求27的管理网络，其特征是通用管理设施在操作期间依靠将被管理系统的管理信息模型的图表转换成该通用管理设施的内部图表来管理被管理系统的管理信息模型中的变化。

29．按照权利要求26－28中任一个的管理网络，其特征是通用管理设施能识别构成管理信息模型的图表的所述特定对象类别的接口。

30．按照权利要求36－29中任一个的管理网络，其特征是每一类被管理对象均有一资源中介单元，对该类被管理对象所专用的

管理系统的协议即在其中介单元中进行端接，所述资源中介单元包含有三部分，即

——被管理对象的辅助接口的次中间单元，它是由所有次中间单元提供的一通用接口，其操作为生成、抹除、写入、读出和调整，并且其中操作总是针对一类被管理对象的一特定实例；

——数据对象逻辑；以及／或者

——求补逻辑；

资源中介单元还将该辅助接口提供给被管理对象作为外部接口，并具有两个内部接口：一个接口用于数据对象，一个接口用于求补逻辑。

31．按照权利要求30的管理网络，其特征是资源中间单元定义包括有由采用机器可解释的语言的被管理对象的特性的定义所得到的被管理对象的技术规范，并定义构成该被管理对象的资源中简单元的特性。

32．按照权利要求30或31的管理网络，其特征是由被管理对象的技术规范生成的辅助接口的实现。

33．按照权利要求1、2、3或20的管理网络，其特征是借助管理信息模型图表，通用管理设施可与一被管理系统这样交互作用，即管理信息模型中的意想被转变成适于外部用户，例如窗口管理系统、数据库管理程序等等应用的图表和结构。

34．按照权利要求33的管理网络，其特征是以软件为基础的通用管理设施以对之进行解释并使之能用于外部用户由此来与被管理系统交互作用地生成被管理系统的管理信息模型的内部表达图表。

35．按照权利要求34的管理网络，其特征是利用被管理系统

的管理信息模型的该内部图表的通用管理设施借助对内部用户的交互作用方式进行分析以及采取或提出对该被管理系统的操作来维护模型的一致性。

36．按照权利要求34或35的管理网络，其特征是通用管理设施由对被管理系统的管理信息模型的内部图表进行解释能按用于该通用管理设施的交互式用户的一致性规则执行或提出操作。

37．按照权利要求1、2、3或20的管理网络，其特征是通用管理设施包括有使得外部用户能利用被管理系统的管理信息模型的图表来与该被管理系统交互作用的功能。

38．按照权利要求37的管理网络，其特征是通用管理设施含有一外部表达图表单元，它将管理设施的内部图表转换成与相对于此通用管理设施为一外部用户（例如，窗口管理系统的用户，数据库管理者等等）相适应的表达图表。

39．按照权利要求37或38的管理网络，其特征是通用管理设施包含有能解释被管理系统的管理信息模型的通用图表的解释程序。

40．按照权利要求37－39中任一个的管理网络，其特征是通用管理设施具有依靠对管理信息模型内部图表的解释能生成可针对被管理系统的一个或数个句法上及语义上均正确的操作的功能。

41．按照权利要求37－40中任一个的管理网络，其特征是通用管理设施具有面对管理网络的访问接口，该接口用于接收被管理系统中发生的事件并向之引导操作，以及用于传送和访问被管理系统中的管理信息模型的表达图表。

42．按照权利要求37－41中任一个的管理网络，其特征是通用管理设施包括有能使被管理系统的管理信息模型的相同通用表达

图表通过不同形式的通讯网络发送的访问接口。

43．按照权利要求37－42中任一个的管理网络，其特征是通用管理设施是按照一模型实现的，该模型包含的功能使得能利用被管理系统的管理信息模型的图表并进行与外部用户相适应的图表的转换，而无需采用除该被管理系统中已规定和存储的之外的任何额外／新的信息到管理信息模型的表达图表。

44．一具有至少一管理系统和至少一被管理系统的远程通讯或开放系统的管理网络，其中所述被管理系统包含有多个由管理系统作为按资源的数据映象形式的被管理对象进行管理的物理和／或逻辑资源，以及其中管理系统为针对被管理系统的操作而利用一包含有与管理系统的操作模式相适应的所有被管理对象的说明的被管理系统的信息模型，其特征是该管理信息模型的一表达图表被实现在该被管理系统中作为一特定类别被管理对象的实例。

45．按照权利要求44的管理网络，其特征是管理信息模型规范采取该管理信息模型的可判断图表的形式，所述模型定义

——被管理系统可能具有的从管理角度上看具有意义的状态；

——被管理系统可能接收的操作；

——可以针对某一特定状态下的被管理对象的操作；以及最后

——当被管理系统受到一特定操作时可能到达的状态，

其中可判断图表模型意味着上述定义能以机器可解释语言表述以使得可由技术规范确定上述特性。

46．按照权利要求45的管理网络，其特征是在所述被管理系统在特定状态中所允许的特殊组操作由作为该类被管理对象或这一类别组的逻辑部分的先决条件和／或结束条件所决定，其中先决条件说

明，被管理系统为接收某一操作所必须处于的状态，而结束条件则说明被管理系统在一更新事务后所应具有的状态。

47．按照权利要求46的管理网络，其特征是结束条件被定义为它们根据下述二对策之一来满足：

——管理系统更新被管理系统应能保持该结束条件，在这种情况下一管理设施负责满足该条件，而如果该管理设施忽略了这一点，被管理系统即拒绝该更新事务，或者

——被管理系统自动执行为满足结束条件所需的二次更新来维持设定的限制。

48．按照权利要求46或47的管理网络，其特征是结束条件可受制于调节和生成操作。

49．按照权利要求44－48中任一个的管理网络，其特征是结束条件在被管理系统的管理信息模型中加以规定，此结束条件指明被管理系统中必须不违背的、并且适用于该被管理系统中的数据库且与存贮在该数据库中的对象实例及它们的属性值相关连的固定的一致性限制条件。

50．按照权利要求44－49中任一个的管理网络，其特征是结束条件可以说明

——属性值之间的依从关系；

——属性和数据库关系的组成数量，即对一属性的取值数量的限制；

——关于一对象型式的实例数的限制。

51．按照权利要求44～50中任一个的管理网络，其特征是先决条件表明在以数据库起始的特定操作的事务之前必须满足的被管理系统的数据库的状态限制条件。

52．按照权利要求44－51的管理网络，其特征是在采用第一对策时，一致性控制是在事务中在其被提交之前执行的，该事务只有在不违背任何结束条件时才进行下去，否则即重复进行；而在采用第二种对策时在事务被提交之前，例如在一特定属性被更新之前，在该事务中采取自动纠正措施。

53．按照权利要求44～52中任一个的管理网络，其特征是为设定包含有一被管理系统的一被管理对象中可能具有的功能性的可重复利用的部件，可以设置具有特定功能性的部件以生成所述部件不知道的一些部件能接的事件。

54．按照权利要求53的管理网络，其特征是为了这种设定，包含多种属性的部件为在这些属性值变化时产生的诸多事件。

55．按照权利要求44－54中任一个的管理网络，其特征是管理信息模型图表被分别用来解释被管理系统的数据结构和将数据结构装入被管理系统。

56．按照权利要求44～55中任一个的管理网络，其特征是管理信息模型及被管理对象的实现是以同一采用机器可解释的语言的技术规范作成的，采用这种语言的多个被管理对象的技术规范一齐构成管理信息模型的规范。

57．按照权利要求56的管理网络，其特征是被管理对象的技术规范被传送到一汇编程序，由此为该被管理对象的管理信息模型的图表产生实施码和任意的中间格式。

58．按照权利要求57的管理网络，其特征是该实现连同该中间格式一齐组装成一装载程序包被装载到被管理系统中，而在这一装载过程中该中间格式被用来将读被管理对象的管理模型加到系统管理

信息模型。

59．按照权利要求58的管理网络，其特征是为了在被管理系统中装配所述特定类别的对象，以被管理对象的实现码执行一装配方法，结果就会装配进一所述特定类别的实例。

60．按照权利要求要求44或45的管理网络，其特征是通用管理设施借助管理信息模型的图表可以这样来与被管理系统交互作用，以使得管理信息模型中的意想被变换为适宜于被外部用户，例如一窗口管理系统、数据库管理程序等所应用的图表和结构。

61．按照权利要求60的管理网络，其特征是以软件为基础，通用管理设施以对之进行解释并使之能为外部用户由此来与被管理系统交互作用地生成被管理系统的管理信息模型的内部表达图表。

62．按照权利要求61的管理网络，其特征是通用管理设施应用被管理系统的管理信息模型的内部图表以对外部用户的交互作用方式进行分析和采取或提出针对该被管理系统的操作来维持模型的一致性。

63．按照权利要求61或62的管理网络，其特征是通用管理设施借助对被管理系统的管理信息模型的内部图表的解释能够为该通用管理设施的交互式用户指引／提出符合一致性规则的操作。

64．按照权利要求44或45的管理网络，其特征是通用管理设施具有能使外部用户利用被管理对象的管理信息模型的图表来与被管理系统交互作用的功能。

65．按照权利要求64的管理网络，其特征是通用管理设施具有一外部表达图表单元，它将对通用管理设施来说为内部的表达图表变换成适应于相对通用管理设施为外部的用户（例如窗口管理系统的

用，数据库管理设施等）的表达图表。

66．按照权利要求64的管理网络，其特征是通用管理设施具有一模型解释程序，能解释该被管理系统的管理信息模型的通用图表。

67．按照权利要求64－67中任一个的管理网络，其特征是通用管理设施包括有以解释管理信息模型的内部图表而能产生一个或数个句法上及语义上正确的、能针对被管理系统的功能。

68．按照权利要求64－67中任一个的管理网络，其特征是通用管理设施包括有被管理系统访问接口，用来接收被管理系统中的事件和对该管理系统支配操作，以及用来传送和访问存贮在该被管理系统中的管理信息模型的图表。

69．按照权利要求64～68的管理网络，其特征是通用管理设施包括有能使被管理系统的管理信息模型的相同的通用图表通过不同型式的通讯网络传送。

70．按照权利要求64－69中任一个的管理网络，其特征是通用管理设施是根据一模型实现的，该模型包括有使得能应用被管理系统的管理信息模型的图表并进行向适宜于外部用户的图表的转换，而无须增加任何除被管理系统中所指定和存贮的之外的额外／新的信息到管理信息模型的图表的功能。

71．一种在具有至少一管理系统和至少一被管理系统的管理网络中的被管理系统中的一子系统中实现被管理对象的用于通讯或开放系统的方法，其中所谓的子系统是指包含有一个或多个被管理对象的被管理系统的每一部分，所述方法的特征是被管理对象是这样地与其他子系统互相独地在其子系统中实现的，即它们能与其他子系统中的被管理对象相连并传送消息而并不了解该其他子系统中的对象的形式。

7 2．按照权利要求 7 1 的方法，其特征是一第一对象被设定来与一定义一可被第一对象访问的由未实现的方法构成的接口的抽象对象相协作，以及在以后设定一所述第一对象不知道的和欲与该第一对象相协作的第二对象时，此另一对象承袭该抽象对象并实现被承袭的方法，以使该第一对象在与该第二对象协作时将其看作为所述抽象形式。

7 3．按照权利要求 7 2 的方法，其特征是在调用该抽象对象接口中定义的方法时，此调用将用于依靠随后的连接的实际对象的实现。

7 4．按照权利要求 7 2 或 7 3 的方法，其特征是在装载所述第二对象的情况下在被管理系统中重新装载所述第一对象是依靠各子系统之间的动态链接完成的。

7 5．按照权利要求 7 2 － 7 4 中任一个的方法，其特征是所述第一和第二对象分别处于第一和第二子系统中。

7 6．按照权利要求 7 5 的方法，其特征是利用动态链接使得在被管理系统中装载所述第二子系统无须重新装载所述第一子系统。

7 7．按照权利要求 7 1 的方法，其特征是通用管理设施包含有促使外部用户能利用被管理系统的管理信息模型的图表来与被管理对象交互作用的功能。

7 8．按照权利要求 7 7 的方法，其特征是通用管理设施具有一外部表达图表单元，它将对通用管理单元来说为内部的图表转换成与用户（例如窗口管理系统的用户，数据库管理程序等）相适配的、而相对于通用管理设施来说为外部的图表。

7 9．按照权利要求 7 7 或 7 8 的方法，其特征是通用管理设施包括有一能解释被管理系统的管理信息模型的模型解释程序。

80．按照权利要求77－79中任一个的方法，其特征是通用管理设施具有依靠对管理信息模型的内部图表的解释能生成一个或数个句法及语义上均正确的能针对被管理系统的操作的功能。

81．按照77－80楷　任一个的方法，其特征是通用管理设施包含有面向管理网络的访问接口，用以接收被管理系统中的事件并对之支配操作，以及用以传送和访问存贮在被管理系统中的管理信息模型的图表。

82．按照权利要求77－81中任一个的方法，其特征是通用管理设施包括有能使被管理系统的管理信息模型的同一通用图表通过不同型式的通讯网络发送的访问接口。

83．按照权利要求77－82中任一个的方法，其特征是按照一模型来实现通用管理设施，该模型具有的功能使得可能利用被管理系统的管理信息模型的图表并能加以转换到适宜于外部用户的图表，而无须给管理信息模型的图表增加除被管理系统中已指定和存贮的之外的任何额外／新的信息。

84．一具有至少一管理系统和至少一被管理系统的远程通讯和开放系统的管理网络，其中所述被管理系统包含有为管理系统作为资源数据映象型式的被管理对象加以管理的多个物理和／或逻辑资源，以及其中该管理系统为了其针对被管理系统的操作而应用该被管理系统的包含有所有适应于该管理系统的操作模型的被管理对象的说明的信息模型，其特征是

管理信息模型技术规范作成该管理信息模型的可判断的图表形式，所述模型定义

——被管理系统所能具有的从管理角度看有意义的状态；

——被管理系统所能接收的操作；

——可能针对处于某种特定状态下的被管理系统的操作；以及最后

——在经受某一特定操作时被管理系能到达的状态，

其中所谓的可判断图表模型是指上述定义应能以一种机器可解释的语言来表达，以便能由该技术规范来决定上面提到的特性，并且为定义某些被管理系统的状态规定

——可能存在的被管理对象的实例；

——它们可能具有的属性；及

——这些属性的可能值。

85．按照权利要求84的管理网络，其特征是在所述被管理系统特定状态下所允许的一组专用操作由作为一类被管理对象或一组这种类别的一个逻辑部分的先决条件和/或结束条件所指定，这里先决条件表明为接收一操作被管理系统所必须成为的状态，而结束条件表明在一更新事务后被管理系统应当成为的状态。

86．按照权利要求46的管理网络，其特征是结束条件被定义为它们根据两个对策中之一来实现，即

——管理系统更新被管理系统来维持结束条件，此时一管理设施负责满足该条件，而如果该管理设施忽略了这一职责，被管理系统即拒绝该更新事务；或者

——被管理系统以自动地执行为满足该结束条件所需要的二次更新来维持设定的限制。

87．按照权利要求85或86的管理网络，其特征是联编结束条件可能进行调节和生成操作。

88．按照权利要求85－87中任一个的管理网络，其特征是

结束条件在被管理系统的管理信息模型中加以指定，其中结束条件表明被管理对象中一定不得违背的固定的一致性限制，该结束条件可用于被管理系统中的数据库，并与存贮在数据库中的对象实例和它们的属性值相关。

89．按照权利要求85－88中任一个的管理网络，其特征是结束条件能表明

——属性值之间的依从关系；

——属性和数据库关系的组成数量，亦即对一属性的取值数量的限制；

——对一对象型式的实例数目的限制。

90．按照权利要求85－89中任一个的管理网络，其特征是先决条件表明被管理系统的数据库的状态的限制，即在带有该数据库起始处一特定操作的事务之前必须充分满足的限制。

91．按照权利要求86－90中任一个的管理网络，其特征是

——在采取第一对策时，一致性控制在该事务开始之前执行，只有在不违背任何结束条件时该事务才继续进行下去，否则即重复进行。

——在采取第二对策时，在事务开始之前采取自动纠正措施，例如在当一特定属性被更新之前。

92．按照权利要求84－91中任一个的管理网络，其特征是为设定包含有可能包括在被管理系统的被管理对象中的功能性的可重复利用的部件，可以设定具有特定功能性的部件以产生所述部件所不了解的那些部件能接收的事件。

93．按照权利要求92的管理网络，其特征是为设定包含有属性的部件，产生属性值变化条件下的多种事件。

94．按照权利要求84－93中任一个的管理网络，其中一被管理对象在被管理系统的子系统中实现，所谓的子系统是指包括有一个或多个被管理对象及被管理对象之间的数据库关系的被管理系统的每一部分，其特征是被管理对象在子系统中与其它子系统不协同相关地实现，而　要使得它能与其他子系统中其他对象连接并传送消息，而无须知道该具子系统中对象的形式。

95．按照权利要求94的管理网络，其特征是设定第一对象以与一定义含有未实现的能为第一对象调用的方法的接口的抽象对象相协作，并在随后设定一所述第一对象所不了解的并试图与该第一对象相协作的第二对象时，该第二对象承袭该抽象对象并实现所承袭的方法，以使得第一对象在与第二对象协作时将其当作为所述抽象形式。

96．按照权利要求95的管理网络，其特征是在调用抽象对象的接口中定义的方法是，该调用将依靠稍后的连接传送给实际对象中实现。

97．按照权利要求84和96的管理网络，其特征是结束条件系针对一组可能属于不同子系统的类别的类别组。

98．按照权利要求95或96的管理网络，其特征是依靠各子系统间的动态链接可以免除在装置所述第二对象时被管理系统中的原先对象的重新装载。

99．按照权利要求96－98中任一个的管理网络，其特征是所述第一和第二对象分别位于第一和第二子系统中。

100．按照权利要求99的管理网络，其特征是依靠应用动态链接使得能在被管理系统中装载所述第二子系统而无须重新装载所述第一子系统。

101．按照权利要求84的管理网络，其特征是通用管理设施借助管理信息模型的图表可能与一被管理系统这样地交互作用，即管理信息模型中的意想能转换到与外部用户（例如窗口管理系统的用户，数据库管理程序等）的应用相适应的表达图表及结构。

102．按照权利要求101的管理系统，其特征是通用管理设施以软件为基础依靠进行解释并将其提供供外部用户使之能与被管理系统交互作用来生成一被管理系统的管理信息模型的内部图表。

103．按照权利要求102的管理网络，其特征是通用管理设施利用被管理对象的管理信息模型的图表以对外部用户的交互作用方式进行分析并采取或提出针对被管理系统的操作来维持该模型的一致性。

104．按照权利要求102或103的管理网络，其特征是通用管理设施依靠对被管理系统的管理信息模型的内部图表的解释可以为该通用管理设施的交互式用户导引／提出符合一致性规则的操作。

105．按照权利要求84的管理网络，其特征是通用管理设施包括有使外部用户能利用被管理系统的管理信息模型的图表与该被管理系统交互作用的功能。

106．按照权利要求105的管理网络，其特征是通用管理设施具有一外部图表单元，用于该通用管理设施将内部图表变换成与相对于该通用管理设施来说为外部的用户（例如窗口管理系统的用户，数据库管理程序等）相适配的表达图表。

107．按照权利要求105或106的管理网络，其特征是通用管理设施包括有一能解释被管理系统的管理信息模型的通用表达图表的模型解释程序。

108．按照105～107中任一个的管理网络，其特征是通用管理设施具有以对管理信息模型的内部图表进行解释而能生成一个或数个句法上和语义上均正确的能针对该被管理系统的操作的功能。

109．按照权利要求105－108中任一个的管理网络，其特征是通用管理设施包括有面对被管理系统的访问接口，用于接收被管理系统中的事件并向之引导操作，和用于传送及访问存贮在被管理系统中的管理信息模型图表。

110．按照权利要求10－109中任一个的管理网络，其特征是通用管理设施包括有能使被管理系统的管理信息模型的同一通用图表能通过不同型式的通讯网络传送。

111．按照权利要求105－110中任一个的管理网络，其特征是通用管理设施是按照一模型实现的，该模型具有的功能使得可以利用被管理系统的管理信息模型的图表，和进行向适宜于外部用户的表达图表的变换而无须给该管理信息模型的图表提供任何除被管理系统中已指定和存贮之外的额外／新的信息。

112．一具有至少一管理系统和至少一被管理系统的远程通讯或开放系统的管理网络，其中所述被管理系统包括有多个为管理系统作为资源的数据映象形式的被管理对象进行管理的物理和／或逻辑资源，和其中所述管理系统为其针对被管理系统的操作采用一包含有所有与该管理系统的操作模型相适配的被管理对象的说明的被管理系统的信息模型，其特征是一具有使得外部用户依靠利用被管理系统的管理信息模型的图表能与该该管理系统交互作用的功能的通用管理设施。

113．按照权利要求112的管理网络，其特征是该通用管理设施包括有将通用管理设施的内部图表转换为适配于相对通用管理设

施为外部用户（如窗口管理系统的用户，数据库管理程序等）的图表的外部图表单元。

114．按权利要求112或113的管理系统，其特征是通用管理设施包括有能解释被管理系统的管理信息模型的通用表达图表的模型解释程序。

115．按照权利要求112－114中任一个的管理网络，其特征是具有一借助解释管理信息模型的内部图表而能生成一个或数个句法及语义上均正确的能针对该被管理系统的操作的功能。

116．按照权利要求112－115的管理网络，其特征是通用管理设施包括有一面向被管理系统的访问接口，用于接收被管理系统中的事件并向之导引操作，以及用于传送和访问存贮在被管理系统中的管理信息模型的图表。

117．按照权利要求112－116中任一个的管理网络，其特征是通用管理设施包括有使被管理系统的管理信息模型的同一通用图表能通过不同型式通讯网络发送的访问接口。

118．按照权利要求112－117中任一个的管理网络，其特征是通用管理设施是按照一模型实现的，该模型具有功能使得可能应用被管理系统的管理信息模型的图表，并能进行向适宜于外部用户的表达图表的转换与无须给该管理信息模型的图表提供除被管理系统中指定和存贮的之外的任何额外／新的信息。

说　明　书

远程通讯及开放系统中的管理设施

本发明是关于用于远程通讯或开放系统的具有至少一个管理系统和至少一个被管理系统的管理网络，其中所说的被管理系统包括有许多以管理系统作为被管理对象加以管理的资源数据图象形式的物理和／或逻辑资源，和其中所说的管理系统其操作系针对被管理系统，采用一种该被管理系统的信息模型，该模型包含有适应于该管理系统的操作方式的所有被管理对象的说明。

本发明还关系到在一用于远程通讯或开放系统的具有至少一个管理系统和至少一个被管理系统的管理网络中实现一被管理对象的方法，其中以子系统来表明一被管理系统的每一含有一个或多个被管理对象的部分。

“具有至少一个管理系统和至少一个被管理系统的管理网络”是指该管理网络可以至少包括一个能管理一个或多个同样能构成该管理网络的部分的被管理系统的管理系统。

开放系统是指一种ＣＣＩＴＴ（国际电报电话咨询委员会）应用的开放系统互连（ＯＳＩ）的参数模型中的定义的系统，Rec.X.200。

为了执行一管理范畴内的管理工作，就必须具有至少一个承担资源管理的管理者。资源则是某种包括该范畴的意想和能力的事物。例如一个范畴是一个远程通讯网络，此时，资源就是转换器，中继线等，而管理单元则为例如机务装备和网络管理系统。

为电话网络的运行各个公司已采用了许多各种不同的运行维护系统。ＣＣＩＴＴ已制订了一种称之为ＴＮＮ（远程通讯管理网络）的用于电话网络中的运行维护标准模型。ＴＭＮ的基本原理是指明一种能将各种不同管理系统连接到远程通讯设备的有组织的网络结构。这是借助应用标准化的协议和接口来实现的。电话公司和其他控制器将要求将来的远程通讯与ＴＭＮ相适配。

ＣＣＩＴＴ对此有一个在研制中的建议，M.3000系列。

ＴＭＮ将全部网络节点都看作为网络部件（ＮＥ）。这些网络部件被作为电话交换机和发送传输网络产品。

ＴＭＮ的功能组成包括有：

——管理功能（ＯＳＦ，Operations Support Functions ），管理用户可用的应用程序，例如用于“事务管理”和服务以及网络支配的管理功能；

——数据通讯功能（ＤＣＦ，Data Communication Functions），管理管理系统ＯＳＳ和被管理系统ＮＥ之间的数据通讯；

——中介功能（MF，Mediation Functions），变换信息（例如在被管理对象之间），管理数据，集中，归纳和编辑，对例如阀值极限及存储数据作出决定，以识别设备和网络；

——网络元件功能（NEF，Network Element Functions），管理切换功能及传送的远程通讯过程，并参与故障查找和保护连接的远程通讯管理过程；

——接口适配功能（QAF，Q-Adapter Functions），进行非标接口至标准接口之间的转换；

——工作站功能（WSF，Work Station Functions），组成ＴＭＮ

的用户终端，显示信息并协助管理技术人员管理网络。

TMN还包括一被称之为Q_3的接口。Q_3除作为一通讯协议外，还是一个由数据型式、操作和提示组成的信息模型。Q_3接口的细节及其协议见CCITT建议Q_{961}和Q_{962}。

TMN将所有物理逻辑对象均看作为被管理对象，它们在TMN中被称之为MO（Managed Qbjects），本文此后亦将交替称用这一名词。这些被管理对象为诸如连接导线、电路、信号终端、传送路径、事件记录器、警告报表等物理或逻辑资源的数据映象。

在一个资源与一被管理对象之间存在有一特定的关系。一个资源可能与一个或多个MO相关，或者根本与任何一个均无关。另一方面，MO也可能与一个或多个资源相关，或者根本与任一个也无关。在一MO受某种形式的操作或维护作业作用时这种关系就很重要。当加之它的下属功能本身被撤消之前，该MO必须不被称走。这一信息模型基于对象取向和关系概念。

管理系统（OSS，Operation Support System），将网络元件和下属管理系统作为一设想的数据库MIB（Management Information Base ）中的一组被管管理对象来加以处理。这些被管理对象由一类MO中的多个实例组成，例如一系列相同形式的信号终端。这样每一终端就将成为该类信号终端的一个实例。

TMN中还存在有基本概念，MIN (Management Information Model），它是集合地指所有与被管理对象有关的信息。MIM是一个与被管理对象有关的全部属性、关系、操作及通告的模型。为了能检索MO实例，采用了一种管理信息树MIT（Management Information Tree）。这一树结构启动网络并指明网络元件和用户或设备。

为了操作和维护，每一个要求信息的分开的单元或资源或受系统外部影响的操作，由一个被管理对象表示每一个与必然受到影响的和必须报告某事件（例如建立数据指定名称或警告控制）的操作或单元有关的信息交换。均以对被管理对象的操作或来自被管理对象的记录形式实现。

一被管理对象包含多个属性，还可能与其他被管理对象有关连。可以有许多不同的操作针对一个被管理对象，而这些对象就可能产生多个事件。

下面将说明TMN的不足之处。

网络元件和从属管理系统由管理系统以针对被管理系统内的被管理对象的操作并监视被管理系统发送出的通告来进行管理。

针对被管理系统中的被管理对象所允许的操作依靠该被管理系统的一个信息模型与可被传送的通告共同决定。此信息模型阐明：

——所定义的被管理对象的类别；

——这些类别中的被管理对象接受的操作；

——被管理对象预期会发出的通告；

——每一类被处理对象中能被产生或去掉的实例数；

——被管理对象之间的依赖性，例如一被管理对象需要另一个的存在；

———被管理对象内的依存性，例如，一属性的特定属性值只有在另一属性被设定到某一特定值时才是允许的，或者该被管理对象如果处于某种特定状态对就只能被去掉；

——被管理对象及它们的通告的目的和意图。

为了付于实际意义，管理系统必须理解被管理系统的信息模型。

在TMN中这被称为“共享管理消息”。

在信息模型改变时，管理模型必须跟随这一变化加以更新。在通常系统中这些改变是由新信息模型的定义作成的。这由用于写成为GDMO/ASN.1模板（GDMO, Guidelines for the Definition of Managed Objects 按照CCITT rec. X. 722 ISO/IEC 10165-4）的被管理对象的及用于该被管理对象的ER图形（Entity-Relationship图形）而编写的规范来实现。此被管理对象规范对被管理对象作（可为机器读取的）句法体系（例如操作和通告）正式的说明。

该信息模型的所有其他部分，如相关性、实例的数目等，则以自然语言作为注解作非正式的说明。

——实现并验证管理系统和被管理系统中新的信息模型。

——由执行合格的测试序列来证实管理系统和被管理系统与此同一信息模型相适配。

——以这一信息模型的新版本来更新组成管理系统和被管理系统的网络。

刚才上述的这些引起许多问题。

首先，管理系统和被管理系统的开发必须协调，这导致高开发成本和／或者延迟其进入市场。

其次，被管理系统规范的欠缺使得管理系统和被管理系统的实现、验证和验收成为一个困难的需要时间的任务，因为对这些规范的解释会存在争议。

第三，网络的更新必须经过仔细的计划和进行，因为管理系统和被管理系统的不同版本之间具有依赖关系。这就会涉及延误网络中实现新的功能。

依据TMN模型的管理的目的是为对远程通讯和开放系统的管理建立标准化机制。管理结构对新系统体制的管理形式具有很强的影响。采用按照TMN模型的管理型式进行全面管理而不仅仅只限于经受标准化这一方面是有充分理由的。这方面的主要理由就在于能以统一的方法开发和设计管理功能是最理想的。

总的说来，本发明的第一目的是提供一种新的管理功能的体系结构，以便使得：

——管理功能的设计费用合理；

——较少需求管理功能和资源更新的协调和计划；

——有效的支持像网络管理这样的综合过程；

——可以相对于资源技术独立地开发管理技术；

——可以合理地利用各种不同的能力，就是说，例如资源领域方面的专家将集中精力于本领域，而熟悉如何作管理设计的人员则将投身于这一方面。

根据本发明的第一特点，这一目的以及在下面的叙述中将出现的其他一些目的是如此达到的，即由引言所定义的管理网络除被管理系统外还包含有一通用的管理设施和一管理信息模型的图表，其中该通用管理设施在运行期间的性能即由该模型图表来决定。

根据第二特点，所述目标是这样达到的，即在所讨论的管理网络中为被管理系统设置一个作为特定类型被管理对象实例的管理信息模型图表。

根据第三特点，所述目标的达到则在于：讨论中的管理网络的特征是该管理信息模型规范采取的是管理信息模型的可加以判断的图表形式，所述模型定义：

——从管理上的观点出发，被管理系统能采取的有关状态；

——被管理系统能接收的操作；

——能针对一特定状态下的被管理系统的操作；以及最后

——被管理系统在经受到一特定操作时所到达的状态。

其中所谓的可判断图表模型是指，上述定义的内容能以一种机器可解释语言表示，以使得能根据该规范来决定上面说明的特性，和定义某一规定了可能存在的被管理对象实例，它们可能具有的属性和这些属性的可能取值的被管理系统的状态。

根据第四特点，所述目标是这样达到的，即讨论中的管理网络的特征是通用的管理设施，它具有使外部用户能通过运用被管理系统的管理信息模型图表来与该被管理系统进行互相配合的功能。

所述目标还是这样达到的，即在前言中提出的方法的特点在于被管理对象是这样被与其他子系统互相独立地设置在其子系统中的，它们能与其他子系统中的别的对象相连接传送消息，但不知道那些其它子系统中的对象的形式。

已获得本发明的各种有利方案，各从属权利要求中描述其特点。

将管理系统分开具有下列优点：

——通用管理设施可以对被管理系统中较多的应用程序的管理功能重复应用；

——通用管理设施不受信息模型中的变化影响；

——具有不同方案中的被管理系统和不同功能的异种网络可被一个同样的通用管理设施加以管理。

实现在被管理系统中的信息模型图表的优点如下：

——该图表总是存放在网络的一个节点中；

——该图表总是与被管理系统的模型相一致；

——共享管理资料易于管理；

——系统和网络的更新易于管理；

——能不产生操作干扰地更新系统和网络。

不会出现管理系统和被管理系统之间关于哪一信息模型有效的分歧的时间间隙。

将信息模型规范作成可判断的信息模型图表的形式也就带来一系列优点：

——此通用管理设施可以作得更为有效，因为它能断言被管理系统在一特定操作之后的新状态，而且它还能指出某种特定状态下所能容许进行的操作；

——被管理系统的实现和验证以及管理应用程序均简单化了，因为完善地指明了预期的性能。由该规范还能产生大部分工具程序。

——运行期间的耐用性得到改善，因为只有导致被管理系统中允许的状态转　变的操作才接收。

——信息模型的早期仿真和评估简单化了，这使得订立规范的工作简易可行；

——在此结构投入运行状态之前就可将管理模型作得其操作次序能较自由但仍保证被管理系统结构的完整性从而更可靠和易于应用。

下面将对本发明的多个实施例参照附图作较详细的讨论。所列附图中：

图 1 为描述本发明的一个基本原理的方框图；

图 2 为实现图 1 中方框单元之一的流程图；

图 3 为按照本发明管理网络的体系结构方框图；

图 4 为图 3 中单元之一的方框图；

图 5 用于描述一资源的控制管理系统状态的状态图；

图 6 为说明型式与实例等级之间的关系的状态图；

图 7 和 8 分别为描述一个实例等级上的数个单元如何共享第一和第二系统中的形式等级上的一共用单元的状态图；

图 9 为说明按图 7 和 8 二系统的控制管理系统状况的状态图；

图 1 0 为描述存放在一型式单元中的信息模型示例的状态图；

图 1 1 说明控制管理系统能以解释图 1 0 中的资源图表而执行的操作；

图 1 2 描述信息模型的改变；

图 1 3 描述一个违背图 1 0 及图 1 2 中的信息模型、并决不可能针对被管理系统的操作的示例；

图 1 4 为描述在一个具有两个控制管理系统的管理范畴内新和归技术如何能共存的方框图；

图 1 5 为表明一按照图 1 所描述的本发明的原理所得到机制结构内的组成管理系统单元的方框图；

图 1 6 为描述人类用户和资源的图表之间同作用的状态图；

图 1 7 为与图 1 6 类似的描述这样一个无效的协作的状态图；

图 1 8 描述如何将资源图表组织进数据结构；

图 1 9 为用以说明目的的极其粗略地图示一由管理系统和被管理系统所组成的模型驱动系统的总体设计；

图 2 0 示意说明不同类别的两个对象型式；

图 2 1 为与图 2 0 相类似的说明实施本发明时所采用的一特定对象型式的特征；

图 2 2 示意说明为构成管理信息模型，在“标准”的对象类别与特定对象形式的对象之间的连系；

图 2 3 描述在构成信息模型方面进行编写程序期间的各不同瞬间；

图 2 4 ～ 2 9 描述启动对被管理系统中一特定对象类别的信息进行解释的时刻；

图 3 0 ～ 4 0 为伪码定义，其中

图 3 0 为被管理对象的定义；

图 3 1 表明如何以属性型式来决定可能的属性值；

图 3 2 为操作定义；

图 3 3 说明先决条件示例；

图 3 4 说明结束条件示例；

图 3 5 说明一被管理系统检测到一致性时的结束条件示例；

图 3 6 说明被管理系统维持一致性时的结束条件示例；

图 3 7 为一称之为LineDriver的对象的定义；

图 3 8 说明依从关系示例；

图 3 9 说明与一种方法相关连的结束条件的示例；

图 4 0 说明与一生成操作相关连的结束条件的示例；

图 4 1 － 4 4 为概要说明先前有关本发明的介绍中已经涉及到的可应用部分以及其各种状况（具体参照图 3 和 4 ）的方框及功能图；

图 4 5 举例说明被管理对象在属性、方法和前提及结束条件方面的规范的语法体系；

图 4 6 及 4 7 中以运用相同的语法体系来确定的二不同对象形式的规范；

图 4 8 同样以同一语法体系来确定图 4 6 及 4 7 的对象形式之间

的依从性规范；

图 4 9 为表明描述结束／先决条件与它们可采用的概念之间的关系的概念模型的方框图形式；

图 5 0 为应用与图 4 5 ～ 4 7 中同样语法体系来实现维持图 4 6 与 4 7 中对象型式之间的依从性的机制规范的几个示例；

图 5 1 为对同样利用同一语法体系来指定按图 4 6 对象形式中的属性作为从图 4 7 的相关对象形式中的属性中得出的一个属性的说明；

图 5 2 规定由一属性向图 5 2 中的另一属性传播；

图 5 3 以流程图形式表明一具有协同的一致性检测的事务的步骤；

图 5 4 表明一说明文件的公共部分，该文件是在将图 4 6 的对象型式编制成 C+ + 编码级别时得到的；

图 5 5 同样以 C+ + 表示一在图 5 4 的文件中协同工作的方法的实现；

图 5 6 以 C+ + 表明由图 4 7 及 5 0 的规范所产生的说明文件；

图 5 7 的 C+ + 表明在图 5 6 的说明文件中的对象的两个方法的实现；

图 5 8 以流程图形式说明用于一包含有先决条件检测的事务操作的执行的算法；

图 5 9 以 C+ + 说明由检测为去除一对象的先决条件的方法来扩展图 5 4 的语句文件；

图 6 0 以 C+ + 说明图 5 4 的对象的两个方法加实现；

图 6 1 ～ 6 9 以方框图描述在有关管理系统的设计中采用本领域现有技术时可能出现的问题，其中

图 6 1 ～ 6 4 是关于在重复使用被管理系统的被管理对象中的库

存部分时可能出现的问题；

图 6 5 及 6 6 涉及在实测一分层系统结构中的被管理系统时可能出现的问题；

图 6 7 ～ 7 0 描述如何能借助设计一能与未知的未来被管理对象相协作的特定型式对象来总体解决图 6 1 ～ 6 6 的问题；

图 7 1 ～ 7 4 针对与上述有关图 4 5 ～ 5 2 的相同示例并采用同一准语法，指定图 7 0 ～ 7 4 的对象的设计，图 4 7 的对象型式则被看作为属于一工作台系统；

图 7 5 ～ 8 0 说明采用程序语言 C++ 来实现图 7 1 ～ 7 4 对象的设计之间的依从关系。

根据本发明的特点之一是将管理系统归入一通用管理设施与被管理系统的信息模型的（表达）图表分离。

图 1 对此作了示意的说明，其中标号 2 为管理系统。通用管理装置 4 的功能由信息模型 8 的图表 6 决定。换句话说，由模型图表 6，该通用管理设施可确定对被管理系统 1 0 应采取的操作，还可决定为使其得到所希望的状态所需进行的操作。此模型图表 6 还被用来正确地解释分别来自和送往被管理系统 1 0 的数据包。

在当被管理系统 1 0 中加入新的资源时，反只需要改变模型图表 6。

根据本发明的另一特点，被管理系统中被引进（存贮）有该信息模型的一个图表。

为对该系统中被管理对象提出规范并予实现，采用了一种机器可解释的语言。针对被管理对象即以这一语言编写技术规范。这些被管理对象的技术规范共同组成该管理信息模型的规范。

这一规范被传送到一汇编程序，产生实施存根码和管理信息模型的中间格式。

此实施存根码然后就可能经人工提纯处理并经汇编来实现被管理系统。这一实施的结果然后与管理信息模型的中间格式一齐来组成装入程序包。此装入程序包然后被装载到目标系统，即被管理系统，在此装入过程中，管理信息模型的中间格式被用以产生管理信息模型的图表。

在被管理系统中，管理信息模型的图表被作成为一特定类别的被管理对象的实例。这一类别下面被称为MIM-server，(Mrnagement Information Model Server)。对于每一类别被管理对象均生成一MIM-Server类别的实例。

装入程序包的产生可由图2大致看到。

按照图2，在第一步11写入被管理对象的规范12。在下一步13，这一规范被汇编成C++语言的实施存根码14和对象规范的中间格式16。关于引用C++语言的详细说明见Margaret A Ellis, Bjarne Stronstrap 的“The annotated C++ Reference Manal”。

如步骤18所示，被管理系统以C++实现。这一实现即为所生成的包括该规范的中间格式的存根码。在步骤20，C++的实现形式被汇编到22。

在被管理系统中装程序包30时，每一由装入程序包所实现的被管理对象类别均产生一新的MIM-Server类别的实例。

根据本发明的另一特点，管理信息模型规范是作为管理信息模型的一可判断图表工作的。

管理信息模型指明

——被管理系统可能呈现的状态（以管理观点出发的关心的状态）；

——可对被管理系统执行的操作；

——可对特定状态中的被管理系统执行的操作；以及最后

——在被管理系统经受某特定操作时所达到的状态。

这里所说的信息模型的可判断的图表是指：这些定义能以一种可为机器解释的语言来表达，以使得上述特性能由规范所决定。

图 3 中的方框图说明管理功能的体系结构的基本设计模块。

在标号 4 0 的管理系统中，设有带用户接口 4 4 的通用管理设施 4 2。在这一接口中被管理系统的所有被管理对象可加以检验和调整。

ＤＣＦ－Data通讯功能 4 6（在引言中也已经提及过）是一将ＣＭＩＰ或其相似作为管理协议的协议的实施，这一通讯功能并非本发明的部分。

在标号 4 8 的被管理系统中包括有一通用中介单元 5 0，与管理协议相端接。

为被管理系统中的各类被管理对象设置了许多资源中介单元 5 2。每一类被管理对象均具有一个资源中介单元。该类被管理单元所专用的管理协议的端点与该资源中介单元连接。

MIM-Server 54（同样在上面已提到过）总的说来是一个MIM-Server类被管理对象的资源中介单元。此MIM-Server是被管理系统的实际管理信息模型图表的提供者。

数据库管理功能 5 6（DBMS, Data Base Managing System）为一面向数据库服务器的对象。它能存贮、访问和更新构成持续数据的对象。

被管理对象的辅助接口 5 8（MOSI, Managed Objeet Service

Interface）是一个由所有资源中介单元提供的总接口。这一接口中的操作为形成，抹除，写入，读出和整理。ＭＯＳＩ中的操作始终是针对一特定类别被管理对象的特定实例的。

数据管理接口６０（DMI，Data Managing Interface）为面对ＤＢＭＳ的接口。它具有生成、抹除、读出和更新构成持续数据的对象的操作。

此ＭＯＳＩ接口技术规范如下：

```
#ifndef COOAMOSIBASE_HH
#define COOAMOSIBASE_HH
/* $Id: CooaMosiBase.hh,v 1.6 1992/12/01 07:34:52 euassg  $*/

#include<CooaMosiVersion.hh>

class DelosBuffer;
class DICOS_DbTransaction;

enum CooaGetMode
{
 Cooa_getSpecified=0,
 Cooa_getFirst=1,
 Cooa_getNext=2
};

typedef unsigned int CooaAttributeID;

typedef unsigned int CooaActionID;

typedef unsigned int CooaMoClass;

typedef DelosBuffer CooaAccessControl;

enum CooaResultValue
{
 Cooa_accessDenied=2,
 Cooa_noSuchAttribute=5,
 Cooa_invalidAttributeValue=6,
 Cooa_noSuchAction=9,
 Cooa_processingFailure=10,
 Cooa_noSuchArgument=14,
 Cooa_invalidArgumentValue=15,
 Cooa_missingAttributeValue=18,
 Cooa_classInstanceConflict=19,
 Cooa_mistypedOperation=21,
 Cooa_invalidOperator=24,
```

```
 Cooa_invalidOperation=25,
 Cooa_notReplacable=1000,
 Cooa_noDefault=1001,
 Cooa_notAdded=1002,
 Cooa_notRemoved=1003,
 Cooa_false=1004,
 Cooa_invalidCompareMode=1005,
 Cooa_noIteration=1006,
 Cooa_noMoreAttributes=1007,
 Cooa_ok=1008,
 Cooa_notReadable=1009,
 Cooa_interfaceViolation=1010,
};

enum CooaSetMode
{
 Cooa_replace=0,
 Cooa_toDefault=1,
 Cooa_addMember=2,
 Cooa_removeMember=3,
 Cooa_initiate=4
};

enum CooaCompareMode
{
 Cooa_equal=0,
 Cooa_greaterOrEqual=1,
 Cooa_lessOrEqual=2,
 Cooa_present=3,
 Cooa_subsetOf=4,
 Cooa_supersetOf=5,
 Cooa_nonNullSetIntersection=6,
 Cooa_initialString=7,
 Cooa_anyString=8,
 Cooa_finalString=9
};

enum CooaOpenMode
{
```

```
 Cooa_create=0,
 Cooa_delete=1,
 Cooa_update=2,
 Cooa_read=3
};

class CooaMosiBase : public CooaMosiVersion
{
 public:
 virtual unsigned int Cooa_version () const { return 2000; };
 virtual void get (CooaGetMode mode,
         CooaAttributeID& attributeNumber,
         DelosBuffer& attributeValue,
         CooaAccessControl& access,
         CooaResultValue& result,
         DelosBuffer& errorInformation)=0;// pure virtual

 virtual void set (CooaSetMode mode,
         CooaAttributeID attributeNumber,
         DelosBuffer& attributeValue,
         CooaAccessControl& access,
         CooaResultValue& result,
         DelosBuffer& errorInformation)=0;// pure virtual

virtual void action (DelosBuffer& argument,
         CooaActionID actionNumber,
         DelosBuffer& actionResult,
         CooaAccessControl& access,
         CooaResultValue& result,
         DelosBuffer& errorInformation)=0;// pure virtual

 virtual void compare (CooaCompareMode mode,
         CooaAttributeID attributeNumber,
         DelosBuffer& attributeValue,
         CooaAccessControl& access,
         CooaResultValue& result,
         DelosBuffer& errorInformation)=0;// pure virtual
 virtual void mode (CooaOpenMode openMode,
         CooaMoClass moClass,
```

```
            CooaAccessControl& access,
            CooaResultValue& result,
            DelosBuffer& errorInformation)=0;// pure virtual
     virtual CooaAttributeID getPrimaryKey(CooaResultValue&
result) = 0;

     virtual CooaMosiBase* create (DelosBuffer& primaryKey,
            DICOS_DbTransaction& trans,
            CooaMoClass moClass,
            CooaAccessControl& access,
            CooaResultValue& result,
            DelosBuffer& errorInformation)= 0; // pure virtual

     virtual void getCounter (CooaAttributeID& attributeNumber,
            void*& counterObject,
            CooaAccessControl& access,
            CooaResultValue& result,
            DelosBuffer& errorInformation) = 0; // pure virtual
    };

    #endif
```

通用对象管理设施42识别MIM－Server接口。它由MIM-Server读出数据并认定被管理系统中为哪一类被管理对象。

在用户接口44中，例如，有可能提出该通用对象管理设施指明一特定类别被管理对象的全部实例的要求。通用对象管理设施能由MIM-Server 54 读出被选择类别的定义。它辨别如何经由DCF到通用中介单元来访问这一类别。它还认定对来自该通用中间单元的数据应怎样解释。

如图4所示，资源中介单元由三部分组成，即MOSI-Subagent 61，数据对象逻辑62（DOL，Object Logic），和全补逻辑64（CL，Complementary Logic）。它还设有外部接口58，MOSI，并具有两个内部接口，即数据对象接口66（DOI，Data Object Intorface）和求补逻辑接口68／CLI，Complementary Logic Interface）。

对参考图1所一般性描述的将管理系统分为通用管理器和被管理系统模型表示的原理，现在将作更详细的描述。为了以后参引方便起见，将这种划分也称之为划分模型。

为了管理系统内的资源，必须存在这方面的知识，即什么应该得到管理和它又是怎样得到执行的，为此这儿存在着3种所须的主要功能，也就是管理系统的控制部分，下面简称为“控制管理器”，资源的表示和资源，图5详细地描述了这三种功能之间的关系，问题是要给控制管理器一般的可能性去管理特定的资源，这因此需要实现框架结构，该结构允许应用设计，该结构能够管理资源而无须知道资源的内部结构和它的可能约束，更具体结说，应该能够添入新的资源，并且旧的资源能够被取消或者能够改变，而不影响控制管理器。

这样的框架结构能够被用于每一个这样的领域，即在控制管理器

和被管理系统之间存在着关系。

必须指出的是，资源表示总是被表示为在类型水平上的语句，这就意味着一种特定类型的所有实例具有共同的资源表示，在图 6 可以看出，在类型水平上的单元 $T_1 - T_4$ 是独立存在的，而在实例水平上它们能被分配在在单元 $I_1 - I_4$ 中，在特定的实例中图 7 的方案是有效的，类型用户号保持着信息，该信息描述应该怎样翻译（即用户号的格式和它的数据结构类型，这被用来保持电话号码）。

这样的结构技术允许使用通用管理器。如果图 7 的方案对系统 S_1 是有效的，而图 8 的方案对系统 S_2 是有效的，相同的一般管理器能够用来管理两个系统 S_1 和 S_2 。

两个系统的管理器示图在图 9 中给出，这正是在系统 S_1 和 S_2 中使用的同样的一般管理器，并且它用于解释在系统 S_1 和系统 S_2 内的模型表示，系统 S_1 和 S_2 的用户数的管理信息模型是不同的。

在图 9 内管理器把它的操着指向资源，该资源在系统 S_1 和系统 S_2 内是应当被管理的，为了得到正确的句法和数据格式，管理器解释模型表示 R_1 和 R_2 ，从用户的观点来看，管理框架结构是相同的，也就是具有相同的概念，可以使用隐喻和工具，通过这样的手段可以获得独立于它们的实际执行的有效并且简单的管理资源。

现在的目标是实现框架结构，该结构能够设计一般的管理器，该结构处理物理的和／或逻辑的资源、框架结构是通过严格的和形式上分开资源表示（模型）和真实的资源施行的，称为控制管理器的软件单元解释资源表示。

在开始工作之前，管理器并不必事先须要任何特定的资源信息，这意指管理器单元有能力采用任何可能资源表示，它可以以这样的方

式表示，从而使得它能被管理器解释，主要的优点在于可以引入新类型的资源而无须更新管理器。

作为实例，可以使用给定的资源，例如一个用户，在用户中引入新的可能性，并且仍然能够使用同一个管理器管理新的和旧的用户两者。

作为另外一个实施例，可以在执行全新类型资源时把它引入到一个管理器系统，用现存的诸管理器管理新的资源。

这里，对图 5 示出的接口将加以极为详细的说明。

控制管理器使用该接口去联接资源表示并从它收集信息，在执行指向资源的一般管理操作时使用该信息，该例在图 1 0 中示出，通过解释资源表示，管理器能够在用户实例 I_1 上执行例如图 1 1 的操作，而且确保对模型来说句法和语义是有效的。

必须看到的是，同一个管理器是有能力管理图 1 0 的资源表示中的信息更新变化的，这在图 1 2 中已经示出，而不须要重新编译和重新构型。

讨论中的接口具有如下的操作

1 － 联接到表示

2 － 解释表示

资源的存取接口

执行指向真实接口的管理操作时使用该接口，在该接口的操作经常不是独立执行的。

操作：

1 － 执行操作（打开／读／写／查找等等）

2 － 转送事件

管理能力的信息接口

该接口陈述使用特定管理器管理资源的可能性，该接口限制管理器在模型表示的存取接口中的行为。

操作：

1 －检查存取

2 －限制存取

资源能力的信息接口

该接口陈述将被输出并且将在它的资源表示中示出的资源能力，在真实资源中的一些能力形式能够多路，并且在资源表示内被陈述为一个单独的能力。

操作：

1 －输出可能性

2 －多路可能性。

控制管理器

所讨论的框架结构中的控制管理器是能够解释以机器可以解释的方式表示的模型表示的软件单元，这些表示陈述在管理器和被管理的资源之间的可能协作的协议书，表示也陈述了被用来表示在被管理资源中的概念的数据结构，许多控制管理器能使用一种表示，并且一种控制管理器能使用多种表示。

表示

资源的表示从目前的文本来看是“在一个资源中”的一些方面或是所有的方面的视图，这些也可以是一个资源的多视图，在表示内，信息是以这样方式构成的，从而可能对它进行解释使其变换为其它表示，后者更适合与人协作，表示能被用来为执行管理操作馈送带有必

要信息的多少带有智能的软件。

上面描述的划分模型的主要优点在于，它提供了引入和更新在管理范畴内的资源和服务的装置，并且没有影响管理这些资源和服务的单元。近而，管理真实资源的单元能够被替换／更新而不影响被管理的单元，当新技术可以被使用时，可在对管理范畴合适的时间内被引入。

旧的和新的技术能够在管理范畴内共存，在例如通讯网络内设备（资源单元和管理单元）能被引入到一个或几个考虑周到的活动中。

在图 1 4 中示出了一个管理范畴，它由具有不同型式的单元的两个系统组成，形式在每一个系统内和在两个系统之间是不同的，在这种连接中下述是有效的：

1 －类型 1 的“旧”管理器能够管理类型 1，1 0 0 和 2 0 0 的资源。

2 －类型 1 0 0 的“新”管理器能够管理类型 1，1 0 0 和 200 的资源。

3 －还没有已经发展到类型 2 0 0 的管理器。

4 －所有类型的管理器管理着所有类型的资源。

在图 1 4 中有一个工作范畴，其中技术发展以不协条的方式发生了。我们所看到的是，在每一层（水平）中修改已经发生了，但从管理器方面（垂直）来看没有影响单元。

这里将要描述，上述的划分模型是怎样与设计控制管理单元联系起来使用的。

对指向一个资源的管理操作负责的控制管理器在划分模型框架内从模型表示获得必要的知识，该表示包括了资源的模型信息。

管理器是在由划分模型提供的框架结构上形成的，其中，在这个框架结构中的通用管理器是按图 1 5 设计的。

在管理器层 8 0 中设计包括了单元数，该层的命名在图中看是明显的，而在模型层 8 2 内包括了资源表示。

配合外部用户接口的单元（在 8 0 处）被用来使管理器适配外部单元，这些例如是：

－窗口系统，通过操作可再生的表示，诸如表格，菜单和图形表示，使使用人和资源协条工作。

－其它的计算机／计算机系统

－数据库

－管理范畴本身的一个单元，管理器也能被使用作为控制和作出决定的单元。

作为用户与其进行协作的表示的实例，可考虑图 1 6，它给出了窗口是怎样与图 1 0 的管理信息模型相关的。

该信息也被用在检验输入信号根据模型是正确的，在上述的例子中，用户并不能引入一个不适合图 1 0 的资源表示的电话号，管理器能够使用知识去指导用户或建议该作什么，根据图 1 0 是无效的并且在图 1 2 示出的用户协同操作不能传给管理操作但被传送到资源。图 1 7 描述的操作构成管理器模型解释能力的结果，该管理器使用了划分模型框架结构。

真实管理的资源（my Suboir）始终处在有效的状态，并没有企图要执行无效的操作。

使用这里描述的管理器类型，若干单据传送到模型的使用者，也就是一般的管理器，而不是由资源本身执行，这就暗示，在不同用户

的情况下，资源有能力实现其愿望，用户情况或许会使用不同的时间，在传统的情况下，预测可能出现的未来用户情况是必须的，而在目前的框架结构情况下，只需要作出另一种模型表示。

在更传统方式的实例中“用户号资源”能够执行在图10中描述为表示资源的编码中的C++语句的规则。从而资源的应用被限制到电话号码只能从727开始的文本中。

内部表示单元86变换模型表示为适合控制管理器使用的表示。也就是在模型中的抽象的数据结构等等被换成在管理器工作时能被使用的数据结构。

内部表示单元产生了用来存储资源表示的结构，对每一个管理器范畴至少存在一个该结构，在开始时结构是空的但在后来装有资源表示，该结构能够以若干方式（原始存储器，目标数据库，等等）产生。

从图18可以看出，资源表示R1－R5按数据结构进行组织，控制管理器使用这结构去存取资源表示。

可以看出资源表示也具有关系，这些关系组成了管理器范畴的模型的部分（用户目录具有例如与LIC的关系）。

管理器使用管理信息模型的一般管理器内部表示去保持被管理系统的相容性，并且在某种操作执行后去决定管理系统的状态，操作能导致违反相容性，但在操作被施加到被管理系统之前它能被发现。

管理器可以以不同的方式使用解释模型所获得的知识，例如：

－自动解决违反相容性的操作和产生一组不违反模型的管理操作，

－指导使用的用户去校正一组操作，

－对“如果，则”执行分析。

一般管理器逻辑单元88控制着其它单元的活动，它也根据由其

它单元报导的事件作出决定，管理器以外的单元也报道事件，外部事件总是变换成能被管理器管理的表示和结构。

外部表示单元９０管理着把管理器中的内部表示变换成为传送到一个单元和被它们理解的表示，该单元联接到在８０处的“适配外部用户接口”的单元，这样一种表示的例子是表示一个窗口的结构．该窗口能在Ｘ－窗口装置中重现。

在９４的管理操作的单元“Machine”产生真实的管理操作，该操作应进而引导到管理器范畴中的一个资源，该操作是作为由管理器的逻辑单元８８的其中一个所控制的表示的内部或外部（或同时）合作／操作的结果产生的，代表一种形式的窗口是例如一种由管理器控制的表示，窗口可以示给用户可替代的用户，一些活动可以产生传送到被管理资源的一个或者多管理操作。

在９６的管理系统范畴的单元存取接口负责传送管理操作到受控单元，讨论的存取接口能适合各种管理协议书。

在９２的单元模型解释器负责解释在模型中表达的资源表示，模型解释器读出资源表示并把它们分配给上述的单元作为内部表示。

模型解释器解释一个资源的表示，该表示以一种约定的格式来表示。

为了执行管理操作，也就是改变用户的号码，需要执行下面的操作：

１．建立与管理范畴的联接。

２．用户（对外部表示而言，估计可以是单元之外的用户）对将被管理的资源（也就是用户）寻址。

３．如果这儿已经存着打算被管理的资源的内部表示，管理器逻

辑单元通过单元检查内部表示。

4．如果这样的表示并不存在，管理逻辑单元命令管理操作“机器”通过管理范畴的存取接口执行管理操作以收集被管理系统的资源表示（否则每一个都继续下面的步骤7）。

5．被收集的淘汰表示传送到用于内部表示的单元，该单元把它变换成适合管理器的格式。

6．用于内部表示的单元确定该表示在一个结构中的位置，该结构表示了在管理范畴内的资源表示的和。

7．如果这样的表示存在，管理逻辑单元变换内部表示为使用外部表示单元的适当的外部表示。

8．外部表示进一步引到供外部用户采用的接口，在那里用户可以（通过操作以X窗口示出的一种形式）使用表示。

9．外部用户改变用户号码符号的值（用户输如例如一个值到为此指定的一台设备）。

10．模型解释器检查是否输入的值在资源表示中或在资源表示出现的文本中违反了什么（意味着可能需要对被管理系统进行一些管理操作）。

11．如果检测出违反的事实，以适当的方式（建议，注释等）提醒用户。

如图15所示指定的并且用在划分模型文本中的管理器的主要优点是能够在变化的环境下管理资源单元而不须要更新管理单元为此相关的资源单元，在分别发展管理和资源单元时能使用最方便的技术。

下面对一个实施例进行描述，该实施例将涉及到在上述的划分模型中的模型表示是如何存储在被管理的系统内的，技术问题是如何始

终保持管理系统带有能随时间改变的与特定的被管理系统相关的正确的管理信息模型。

在图19给出了根据划分模型驱动的管理系统100，该系统能操作在另外一个系统中的目标，管理系统通过媒介104与被管理系统102进行通讯，为了能够操作目标106，管理系统需要描述被管理系统的信息，问题是管理系统的管理器如何得到这一信息。

如果需要的时候管理系统能读出管理信息，那它就变得更独立于被管理系统，如果管理系统和被管理的系统之间的唯一依赖是管理信息所建立的方式，那么就可能独立地发展两个系统，为了解决这些问题新的目标被引入到被管理系统，因为每次管理系统能够从每个被管理系统读出数据，存储有关被管理系统的数据的最好方式是把它存储在被管理的系统自身内，使用这种新的目标产生新的问题，例如它们是如何构成的和它们又如何和何时装配在系统中等。

在被管理系统中存储描述被管理系统的其它目标信息，是让管理系统能够在任何时间读取这样的信息的一条途径。为了使用一个共同的目标去描述目标的所有种类，这无论如何必须进行分析，为了执行它的任务管理系统须要哪一类的信息。

当在一个被管理系统内构成所有的目标时可以发现，正是该信息描述了必要包括在新的目标内的被管理目标的分类。如果要比较不同分类的两个被管理目标时，这在图20中出现，这就可以看出两种目标类型具有属性、关系和方法，不同的是属性等的姓名以及属性、关系或方法的数目，使用该信息可以形成一般描述被管理目标的样板，该样板包括在被管理目标MIM－服务器内，这也在前面谈到，并在图21中给出，使用样板可以描述每一个能够在被管理系统中使用的

被管理目标，如果装配包括信息的ＭＩＭ－服务器－目标，该信息描述我们同时在被管理系统中装配的一个目标的类型，并且如果管理器系统能够解释ＭＩＭ－服务器的目标类型，这也就可能使管理系统在任何时间内去收集在被管理系统内有关目标的信息。

ＭＩＭ－服务器－目标能包括什么样管理信息的限制仅仅取决于能如何明确指出被管理的系统，因为讨论的被管理目标是自动产生的和由源于目标清单本身的数据添充的。

ＭＩＭ－服务器－分类的结构实例在下面示出，该实例能够容易地读出，而且能用任意的语言所写出，一些缩写语句须要解释：

——ＡＤＴ（Abstract Data Type）＝抽象数据类型

——ＤＤ（Data Domain）＝数据类型

——Persistent＝应当存储在数据库内

```
PERSISTENT ADT Mim IS
    PRIMARY KEY myClassId;
    ATTRIBUTES
    myClassId:  Integer;
    myClassName:    String:="NoName";
    myClassVersion:     String:="1.0";
    myAttributeList:    AttributeArray;
    myActionList:   ActionArray;
    myNoticationList:   NotificationArray;
    myNameBindingList:NameBindingArray;
END ADT Mim;
```

可以发现属性表（myAttributelist）被指定为属性字段(Attri-

buteArray），事实上该字段是动态字段，这意指它没有预先指定的大小，而且随着每个元素的增加而扩大。

```
TYPE AttributeArrayIS
    ARRAY OF Attribute
END TYPE;
```

该字段包括了元素，元素的名称是“Attribute”，因为在一批中的每一个元素将描述在ＭＤ－说明中的真实属性的性质，在仔细地观察属性说明时可以发现，存在如myName等的属性，它们都描述了属性的性质：

```
ADT AttributeIS
    ATTRIBUTES
    myName:          String:="NIL";
    optional:    Boolean:=False;
    myExternId:      IntegerArray;
    myInternId:      Integer:=0;
    myDD:        DD;
END ADT Attribute;

TYPE IntegerArrayIS
    ARRAY OF Integer
END TYPE;
```

当考虑MyDD时可以发现，它定义为ＤＤ，也就是类型，这在下一个说明中得到进一步解释。

```
ADT DD IS
    ATTRIBUTES
    WhichOne:    WhichOne;
    myIntDD:          ItDD              OPTIONAL
    myRealDD:    RealDD            OPTIONAL;
    myTextDD:    TextDD            OPTIONAL;
    myEnumDD:    EnumDD            OPTIONAL;
    myOctetDD:   OctetDD           OPTIONAL;
    myRangeDD:   RangeDD                    OPTIONAL;
    myArrayDD:   ArrayDD                    OPTIONAL;
    myStructDD:       StructDD          OPTIONAL;
    myRefDD:          ReferenceDD                OPTIONAL;
    myExtDD:          ExternalDD        OPTIONAL;
END ADT DD;
```

由于属性是例如整数，实数，字符串等的一种类型，这一信息是必须的，问题是属性在同一时刻不能是所有的类型而仅仅是一种类型，这是为什么须要选取一些不同的类型和提供更多的信息，这由Whichone完成。

```
TYPE WhichOne IS ENUM
     IntDD,
     RealDD,
     TextDD,
     EnumDD,
     OctetDD,
     RangeDD,
```

```
    ArrayDD,
    StructDD,
    RefDD,
    ExtDD
END TYPE;
```

属性的规则类型当然是整数，十进制小数和正文字串，它们能近似地指定的：

```
ADT TextDD IS
    ATTRIBUTES
    isString: Boolean:= True;
END ADT TextDD;

ADT RealDD IS
    ATTRIBUTES
    is32bit: Boolean:= True;
END ADT RealDD
```

这里进一步有其它的共同类型，也不应忘记，例如八位［二进制的］位组，举例为：

```
ADT OctetDD IS
    ATTRIBUTES
    isOctet: Boolean:= True;
END ADT OctetDD;

ADT EnumDD IS
    ATTRIBUTES
    myEnumList: EnumElementArray;
END ADT EnumDD;
```

```
TYPE EnumElementArray IS
    ARRAY OF EnumElement
END TYPE

ADT EnumElement IS
    ATTRIBUTES
    myName: String;
    myValue: Unsigned;
END ADT EnumElement;
```

这儿也还有其它类型，但不是通常用的，但在完整地描述一个属性还是需要的，例如结构类型和字段类型（结构和阵列），也可能指定自己的类型。域（Range）是可能划归到这类属的一个类型：

```
ADT ArrayDD IS
    ATTRIBUTES
    myElement:  DD;
    mySize:          Integer OPTIONAL;
END ADT ArrayDD;

ADT StructDD IS
    ATTRIBUTES
    myAttributeList: AttributeArray;
END ADT StructDD;

ADT RangeDD IS
    ATTRIBUTES
    myMin: Integer:= 0;
    myMax: Integer:=1;
END ADT RangeDD;
```

```
ADT ExternalDD IS
    ATTRIBUTES
    myClassName:        String:= "NIL";
END ADT ExternalDD;

ADT ReferenceDD IS
    ATTRIBUTES
    myClassId:  Integer     :=0;
    myClassName:    String  :="NIL";
    myInverse:  Integer     OPTIONAL;
END ADT ReferenceDD;
```

属性也可以有系统设定值，为了包括这一信息，特殊的“系统设定”类型被指定，这一类型也使用Whichone类型去陈述考虑的是什么系统设定值。

```
ADT Default IS
    ATTRIBUTES
    whichOne:   WhichOne
    myIntDD:            IntDefault      OPTIONAL;
    myRealDD:   RealDefault     OPTIONAL;
    myTextDD:   TextDefault     OPTIONAL;
    myEnumDD:   EnumDefault     OPTIONAL;
    myOctetDD:  OctetDefault    OPTIONAL;
    myRangeDD:  RangeDefault    OPTIONAL;
END ADT Default;

ADT IntDefault IS
    ATTRIBUTES
    myUnSignedValue; UnsignedInteger OPTIONAL;
    mySignedValue: Integer OPTIONAL;
END ADT IntDefault;
```

```
ADT RealDefault IS
    ATTRIBUTES
    my32bitValue: Real  OPTIONAL
END ADT RealDefault;

ADT TextDefault IS
    ATTRIBUTES
    myString: String    OPTIONAL
    myChar: Character OPTIONAL;
END ADT TextDefault;

ADT EnumDefault IS
    ATTRIBUTES
    myValue: EnumElement;
END ADT EnumDefault;

ADT OctetDefault IS
    ATTRIBUTES
    myValue: OctetString;
END ADT OctetDefault;

ADT RangeDefault IS
    ATTRIBUTES
    myValue: Integer;
END ADT RangeDefault;
```

使用属性，方法，通知等可以指定被管理的系统，为指定信息而使用该技术时每一件事能适合MIM－服务器－分类，下面是被存储的信息的各种其它类型的目录：

```
TYPE ActionArray IS
    ARRAY OF Action
END TYPE;

ADT Action IS
    ATTRIBUTES
    myName:          String       :="NIL";
    myInternId:      Integer      :=0;
    myExternId:      IntegerArray;
    myArguments:     AttributeArray;
    myReturnType:    DD;
END ADT Action;

TYPE NotificationArray IS
    ARRAY OF Notification
END TYPE;

ADT Notification IS
    ATTRIBUTES
    myName:          String       :="NIL";
    myInternId:      Integer      :=0;
    myExternId:      IntegerArray
    myArguments:     AttributeArray;
END ADT Notification;

TYPE NameBindingArray IS
    ARRAY OF Name Binding
END TYPE;
```

```
ADT NameBinding IS
    ATTRIBUTES
    myOwnClassName:                String;
    myOwnClassID:              Integer;
    myChildClassName               String;
    myChildClassID:                Integer;
    myChildRDNAttributeName:                String;
    myChildRDNAttributeID:     Integer;
END ADT NameBinding;
```

所有MIM－服务器－目标共同构成了被管理系统的完整的模型，使用目标的这种类型，这就可能使管理系统一件一件地从被管理的系统得到信息，通过读取已经装配到被管理系统内的MIM－服务器－目标的信息，可以在操作期间改变被管理系统（即模型），并且使管理系统更新它对被管理系统的视点而不须要重新编译。

图22给出了MIM－服务器－情况和被管理的目标的分类之间的关系，对于“正常”的被管理的目标的每一类120和122，在被管理的系统130内都存在着“特定”的被管理目标的类128的两个分别的情况124和126，这些特定的被管理目标组成了被管理系统的管理信息模型132的表示。

为了更新被管理系统，仅仅一个新的MIM－服务器－目标须要被装配以反映在被管理系统中的新的目标类型，象对其它的MIM－服务器－目标做的那样，管理系统能以同样的方式收集该目标类型，并且对被管理系统的视图发生了改变而无须重新编译代码，这使管理系统非常稳定和被管理的系统能经常更新而无须担心管理系统的更新。

为了对MIM服务器编码产生的实施例进行详细介绍，早期描述的内容简短摘要在这里参考图2将第一次给出。

产生MIM－server－和MD－码是链，它起始于对MD的说明，比较图23，首先设计者使用特定说明语言说明MD，然后使用特定的编译程序进行代码编译，编译程序用例如C或C++的标准编程语言产生源码。主目标码当然是被管理的系统能够执行的编码，但这也是产生管理信息模型的被管理的目标部分的中间格式的最佳时间，为了能做这些编译程序必须具有MIM－服务器－分类的结构的知识，编译程序考虑MD－说明和计算属性，方法等等的数目，使用这些信息去设计MIM－服务器－情况，所有这些信息被引入到被管理系统的源码中的“分类－方法”之中，并且在装配到被管理的系统后能够自由地执行。

使用所描述的技术保证了它所描述的模型和目标的一致性，这是因为产生的MIM－服务器－信息和目标码源于相同的MD－说明，这也使得负责产生模型信息的设计者较容易地处理它，信息产生的高度自动化的链不仅使设计被管理的目标变得容易，而且使其更快地更可靠地完成。

这里给出MIM－服务器－编码－产生的实例：

```
PERSISTENT ADT MO IS
ATTRIBUTES
time:    Time;
NrOfLinks:  Integer;
LinkName    String;
LinkId:     String;
METHODS
LockLink (IN aValue);
END ADT MO;
```

```
ADT Time IS
ATTRIBUTES
hour:    IntRange(0,23),
minute:      IntRange(0,59);
second:      IntRange(0,59);
END Time;
```

编译程序能为目标产生源码，在C++中它看起来近似如下（可以看到编码并不是严格的，仅作为例子给出，其中仅仅为分类MD给出说明文件）

```
class MO
public:
MO();
LockLink(aValue);
void                set_time();
void                set_NrOfLinks();
void                set_LinkName();
void                set_LinkId();
time                get_time();
int                 get_NrOfLinks();
char*    get_LinkName();
char*    get_LinkId();
void                init();
private:
Time                time;
int                 NrOfLinks;
char*    LinkName;
char*    LinkId;                };
```

由于时间是复杂类型，这里将给出单独的分类：

```
class Time
public:
Time();
void        set_hour();
void        set_minute();
void        set_second();
int         get_hour();
int         get_minute()
int         get_second();
void        init();
private:
int         hour;
int         minute;
int          econd;};
```

现在能够产生MIM－服务器－信息，编译程序将进行MD－说明和一件接一件地设计MIM－服务器－情况，在分类方法下收集信息，这里用mimInit表示：

```
void
MO::MimInint()
// Build the first attribute "Hour"
IntRangeDD tmpIntRangeDD;
tmpIntRangeDD.nyMin(0);
tmpIntRangeDD.myMax(23);
// Set the choice switch to IntRange
WhichOne tmpWhichOne;
tmpWhichOne(IntRangeDD);

// Set the DataDomain values
DD tmpDD;
```

```
tmpDD.whichOne(tmpWhichOne);
tmpDD.myIntRangeDD(tmpIntRangeDD);

// Set the attribute values
Attribute tmpAttribute1;
tmpAttribute1.myName("Hour");
tmpAttribute1.myDD(tmpDD);
.
.
.
// Build the second attribute "Minute"

IntRangeDD tmpIntRangeDD;
tmpIntRangeDD.myMin(0);
tmpIntRangeDD.myMax(59);
WhichOneType tmpWhichOne;
tmpWhichOne (IntRangeDD);

DD tmpDD;
tmpDD.whichOne (tmpWhichOne);
tmpDD.myIntRangeDD(tmpIntRangeDD);
Attribute tmpAttribute2;
tmpAttribute2.myName("Minute");
tmpAttribute2.myDD(tmpRangeDD);
.
.
// Build the third attribute "Second"

IntRangeDD tmpIntRangeDD;
tmpIntRangeDD.myMin(0);
tmpIntRangeDD.myMax(59);
WhichOneType tmpWhichOne;
tmpWhichOne(IntRangeDD);
```

```
DD tmpDD;
tmpDD.whichOne(tmpWhichOne);
tmpDD.myIntRangeDD(tmpIntRangeDD);
AttributeType tmpAttribute3;
tmpAttribute3.myName("Second");
tmpAttribute3.myDD(tmpIntRangeDD);
.
.
```

到目前为止仅仅产生了这样的属性，即Time是由什么构成的，以及为什么现在也必须产生时间属性本身并且向其指定其它属性。

```
AttributeArray tmpAttList;
tmpAttList.add(tmpAttribute1);
tmpAttList.add(tmpAttribute2);
tmpAttList.add(tmpAttribute3);

StructDD tmpStructDD;
tmpStructDD.myAttList(tmpAttList);
WhichOneType tmpWhichOne;
tmpwhichOne(StructDD);
DD tmpDD;
tmpDD.whichOne(tmpWhichOne);
tmpDD.myStructDD(tmpStructDD);

Attribute tmpAttribute4;
tmpAttribute4.myName(myTime);
tmpAttribute4.myExternId("OID");
tmpAttribute4.myInternId(1);
tmpAttribute4.optional(FALSE);
myDD(tmpDD);
```

```
// Now myTime is done, the next tribute to do is NrOfLinks.

// Next attribute to build is NrOfLinks

IntDD tmpIntDD;
WhichOneType tmpWhichOne;
tmpwhichOne(IntDD);
DD tmpDD;
tmpDD.whichOne(tmpWhichOne);
tmpDD.myIntDD(tmpIntDD);
Attribute tmpAttribute5;
tmpAttribute5;myName/NrOfLinks);
tmpAttribute5;myExternId("OID");
tmpAttribute5;myInternId(2);
tmpAttribute5;optional(FALSE);
tmpAttribute5;myDD(tmpDD);
.
.
// The compiler keeps building attributes this way
// until all attributes and so forth is created
// After that it creates the MIM instance and
// assigns the different attributes to it.

// The MIM instance is assigned the same name as the
// classid of the class the information represents
AttributeArray tmpAttList;
tmpAttList.add(tmpAttribute4);
tmpAttList.add(tmpAttribute5);
tmpAttList.add(tmpAttribute6);
tmpAttList.add(tmpAttribute7);
```

```
Mim 24337;
2433.set_myClassName(myMO);
2433.set_myClassID(24337);
2433.set_myVersion(1,0);
2433.set_myAttList(tmpAttList);

//
24337.set_myNotificationList(tmpNotificationList);
24337.set_myNameBindingList(tmpNameBindingList);    }
24337.set_myActionList(tmpActionList);              }
//
}
```

分类现在准备由一个普遍编译程序来编译，在此基础上它能装配在被管理系统中，MimInit方法已准备执行，并且将用所有引入的信息产生ＭＩＭ－服务器－情况，正哪前面早些时候所描述的，首先检查软件然后装配到被管理系统。

为了能执行完成ＭＩＭ－服务器－分类情况的方法，这当然须要ＭＩＭ－服务器－分类已经装入到被管理系统，ＭＩＭ－服务器－分类应当是被装配到被管理系统中的第一类。

MimInit方法是分类方法，并且仅仅打算被执行一次，因此它并不是一个管理系统能执行的一般方法。

当决定被管理的系统应当用新的分类更新时，在使用前必须仔细地测试软件，当做完每一件事并且分类应当是清楚的时候，在每一分类中，为了更新，被管理系统软件执行MimInit方法因为每一个分类有它自己的MimInit方法，在每一分类的一个情况将在数据库内进行装配，在每一个ＭＩＭ－分类已经做完一个情况后传送一个那一类已

经被装配完的一个注释，管理系统亦即这些注释的用户当然获得该注释和决定应该做什么，是更新旧的ＭＩＭ－服务器－信息，还是不考虑注释。

使被管理系统的模型表示存在于被管理系统本身之中，而不是在管理系统中对其编码，具有这样的优点，即它总是与其所描述的被管理系统一致。同样地，管理系统能够被联接到任意的被管理系统，读出它的模型表示和能够管理它，这也可能让被管理的系统具有描述ＭＩＭ－服务器－分类本身的ＭＩＭ－服务器－分类，因为它是装配在被管理系统中的，与其它分类类似的一个分类。这可以被引导到一个管理器，该管理器首先要读出描述ＭＩＭ－服务器－分类及其版本等等的ＭＩＭ－服务器－分类－情况，并且使用这些信息决定在系统中的ＭＩＭ－服务器－分类－情况应该怎样解释。

现在将描述管理系统怎样存取和解释存储在被管理系统内的模型信息。

为了能够在管理系统和被管理系统间传送数据和清除信息到它最初的形式，这就要求接收系统或者准确地知道传送什么和以什么样的序列，或者信息能被自我鉴别，通常使用标记，或换句话说使用标识成分，然而无论使用什么方法，总是需要一些共同的信息或规则，经常称为协议书来做这些。

因为使用不同个数的位或甚至相同个数的位对不同的数据类型进行编码，当接收到的位串被解释时，重要的是要准确知道在每一个数据类型中包括了多少个比特位，这通常在这样的正文中是个问题，许多不同的协议书是可行的，为了在不同的手段中选取传送和再生数据的手段，需要考虑到，信息可能是相当复杂的，并且一旦信息在一个

接收系统内，它必须以可用的方式进行重构。

模型信息因此应该表示在通讯双方都知道的结构内，为了编码和解码简单的数据类型成为比特串，在被管理的系统内为其内部通讯所使用的编码和解码系统在这儿描述的实施例中使用。为了避免数据出错，联系双方都必须知道该编码和解码系统。

为了传送简单的数据类型为可通讯的比特串，具有涉及这些比特串编码和解码的策略是基本的，在这儿使用的方法是让整数类型使用32位去表示它的值，使用表示串内字符数目的32位整数对一个串进行编码，其中每个字符用8位表示。

为了把简单的数据类型编码成可通讯的毕特串，每个数据类型具有移位的方法，它简单地示出了发生了什么。

```
Code                Decode
int>>bitstring      bitstring>>int
float>>bitstring        bitstring>>float
char*>>bitstring            bitstring>>char*
char>>bitstring         bitstring>>char
boolean>>bitstring  bitstring>>boolean
.                               .
.                               .
.                               .
```

通过对简单的数据类型使用编码和解码原理，这就可能为更复杂的数据结构产生编码和解码方法，这可以由执行一系列的简单的编码和解码方法来实现：

```
complex                    Buffer operator >>(Buffer buf, complex X)
{                  {
int a;                 buf>>X.a;
char*b;                    buf>>X.b;
}                      }

                   Buffer operator<<(Buffer buf,complex X)
                   {
                   buf<<X.a;
                   buf<<X.b;
                   }
```

通过使用为复杂数据结构而设计的编码和解码的这一技巧，这就可能设计几乎所有类型的编码和解码方法。

当管理系统启动读操作指向被管理系统情况时，必须指出它想读什么样的属性，使用较早描述的方法编码属性中的数据，传送到管理系统，并在那里解码，如果接收系统具有和传送系统同样的编码和解码方法，接收者就知道使用了什么方法对该信息进行了编码，这就不会有任何困难去读出复杂的信息。

为了能够解释ＭＩＭ－信息，ＭＩＭ－服务器－分类编码和解码程序被合并在管理器内。这样，使用ＭＩＭ－服务器－分类构成的情况可以解释从被管理系统接收的信息，这样毕特位从数字缓冲器中移出进入到ＭＩＭ－服务器－情况。

管理系统在图１５中的解释部分模型解释器中包括ＭＩＭ－服务器－分类，当管理系统从被管理系统的ＭＩＭ－服务器－情况中读出属性值时和接收带有编码数据的缓冲器时，通过把毕特位从缓冲器中移出并送到ＭＩＭ－服务器－分类的空的局部情况内时，很容易地完

成了解码，由于ＭＩＭ－服务器－分类具有把数据从缓冲器移出或移入的方法，这就变成很容易的任务，主要问题在于解释其它的分类而不是ＭＩＭ－服务，在该情况下就必须使用ＭＩＭ－服务器－信息以解码信息。

进一步的解释来于实施例，该实施例借助于图２４－２８详细描述如下。

为了解释仅通常ＭＤ读出的属性值，管理系统需要描述ＭＤ－分类的ＭＩＭ－服务器－信息，它起始于产生ＭＩＭ－服务器的空的情况以及从通常的ＭＤ读属性值中读出的属性的处理（UsrMO）。

1. MIM local_MIM_Link;//产生一MIM-服务器情况

2. UsrMO local_UserMO_Linki//产生一个UsrMO情况

3. get (MIM, Link, myAttList) ;读模型信息

上述三步骤在图２４中示出

4. Buffer>>local_MIM_Link. myAttList; //付本被填上信息。

管理系统能够审视局部表示的ＭＩＭ－服务器并且调查分类联系，下一步是以它所感兴趣的ＭＯ－目标中推演情况数据，这个过程相当复杂，但能够在下列步骤中实现。

从图２５可以看出，管理系统是如何通过链Ａ推导出情况数据。

5. get(Link, A.all)；读出所有的属性值。

现在从Ａ来的情况数据在缓冲器表内等待被解码，管理系统将使用移位操作以获得正确的毕特位的数量和把它们解码成正确的数据类型。

通过研究局部的ＭＩＭ－服务器去确定第一属性是什么样的数据类型，第一属性可以从缓冲器中获得和进行解码。

图 2 6 示出了表示MO－分类的局部MIM－服务器是如何被用来构成MO－情况的第一局部表示。

6．当管理系统到达数据类型器期时，将发现它以形成结构和须要深入结构内以进一步进调查。这在图 2 7 中给出。

7．解释器研究“日期”结构和通读属性表以研究属性类型，通过使用该信息就能够解释下一个缓冲器，这在图 2 8 中示出。

8．解释被执行，并且管理系统能向外部用户表示属性值，图 2 9 给出怎样解释数据和下一个属性类型从局部MIM－服务器表示中读出。

这里现在将要描述，前面较早讨论的管理信息模型是怎样能被做出决定的，使用这个我们这里意指可以为计算机编程，读出它的说明并且解释：

——被管理的系统可以采用什么样的状态（从管理的系统来看感兴趣的状态）

——可以对被管理系统采取什么操作。

——可以对特定状态的被管理系统采取什么操作。

——当特定的操作指向它时，被管理的系统应进入什么状态。

为了能被称之为是可决定的，管理信息模型必须能够以机器可解释的语言表述，在传统的技术内最经常使用自然语言去表达什么操作能指向被管理的系统和当特定的操作指向它时被管理的系统应进入什么样的状态。

通过使用可决定的说明，可以达到诸多优点：

——由于在特定的操作后能预测被管理系统的新状态，一般的管理器能具有更强的处理能力，同时它也能提出这样的操作，该操作能

被允许在特定的状态和／或导致所希望的状态。

——当所期待的执行被很好地指定时，简化了被管理系统的执行和检验及（不是一般地）管理的应用，这也可能以说明中产生大部分的执行码。

——因为仅仅导致在被管理系统的状态变换的操作被接收，操作的耐久性改善了。

——管理信息模型的早期模拟和求值都简化了，这使得说明任务得以简化。

——在构形在激活态被取消以前，通过允许相当自由的操作序列但又始终保证被管理系统的完整构形，能够构造管理系统的模型使其既耐用又容易被使用。

这里将确定第一可能的状态。

在一特定时刻被管理系统的状态是这样指定的：即在这一时刻在该被管理系统中有什么被管理目标情况以及这些情况有什么属性值。

为了定义特定被管理系统所能采取的状态，必须给出定义：

——能够存在的被管理目标的什么情况

——它们具有什么属性

——这些属性具有什么值

从图 3 0 的第一行给出的说明来看，很明显可能存在分类用户的被管理目标，该分类被管理管目标具有属性，行 3 － 8：

——NumLer，它描述了用户数目，

——AdmState，它描述了管理状态，

——Opstate，它描述了操作状态，

——UsageState，它描述了使用状态，

——Line，它为用户物理终端的行驱动器提供参考。

通过这样的说明扩大了被管理系统的可能的状态空间，新的状态空间由具有所有可能组合的属性值的新的潜在的用户组成。

根据图31行24，28，32，37的属性类型决定潜在的属性值。

现在描述可能的操作是怎样被定义的。

指向被管理系统的操作产生、取消、操作和检查被管理目标的情况并且能改变被管理系统的状态。

产生操作产生被管理目标的所述分类的新情况。

取消操作取消被管理目标。

写操作改变被管理目标的分类的情况的属性值。

方法操作为被管理目标的分类调用情况的方法。

读操作返回情况的属性值。

对被管理目标的所有类而言，读、写、产生和取消操作是通用的，并且具有相同的句法和语义规则，产生和取消操作对被管理目标的给定分类而言被规定为可能或不可能。

读操作总是可用于被管理目标的类的每一属性，因此在定义属性和它的类型的被隐含地指定了。

写操作总是可以自由选取，它被指定用来定义可以被更新的属性。

方法操作是为被管理目标的一个分类所特用的操作的换码程序。这些操作是作为方法执行的，每一种方法被定义为指向受控目标的分类的情况的方法，每一个方法接收一个参数表作为输入数据并且返回一个结果。

图32定义了操作，行61－64定义如下的操作：

——属性Line、Number、AdmState、OpState，UsageState，的读操作，行62－63，

——属性Line的写操作，行61，

——LockRequest方法，行64。

该被管理的目标被定义为没有消除和产生操作，行65。

下面要描述的是，允许的操作／状态组合是如何被指定的。

取决于在一个被管理系统中的被管理目标的情况和它们的属性值，在特定的状态下存在着唯一组允许的操作。一个例子是，如果情况是在使用中（由它为属性所描述），它就不能被取消，另一个例子是，因为存在着执行相依赖的限制，该限制指出必须仅仅是9个情况，因此就不允许产生被管理目标的某个分类的第10个情况。

通过先承条件和／或端点条件，一般而言，有两种方法去指定这些操作／状态的组合。

为了使操作能被通过，先承条件描述什么样的条件必须被执行，在本情况下它为每一个操作陈述了为了接收指令,被管理的系统必须是什么样的状态，先承条件组成了被管理目标的类的逻辑部分。

在我们的带有目标分类Subscriber的实例中，在用户被解除之前（这由属性AdmState=unlocked来指示），我们希望防止属性行是空的，如所述，Line属性是为用户物理终端的线驱动单元提供的参考。当Subscriber被使用时，希望这个参考存在。

图33的行79和80示出这是如何表示的。

终端条件指出在更新事务处理后什么样的条件必须被执行，在本例中，对于被管理目标的每一个分类，可以指出分类的情况必须是怎样状态才能处于一种相容的状态，终端条件逻辑上是被管理目标的一

个分类的一部分或是一组这样的分类，后者对目标之间相互依赖时是有效的。

在具有目标分类Subscriber的本例中，如果属性Line是空的，我们希望防止属性Adm状态是开启的。当用户被使用时，我们希望确保后面的参考存在，在图34中这点在行106和107上表示。

终端条件能使用两种方式中的一个实现。管理系统以这样一种方式更新被管理系统，从而使终端条件得以满足。在这种情况下管理器有责任去完成条件，如果管理器忽略了这个职责，它将拒绝更新被管理系统的事务处理，完成终端条件的另一方法是通过自动完成必要的更新让被管理系统维持限制。

在第一种情况下管理器的策略是由管理信息模型说明决定的，从该说明可以决定在当前状态下什么操作是允许的，在另一方式中随后的操作由被管理的系统自由地执行。

在图34的实例中可以考虑这两种策略的任一个，如果属性行是空的和AdmState=unlocked，AdmState 就能设置为锁定，然而在该例中的终端条件阻止了这第二种策略，第一个策略因此必须被选取，在被管理系统的更新事务处理被委托前，检查终端条件，管理系统就能承担实现条件的职责，图35的行124，125示出了这是如何被表达出来的。

图36的行134示出了第二种策略能如何被指定，致使被管理的系统维持着它的相容性。

到目前为止，仅仅已经讨论了在目标内调节相容性限制的终端条件，然而也必须有可能指出指定目标之间的相容性限制的终端条件。

在图37中指定了目标LineDriver．该目标具有与图33中目标

Subscrible的关系，调节两个目标之间相容性限制的终端条件现在必须能够被指定。

在图34的行118和图37行163上指定了两个目标组成了同一个相互依赖的方案LineAnd-Subscriber的部分，这个相互依赖的方案在图38中给出，图38的行169－171中的依赖方案指出，如果Subscriber被锁上，Line Driver也必须被锁上。

这儿随后将详细描述在操作完成后如何获得状态的定义。

在执行操作后能决定被管理系统的新状态的问题的部分解答是由使用上述终端条件和维持这些条件的规则共同组成。

一个问题仍然存在着，即如何决定方法的顺序和产生操作，为此目的通过联接终端条件到方法和产生操作，使得概念终端条件被扩展，并且稍微加以改变。

在本例中，已经和用户一起定义了先承条件，它指出AdmState必须被锁定为一个接受管理器腾空属性行的先决条件，管理器应该能够知道AdmState怎样被锁上的，因为属性AdmState仅仅能被读取，方法LockRequest事实上是应该被用来达到所希望效果的方法。

图39示出这是怎样在行176－178上指定的。

这些终端条件和那些是单个或成组被管理目标的分类的界限的终端条件之间的区别是，条件总是由被管理的系统维持。

图40给出另一个情况，其中指出，在目标产生后，属性Line并不等于Null。由于属性Line是另外目标的参考属性，由说明可以决定，用户目标本身以另一方式产生或发现目标LineDriver，并且让属性Line参考这一目标。

这随后将详细描述如何决定和实现可决定的管理模型，在这个联

系中，在其它文本中已经提到的语言C++将投入使用。

管理模型包括了目标和关系，这在图4 1中示出，其中给出了4个目标类型，它们之间存在着直接的或间接的关联，在这样的模型中常常在目标之间存在着相容性限制和依赖，例如在图4 1中，为了能够操作，Trunk目标2 0 0依赖与它相关的Port目标2 0 2的状态，这样当Port目标中操作状态的属性（opState）被减活时，Trunk目标也应当被减活，此外在Trunk中的管理状态、属性admstate和Port目标之间也存在类似依赖，类似的依赖关系也能应用到图4 1的其它关系上。

为了使管理信息模型可做出决定，应该能够清楚地陈述这样的依赖关系，而后，管理系统能预测被管理系统的特定更新操作的结果。

在存取数据库的不同应用中，实现这样的依赖关系导致了维持应用的困难，并且不能担保数据库的相容性，因此维持依赖关系和相容性限制的机构应该在数据库系统内执行，维持与存储在数据库的信息相关的相容性规则的逻辑应构成数据库的部分逻辑而不是存储数据库的应用，在这种方式下每一规则仅仅在一个地方指定和实施。

在引入的方法中，前面已经描述的发明的一个方面和它的各个方面，特别是比较图3和图4和与此对应的正文，并参考图4 2－4 4中的方框图和功能图，通过这些应用部分给出概括说明。

被管理系统2 0 4的管理信息模型由具有属性、方法、关系和相容性规则的一组被管理的目标组成，通过ＤＯＬ－逻辑（数据库目标逻辑）和不变的数据库目标，这些目标的大部分能在其它事情中实现，许多被管理的目标（例如那些表示硬件资源的）也包括静态过程208，比较图4 2，该静态过程用来从表示目标的物理装置中存取和接收信

号。

数据库目标能包括通信应用中存取的、并且在管理系统的角度来看是看不见的数据。被管理的目标并不真正作为实际目标来执行，而且仅仅构成实际实施目标的视图２１０，真实目标２０６能被认为是源目标，源目标是包括了管理和通信管理的功能的目标，比较图４２。

每一个源目标２０６包括了一组操作，该操作组成了目标的管理图，一般媒介２１２取存各自目标的管理图，这些管理图共同组成了管理信息模型２１４，该模型２１４下面是完全信息模型２１６。

每一个被管理的目标是一个特殊的类型，目标类型为这一类型的所有目标定义共同的性质，也就是属性、方法、关系和相容规则，对于每一个目标类型都存在着一个源媒介２１８（图４４），该媒介包括了这一目标类型的逻辑，目标类型的每一情况是由数据库的逻辑，目标类型的每一情况是由数据库目标２２０所表示，该数据库目标包括了情况的不变的数据值，对于每个被管理的目标这里都存在着源媒介２１８和数据库目标２２０（在其它事物中共同作为静态过程）。

在图４４当中，一般媒介２１２通过ＭＤＳＴ接口２２２（被管理目标服务接口）存取资源媒介２１８。资源媒介２１８作为目标情况，资源媒介ＤＯＬ逻辑２２４通过ＤＭＩ接口２２６（数据库管理接口）存取情况数据库目标。

解答包括了某种机器可以解释的语言以指定带有特性的不变目标（在概念模型化范围内建立的和指向目标）为属性、方法和关系，并且解答也包括了上述描述的承先和终端条件。

在下面描述和详细解释承先和终端条件时使用了句法，但它并不是现存语言的组成部分，仅用来通过实例的手段在此澄清原理，在前

面描述的图３０－４０和图４５中使用了伪句法，下面的描述部分地重复结合图３０－４０的上述所讲的内容，虽然更一般化。

图４５的实例指定了名命为AnObject的目标类型，行１，它包括了三个不变的属性：attr1，attr2和attr3（这就是存储在数据库中的值的属性），这些属性能接收的值的类型分别是Atype、Integer、和Integer，4-6。AnObject包括两个方法，命名为ｍ１和ｍ２，这些方法分别有类型变元Argtype和AnotherArgtype，行９－１０，方法m_2返回类型Areturn Type的值，行１０。

图４５的AnObject具有前承条件，该条件指定当方法应被执行时attr２的值必须等于attr3的值，行１３，终端条件陈述了attr2的值必须大于或等于attr3的值，行１６，当在数据库执行处理时这是不能违反的限制。

图４５目标类型AnObject给出了先承条件和终端条件的简单的例子，当条件涉及到多个相关的目标类型时就出现了终端条件略复杂的例子，这可通过例子给出，该例结合了两个目标类型：Resource A和Resource B（这两个目标间的依赖性极类似于图４１ Trumk和Port间的依赖性）。

参看图４６，它给出了资源Ａ的说明，这个目标类型包括了两个状态说明：admState和opState属性admState描述了目标是否在操作中，行２３，而在另一边的opState描述了是否目标是可操作的，行24，这里也有所述的参考属性Brep，这些参考属性执行目标之间的数据库关系，在数据库中ResourceＡ的所有情况将包括与在属性Brep内的相关的Resource B的情况有关的参考，行２５。

图４６的前承情况指出，为了移动Reaource A的情况，admState

必须等于locked（这意指目标必须不被使用），行３２，图４６的终端条件陈述了，如果它相关于Resource B的目标，Resource A的情况仅仅能在操作中（admSTate=unlocked），行３５。

在图47中可以发现Resource B的说明，它包括了参考属性,Arep,它包括Resource A中的Bref的倒数，行４５，从图４７中可以看出，Resource B具有和ResourceA一样的承先条件，行５２。

如前所述这里存在着与Resource A和Resource B相关的终端情况，这些终端情况能认为是两种目标类型之间的依赖关系，一个这种依赖关系相关于包括几个目标类型的一个子系统。

在图４８中指定了依赖方案，它指定了相关于Resource A和Resource B的终端条件，第一个条件陈述了，如果它相关的Resource B 目标是“locked”时，类型Resource A的目标不能被“unlocked”，行62和63，第二个条件陈述了，如果在Resource B目标中的opState的值是去激活的，相应资源A的opState的值必须不是激活的，行65和６６。

在图４９给出了概念模型，它示出了终端／承先条件之间的关系，以及可以应用的有关概念，图４９仅谈及它本身而没有详细解释其它部分。

承先和终端条件的语言必须是清楚的和无歧意的，这里将给出这些概念的精确定义，在管理信息模型内指定终端条件，终端条件描述了静态相容规则，该规则在被管理系统内是不能被违反的，该被管理系统适配于管理信息模型，更精确而言，终端条件以应用到被管理系统的数据库，终端条件相关于存储在数据库中目标情况和它们的属性值，比较图４９。

如前所述结束条件例如能陈述属性值之间相依关系，另一个例子是属性和数据库关系之间的重要性，也就是相关于一个属性的值的数目的限制，它也能是相关于一个目标类型的情况的数目的限制。

必须适用于一个指出某个限制的特定终端条件的数据库可能永远不会进入到违反这些限制的状态，当操作更新存储的信息时诸数据库状态变换出现了。

更新操作在事物处理时执行，一个事物处理由一序列操作组成，事务处理能被考虑为原子的，以要么所有的操作都执行或者什么都不执行的方式进行，这是由委托和回卷式操作执行的，如果每一个操作都是成功的才执行事务处理，在事务处理时如果任何一个出错它将卷回（也就是说所有的操作都没有做）。

当事务处理结束时要必须保证数据库具有相容的状态，数据处理仅仅处理数据目标的副本，在数据处理被做之前真实的诸目标并不更新，这就意指在事务处理期间能暂时违反相容性规则，当事务处理被执行时，必须决不违反限制。

前承条件陈述了，当特定操作应被执行时，必须完成一个的规则，比较图 4 9，更精确地说，数据库应处在这样状态，那里，在事务处理开始前必须实现该规则。

承先条件能够用来限制数据库的一系列操作，状态变换的限制也能表示出来。

这里现在随后详细地描述执行机构是怎样被指定的。

当考虑到终端条件时，如前所述，这里存在着两种可以替换使用的执行原理，一条可替换的方案是事务处理的相容性检查，该检查是在进行事务处理之前进行的，仅当没有违反终端条件时执行事务处理，

否则它将卷回。

另一种可替换的解决方法是自动改正措施，这能随触发操作执行，更精确地说，当特定属性已经更新后，在这个特定的事件下触发了操作。

图 5 0 的说明示出了怎样能指定这些可以替换的执行机构，第一个终端条件－相关于各自目标的属性admState－被指定，以在事务处理被做前进行相容性检查，行 7 3 － 7 7 。

第二个终端条件被指定为部分执行，并且具有自动校正，当去激活被指定给Resource B的opState时操作也传送给在ResourceA中的opState，当去激活被分派给Resource B的opState时，必须做相容性检查，行 7 9 － 8 9 。

如果完全地自动执行图 5 0 的终端条件是希望的，执行就略微复杂一些，在Resource A中的属性opState能够被认为是可推导出来属性，它依赖于Resource A的内部状态和Resource B的状态，图 5 1 示出了在Resource A的opState怎样能被指定为可推导出来的属性，行 9 2 － 9 5 ，opState值是从两个其它属性的值当中计算出来的，该两属性分别是internalOpState和ResourceBstate，该属性Resource-Bstate是ResourceB中的opstate的变换，每一次在ResourceB目标中的opState被改变，它应沿着传送到相关ResourceA目标的Resource－Bstae,行117－118，这能在示于图 2 中的ResourceAandB的相关方案中被指定。

从这里开始将详细地描述维持终端和承先条件的机构是如何能在 C＋＋ 内执行的，通过上述刚刚描述的执行机构的说明，执行能自动地产生。

首先处理终端条件。

相容性检验的执行是在各自的目标类型中编译的，这就意指每一个目标是能检验相关于诸目标的所有的限制，在事物处理执行之前，每次当事务处理操纵目标时，目标的相容性检验应当执行。

图 5 3 给出了事务处理的步骤，首先根据 2 5 0 执行所有的操作，然后依照 2 5 2 所有被操纵的目标必须执行相容性检验。如果没有违反相容性限制，在 2 5 4 事务处理结束，否则在 2 5 6 它卷回。在相容性检查前必须完成所有的更新操作。

为了示出相容性检验的自动产生的执行，可以使用图 4 6 的例子，如果没有指定维持特定终端条件的规则，应按缺席形式执行相容性检验，在图 4 6 的ResourceA的终端条件应当执行相容性检验，在C++示出了自动产生的执行码，目标类型Resource A 这样被编译成C++的类，该类的说明文件在图 5 4 中给出。

事实上，图 5 4 的类是资源媒介的实施，通过调用静态方法open（行 1 3 4 ），以主key值为参数，该资源媒介能变换情况为数据库目标，对于资源目标（admState，opSTate和Brd）的每一个不变的属性，这儿存在着两个写和读方法去写和读数据库目标的属性值，对此数据库目标资源媒介已经变换情况。

在该文本中在图 5 4 中感兴趣的是行 1 3 7 的方法，这是在做事务处理前为检验相容性规则而调用的方法，在图 5 5 可以看到这个方法的实施，当相关于几个有关目标的终端条件被考虑的时候(如图48)相容性检查必须在每个目标内执行。

现在描述触发操作和传送。

为了示出如何产生自动校正的执行我们能够使用图 5 0 的说明，

行82陈述了，当去激活被指定给opStae时，在ResourceB的opState值应当传送到Resource A的opState，Resource B的说明文件可以在图56中看到，主要兴趣分别是方法setOpState和propagateOpState，行198和203。

如图57所示，当属性opState由激活变到去激活时，在方法setOpState的方法propagateOpState被调用，这意指在从激活到去激活的状态变换时触发了propagateOpState，该触发操作传送OpState的新值到相关的ResourceA目标。

当特定的操作并执行时，前承条件必须是有效的条件，和终端条件相反，在事务处理时在讨论中的操作执行之前必须决定先承条件，在进行自动校正测量时前承条件很难维持，这样它们仅能和相容性检验一块执行，和终端条件的其它不同是，前承条件必须存取存在数据库的原始属性，而不是事务处理的付本。

图58示出了在执行事务处理操作时的算法如图58在260所示，当违反前承条件时事务处理卷回。

图59的说明是由图46的说明产生的，图59的说明包括了对相关于dalete Object操作的前承条件求值的方法（行268）。当deleteObject操作被执行时该方法应被触发。

图60示出了自动地产生deleteObject和在ResourceA内delete-ObjectCondition方法的执行，在目标真的被移动之前在deleteObject内调用deleteObjectCondition，使用方法admStateUatebaseValue deleteObjectCondition读出存在数据库内的admState的值，如果条件是admState必须被locked，那么例外情况就被抛弃，当发现例外时事务处理就卷回，另一方面如果条件是有效的，通过在deleteObject内

被称为reallyUelete方法目标情况从数据库内取消。

上述的优点如下：

执行码可以从由管理系统解释的同样清楚化的说明中自动地产生（编译）。模型说明的修改将自动地更新信息库执行者的观点和及其执行，被管理的系统因此能保证按着管理系统预测的方式执行。

每一个相容性的限制仅仅需要指定一次，执行和包含在相容性限制的属性装在一块，在这些存取数据的应用中不需要执行相容性检查，编码的维持也就极大地简化了。

系统的所有的相容性限制以一致的方式执行，可靠的编码自动地产生。

执行的策略是以分散的方式执行的，从而依赖性是以所包括的目标之间的通讯保持着，对被管理的系统而言这儿不存在在引入新目标类型时（具有新的依赖性）必须修改的集中化的功能。

包含在事务处理中的诸目标可以瞬时处在不相含的状态（在执行相容性检查前）这简化了以这样方式的数据库的管理，在事务处理中的操作次序就不那么重要了。

这里所描述的解决方案简化了执行不协条发展的系统成分之间依赖关系，使用分别编译和加载的系统成分的解决方案成为可能。

这里将详细地描述不协条执行系统成分的例子。

被管理的系统由几个子系统组成，更具体的说由一个系统前台和大量的应用组成，这些子系统不协条的发展和分别进行加载。

在发展应用或系统前台时，存在着可重用成份库，在不同的子系统内这些成分以不同的方式进行合并和组合。

上述的两个不同问题具有特定的共同性质，这儿存在着不协条发

展的系统成份，但这儿也始终存在着成分之间的依赖关系，为了保持这种依赖关系，这就须要诸成分能够与其它的单独发展的和甚至加载的诸成份进行通信。

下面详细描述的设计过程是能够使用带有简单问题的两个不同的文本。

第一个问题涉及重新使用库成分。

在设计被管理系统时，存在着可重新使用成分的库，这些成分能够合并到被管理系统的诸目标里，如图 6 1 所示，存在着包含有基本函数成分例如MOstate Component 2 6 2 和WorkingStateComponent 2 6 4 的主库 2 6 0，这些成分包含了对被管理目标的许多不同类型是共同的状态属性，带有更加特别的功能的其它诸成分重新使用这些基本成分的功能，在通讯管理库 2 6 6 内，存在着例如StateProgpayationComponent 2 6 8，该成分有能力把在MOstateComponent 262 内的状态变换传送到相关的从属目标 2 7 0 中。

应当强调的是，成分象被管理的目标一样应当是在面向语言的目标内执行的，诸成分和被管理的诸目标是真正的诸目标，在应用时成分并不作为它们本身能够识别的目标存在，它们仅能形成被管理的诸目标的一部分。

执行取决于其它基本成分——例如在图 6 1 的缺席处理和通讯管理库诸成分——的成分的直接了当方法是包括或是通过继承或是通过集合的基本成分，然而这产生了许多问题，这取决于这样的事实，即几个成分，例如FaultHandlingCompanent 272和ResourceManagement Component 274，它们是继承和集合了相同的基本成分，例如MOstate-Component 262，能够被包括在相同的目标路径 2 7 6 内，比较图62

这就意指，将存在被管理目标的基本成分的几个付本，如果基本成分包括了数据，将产生相容性问题，从外部对诸成分而言这将是可见的。

和MOstateComponent包含在里边的诸成分相比，对被管理目标的其它功能性来说MOstateComponent的数据换句话说应是可得到的。

这里存在着避免上述问题的方法。

MOstateComponent不应被依赖该成分的诸成分继续和集合，不使用从属成分，而是将参考属性包括在MOstateComponent中，在包括自己在内的那些目标中，后者将成为它自己的成分，那些须要取存MOstateComponent的诸成分的每一个包含了一个ＭＯ－状态成分的相同情况的参考，比较图６３。

如前面较早所述，例如ResourceManagementComponent 274和FaultHandlingComponent 272的成份能包括一个功能，它由MOstateComponent 262 的状态变化触发，后者必须能够传送状态变化信息到其它的诸成分，一个问题是，在设计MOstateComponent时，这些信息的接收者是未知的，MOstateComponent 262必须以一些方式通过通讯线路和任一的后来出现的成分联系，比较图６４。

第二个问题涉及分层系统结构。

被管理的系统能在分层的结构内执行，更具体地说，这些系统能执行那些被不同诸应用重复使用的基本功能，比较图６５，系统前台２９０包括了不同类型的共同通讯服务，应用能派遣许多任务到系统前台，系统前台２９０和诸应用２９２分别地加载，这就必须可能加载应用把它联到系统前台而无须再加载前台。

系统前台２９０和应用２９２两者都包括了被管理的目标，在前台２９０的目标２９４和２９６能例如表示为资源，开关，传送路径，

主干等等。在应用292.1及292.2的目标298，300，和304能被相关与系统前台的资源合作，比较图66，应用的一目标能委派一些相同的任务到系统前台的一个目标，在应用中的诸目标能这样依赖于能够去工作的系统前台的相关的诸目标，在发展应用时由于系统前台是已知的，为与系统前台的诸目标进行通讯设计应用的诸目标是没有问题的，如前早些时候所述诸应用的诸目标能够依赖系统前台的诸目标，这暗示系统前台的诸目标能够传送状态变化（如前面早些时间所述）到诸应用的诸目标，这暗示前台的诸目标应能够相关于和初始化到调用的诸目标，该诸目标在设计前台系统时是未知的。

现已描述了上述具有类似问题的两个例子。这将描述这些问题是如何加以解决的。

更具体地说，两个例子中任一个的直实问题是设计一个能与未知的将来的诸目标进行通讯的目标，这必须可能使最初的目标与未知的目标相关和合作而不修改或甚至再加载最初的目标。

用图67示出的设计原理可以解决该问题，最初目标330被设计的与抽象的目标332合作，抽象目标定义的接口由未执行的诸方法组成，这些是能被最初目标调用的方法，在设计指定与最初目标合作的目标334时它必须继承抽象的目标332，并且执行它所继承的诸方法，在与目标的未知类型合作时，最初的目标330将考虑这里所说的抽象类型332，在调用抽象目标332的接口所定义的方法时，通过后来联系将委派调用执行真实的目标330。

使用动态联系能实现在加载将来目标时避免再加载最初的目标。

图68示出了怎样使用刚刚描述的设计来设计基本的库成分，MOstateComponent340被设计与诸目标通讯，该目标继承了抽象的目标

MOstateSubscriber 3 4 2，当状态改变发生时这个抽象的目标定义了由MOstateComponent调用的诸方法，如果成分应同意在MOstate-Component内的状态变化，它必须继承MOstateSubscriber 342和执行回答各自的状态变换通知，答复成分当然也必须陈述它本身为Mor-tateComponent的门户。

参考图 6 7 描述的设计原理也能用来进行和应用特定诸目标合作的系统前台的目标设计，这在图 6 9 中给出，这里目标PlatlormOb-ject1 350 被设计为与应用系统 3 5 4 的目标 3 5 2 合作，应与目标PlatlormObject1 350合作的应用的目标必须继承抽象目标Interta-ceObject 356. 进而在PlatlormObject 350和 IntertaceObject 356之间存在数据库的关系，这暗示继承IntertaceObject 356 的诸目标能够和PlatlomnObject 350 建立关系，为了使不用再加载系统前台就能加载应用，使用了动态联系。

应用系统的目标经常是依赖系统前台的另一个目标的状态，当目标执行时，和相互知道的目标一样，该设计使得实质上以同样的方式执行状态依赖关系，这将在下面进一步详细地描述。

两个不协条的目标依赖在实质上和两个协条的目标之间的依赖一样以同样的方式能被指定和执行，这在前面已经描述了，这里将以同样的例子给出设计和执行不协条目标之目的依赖关系，然而这里先假设目标类Resource B 属于前台系统，参看图 7 0，这样各种应用系统的各种目标能够与目标ResourceB 370相关和合作，目标类型Resource B 370通过数据库关系与UserObject 372 相关，ResourceA 374继承了UserObject 372 能够与Resource B 370相关，在下面将示出这些目标是如何指定和执行的，将使用如上的同样的伪句法。

在图 7 1 中可以找到目标ResourceB的说明，说明包括了两个状态属性：admState和opState，一个分配ResourceB的情况和一个前承条件，该条件陈述了，为了能够取消ResourceB目标它必须被locked。这实质和前者完全一样，这里唯一的差别是，在该例中ResourceB与UserObject相关而不与ResourceA相关，关系是通过参考属性user-Ret建立的，行 6 。

在图72内目标UserObject被指定了，它包括了状态属性admState必须注意的是，该属性是作为纯虚拟被指定的，行 1 9 ，这暗示在事实上UserObject将不包括属性admState，UserObject的接口将不包括未执行的该属性的读和写的方法，这些虚拟方法的真实执行将出现在UserObject的各自子类型中，此外这还存在另一种属性resourceBstate被假设用来保持相关目标ResourceB中的opState的值，进而，UserObject具有参考属性Bref，用它来建立和类型resourceB的诸目标的联系行 2 1 。

在图 7 3 中可以找到指定包括目标UserObject和目标ResourceB的终端条件的依赖方案，行 2 9 － 3 1 的终端条件陈述了UserObject的admState以如下的方式依赖依赖 Resource B的admState，如果相关的目标Resource B是被locked，那末目标UserObject不能被locked up.

图 7 3 中进一步地指定当目标ResourceB的opState的值变化时，新的值将传送到UserObject的相关的情况，可以注意到，图 7 3 的说明和图 5 2 的说明之间存在着差别，在图 7 3 中，不存在相关于opState的指定终端条件，仅仅陈述了Resource B中的opState的变化应当传送到UserObject的属性resourceBstate，行 3 6 和 3 7 ，这暗

示，UserObject的所有子类型并不必须是依赖ResourceB的opState，但是UserObject的每一个类型能以自己的方式管理依赖ResourceB的opState。

在图74中示出了ResourceA的说明，ResourceA指定为UserObject的子类型（行41），这暗示ResourceA继承了UserObject的属性、方法、条件和在Userobject中的传送，UserObject的所有纯虚拟的说明必须在ResourceA内指定和执行，如图67所出现的ResourceA的opState值是从属性internalOpState和resourceBrtate（它从User-Object中继承的）推导出的，行46－49。

如前所述，该实施过程与当目标属于同一个系统时实质上是相同的，差别是使用ResourceB的从属目标的功能性的特定部分是在User-Object内执行的。如前一样编译程序的说明能自动地产生编码，在图72中的UserObject的说明产生的说明文件可以在图75中找到。

UserObject中的CheckConsistency方法检验admState属性之间的依赖性，该诸属性是在图73的依赖方案中所描述的，在图76中给出该方法的执行，检验该条件必须读出UserObject的admState的值，这解释了为什么在Userobject中有admState的纯虚拟的说明，即使该属性并不真正包括在UserObject内时也是如此。

在图77内可以找到由图71 ResourceB说明所产生的说明文件，它几乎和图56的相同，然而执行方法propagateOpstate是稍微不同，比较图78，新值被传送到相关的userObject情况的属性ResourceBstate中，在更新目标ResourceB和更新UserObject情况时必须检验User-Object和ResourceB之间的admState从属。这样该条件在ResourceB的checkConsistency方法中执行了。

图79示出了由ResourceA说明中产生的解释文件。通过从resource-Bstate和internalOpState中推导opState的值，执行ResourceB的opState从属，比较图 8 0。

上述的优点如下。

在不协条的诸目标之间的依赖与合作能被指定和执行，以和在同一子系统的诸目标之间依赖的指定和执行方式一样，这暗示它们对管理系统而言是可见的并且能被管理系统解释。

上面示出了以受控和可见的方式维持不协条的诸目标之间的依赖的过程，简化了分层结构的可决定的管理模型的执行。

当组合成分被考虑时，库成分可重用的程度通过高度的娄活性加以改善。

图　1

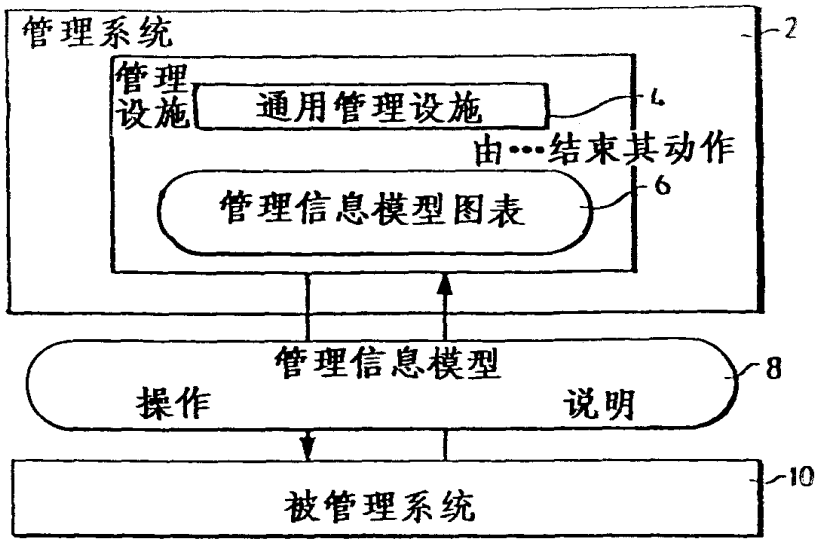

管理系统
2
管理设施
通用管理设施
4
由…结束其动作
管理信息模型图表
6
管理信息模型
操作
说明
8
被管理系统
10

图　2

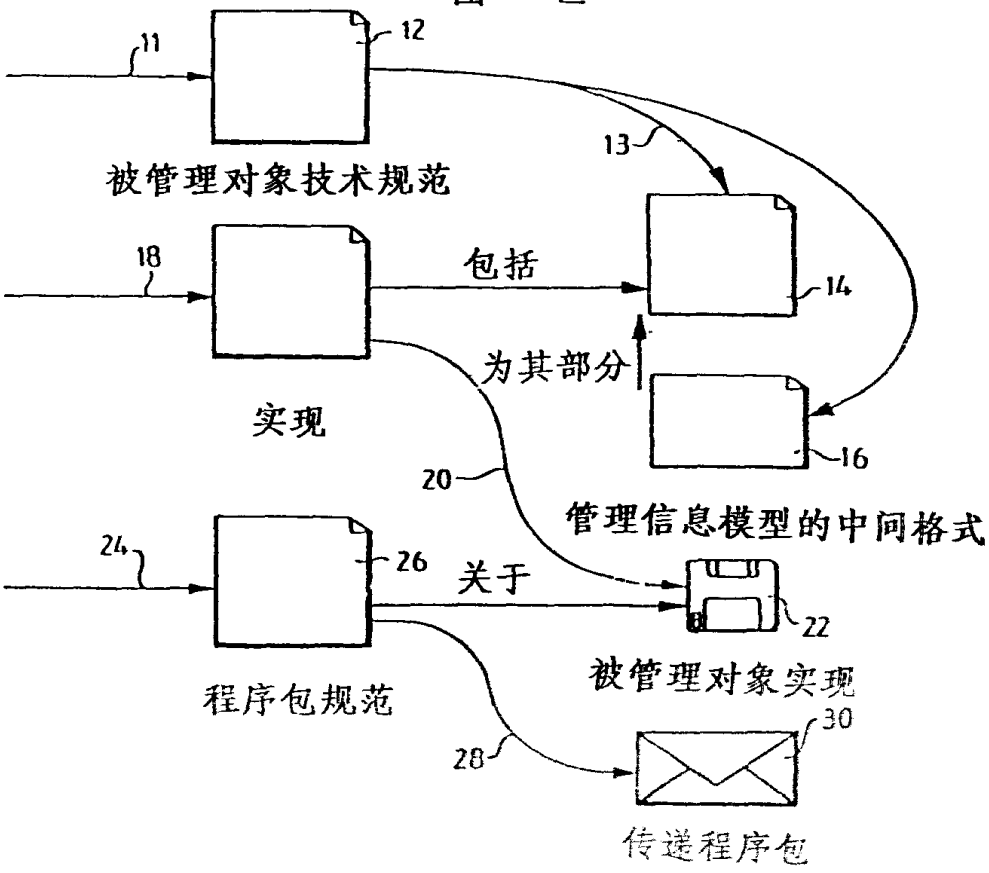

11
12
13
被管理对象技术规范
18
包括
14
为其部分
实现
16
20
管理信息模型的中间格式
24
26
关于
22
被管理对象实现
程序包规范
28
30
传送程序包

图 3

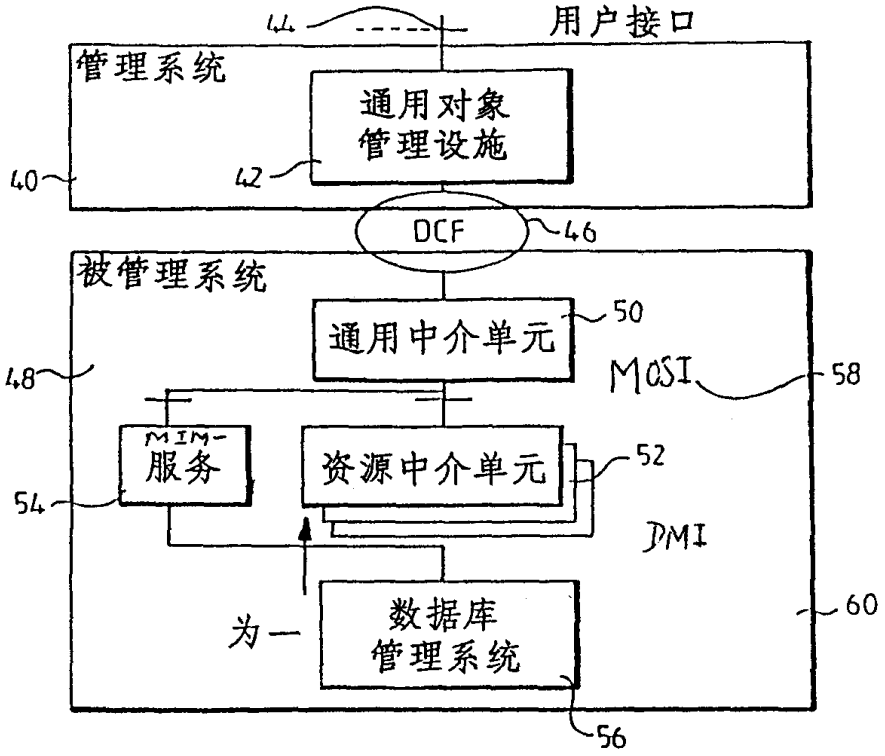

44
用户接口
管理系统
通用对象
管理设施
42
40
DCF
46
被管理系统
通用中介单元
50
48
MOSI
58
MIM-
服务
资源中介单元
52
54
DMI
为一
数据库
管理系统
60
56

图 4

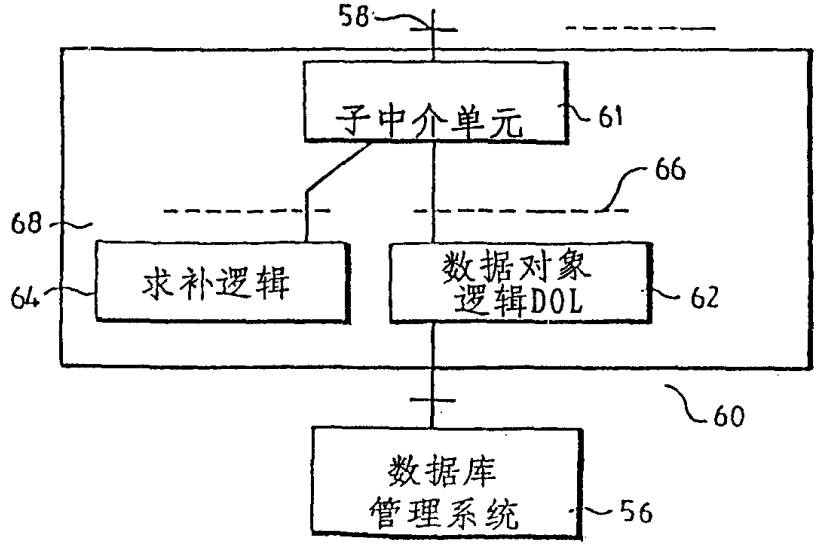

58
子中介单元
61
66
68
求补逻辑
数据对象
逻辑DOL
64
62
60
数据库
管理系统
56

图 5

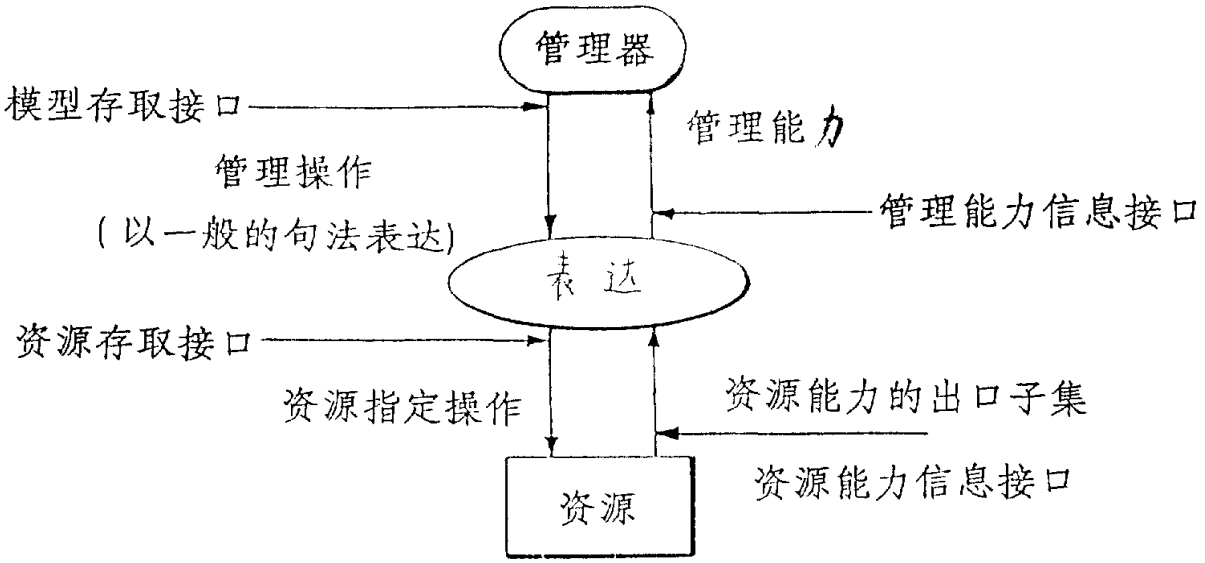

管理器
模型存取接口
管理能力
管理操作
(以一般的句法表达)
管理能力信息接口
表达
资源存取接口
资源指定操作
资源能力的出口子集
资源
资源能力信息接口

图 6

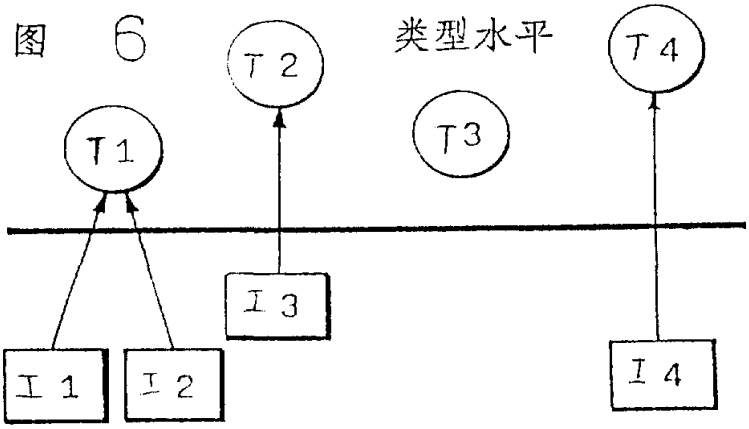

类型水平
T1
T2
T3
T4
I1
I2
I3
I4
情况水平

图 7

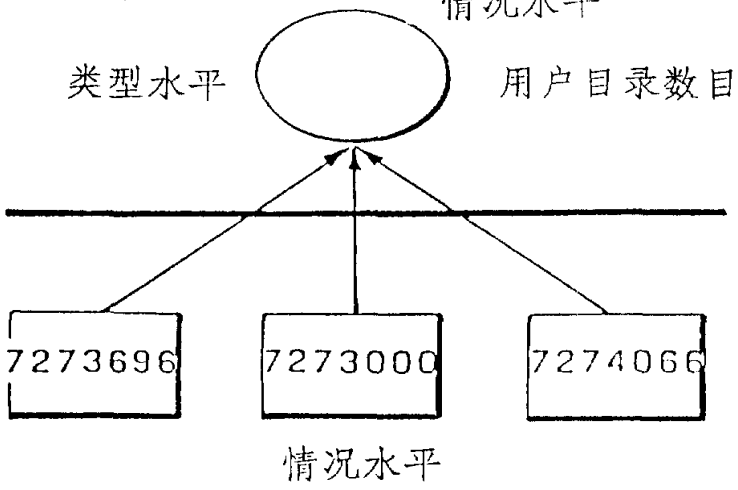

类型水平
用户目录数目
7273696
7273000
7274066
情况水平

图 8

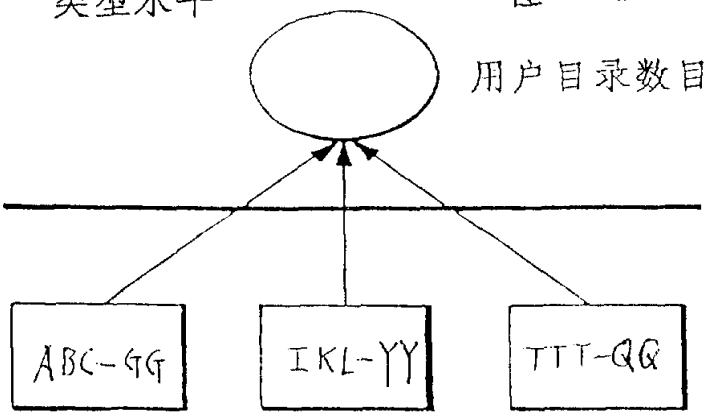

类型水平
用户目录数目
ABC-GG
IKL-YY
TTT-QQ
情况水平

图 9

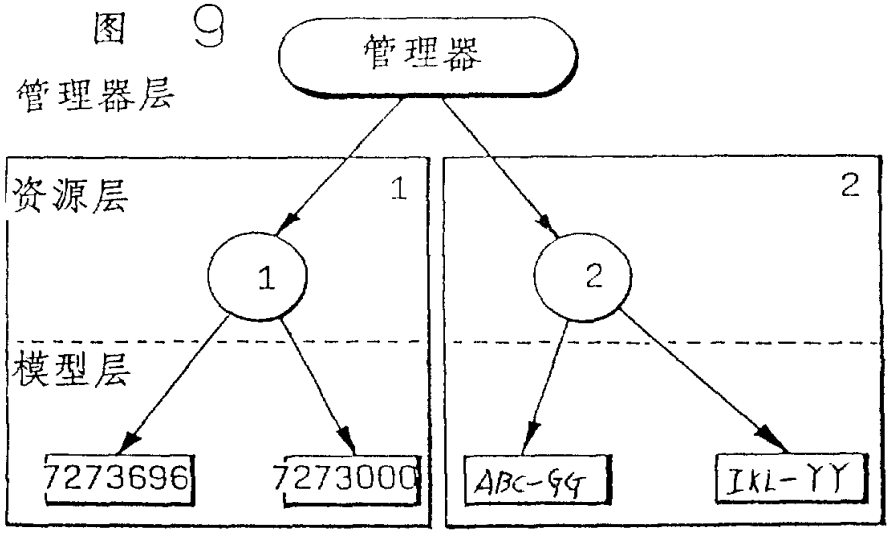

管理器层
管理器
资源层
1
2
1
2
模型层
7273696
7273000
ABC-GG
IKL-YY

图 10

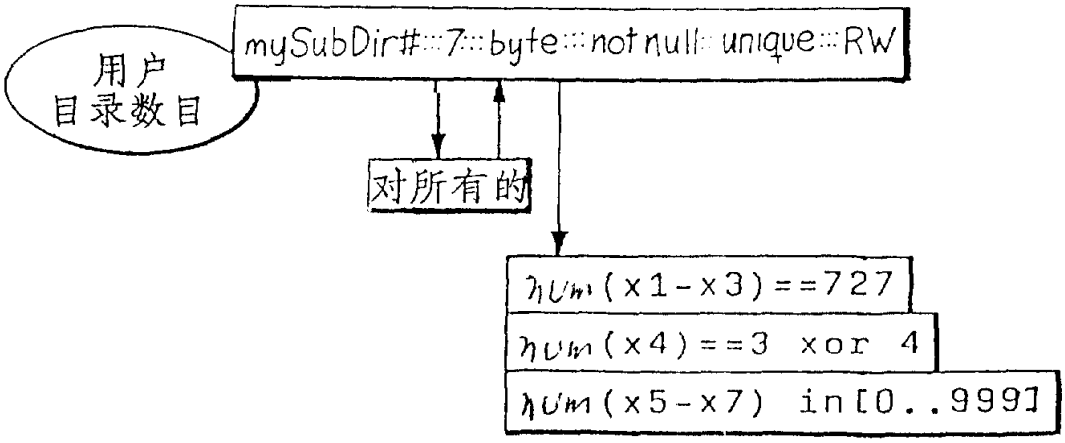

用户
目录数目
mySubDir#:::7:::byte:::not null:::unique:::RW
对所有的
num(x1-x3)==727
num(x4)==3 xor 4
num(x5-x7) in[0..999]

图 11

写(my Sub Dir#, 7273696)

图 12

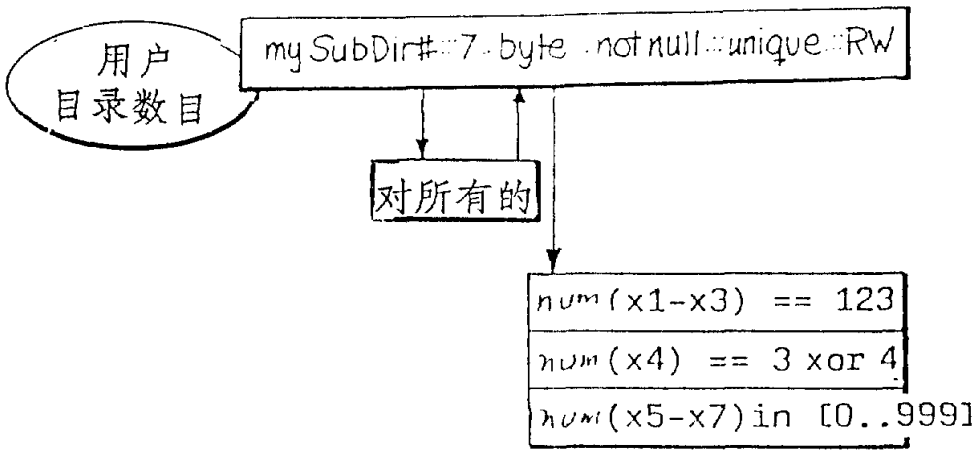


图 13

写(my Sub Dir#, 5553212)

图 14

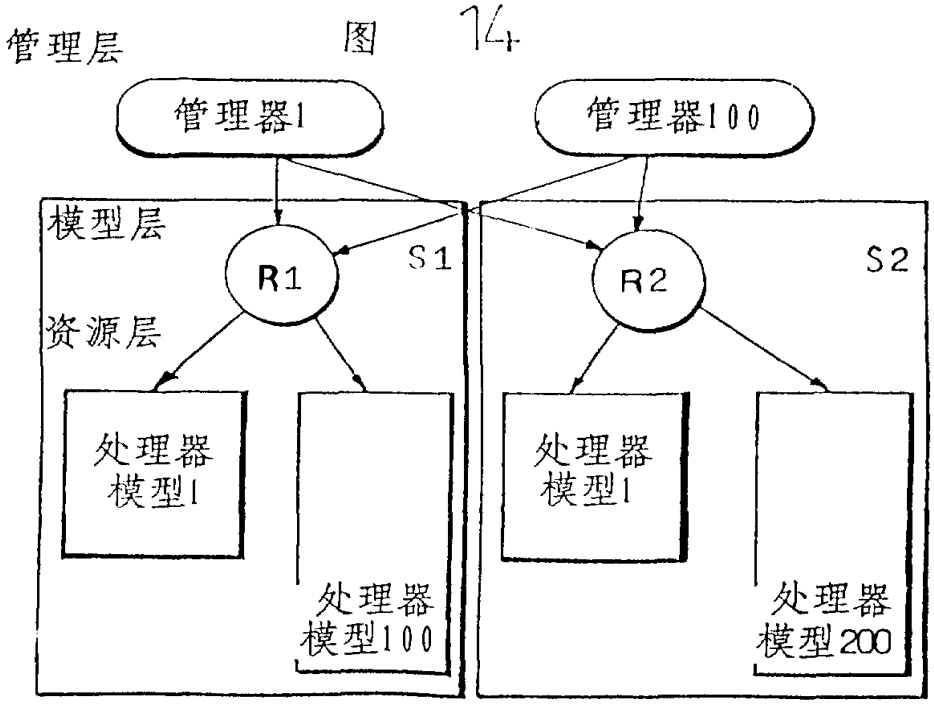


图 15

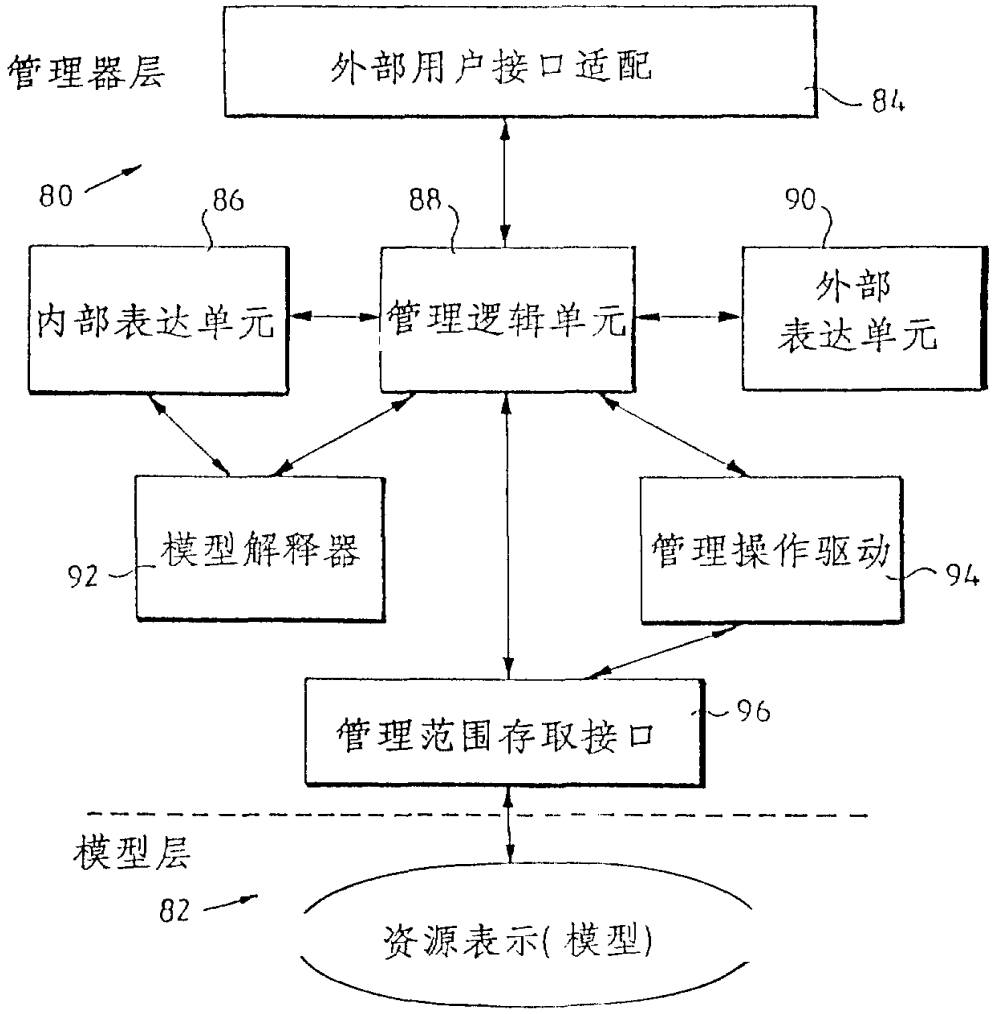

管理器层
外部用户接口适配
84
80
86
88
90
内部表达单元
管理逻辑单元
外部
表达单元
92
模型解释器
管理操作驱动
94
管理范围存取接口
96
模型层
82
资源表示(模型)

图 16

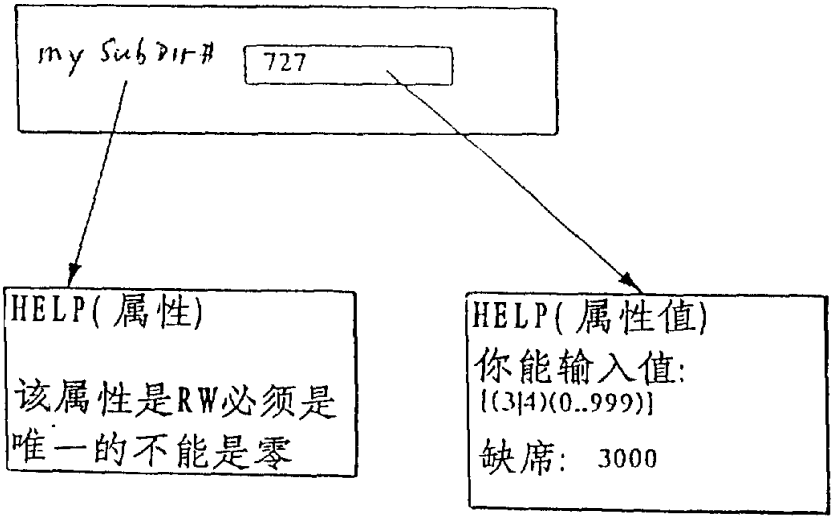

my SubDir#
727
HELP(属性)
该属性是RW必须是唯一的不能是零
HELP(属性值)
你能输入值:
{(3|4)(0..999)}
缺席: 3000

图 17

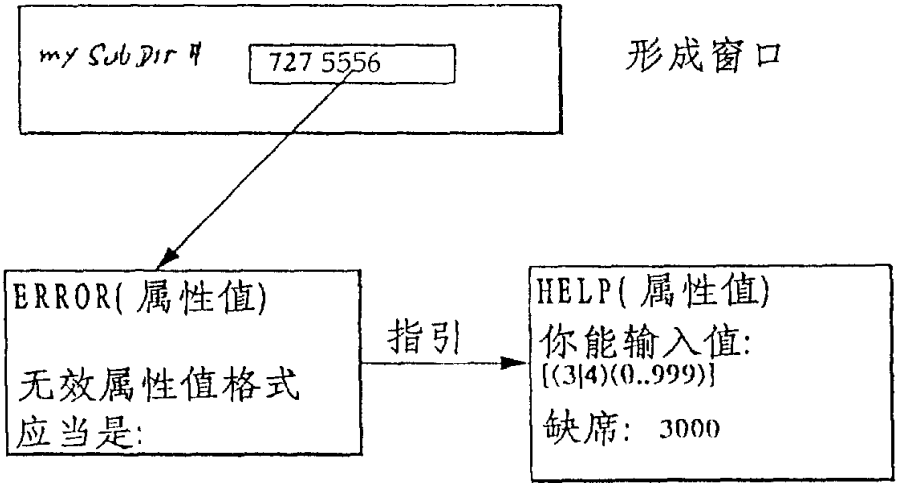

my SubDir #
727 5556
形成窗口
ERROR(属性值)
无效属性值格式
应当是:
指引
HELP(属性值)
你能输入值:
{(3|4)(0..999)}
缺席: 3000

图 18

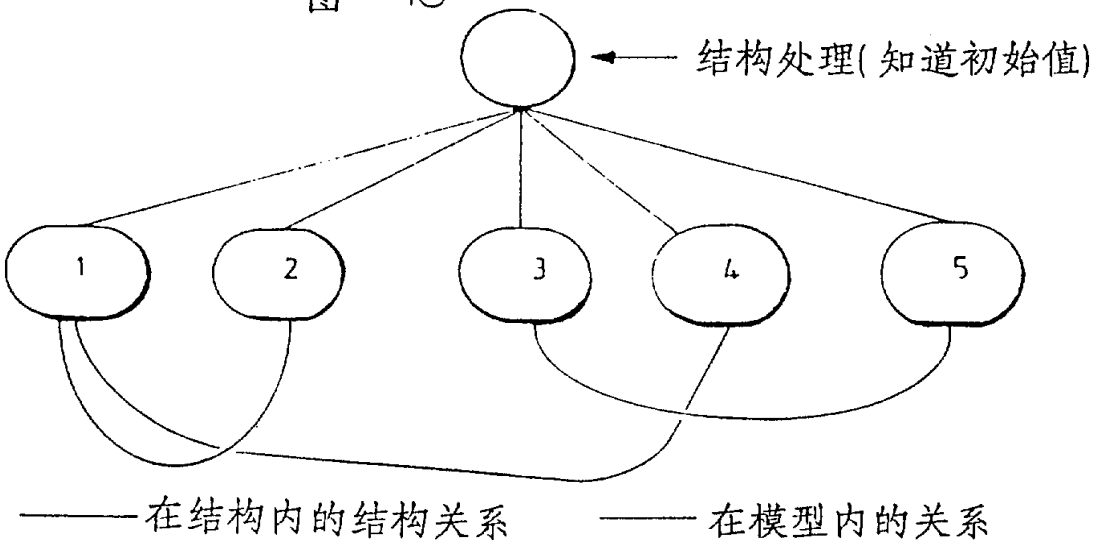

结构处理(知道初始值)
1
2
3
4
5
——在结构内的结构关系
——在模型内的关系

图 19

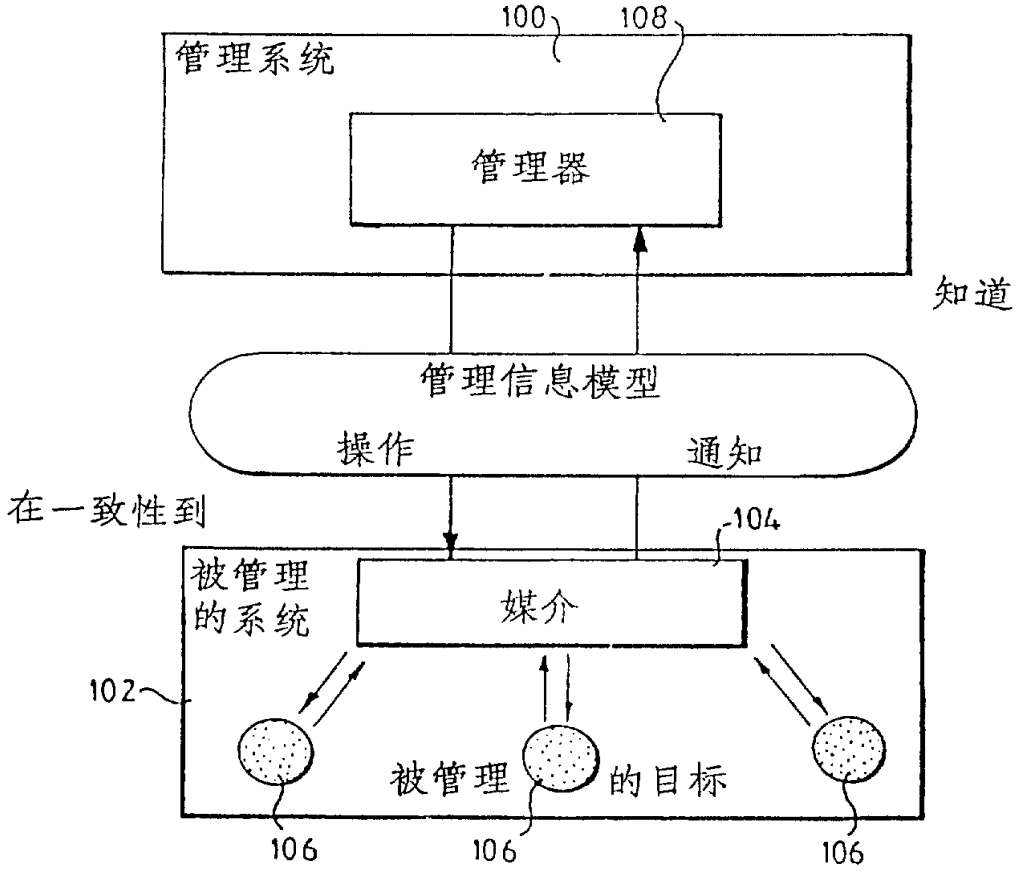

100
108
管理系统
管理器
知道
管理信息模型
操作
通知
在一致性到
104
被管理的系统
媒介
102
被管理的目标
106
106
106

图 20

目标类型A

属性没有链接

关系目标B

行动打开链接
闭合链接

目标类型B

属性没有链接
链接I D

关系目标A

行动发送数据

图 21

属性

属性目录行动目录
关系目录等等

目标类型MI M服务

图 22

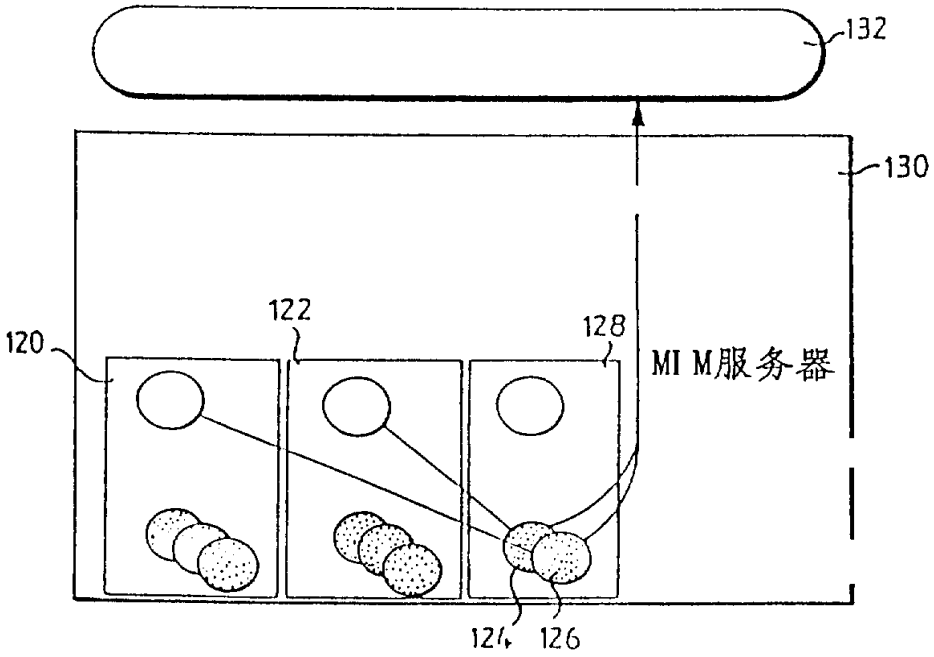


图 23

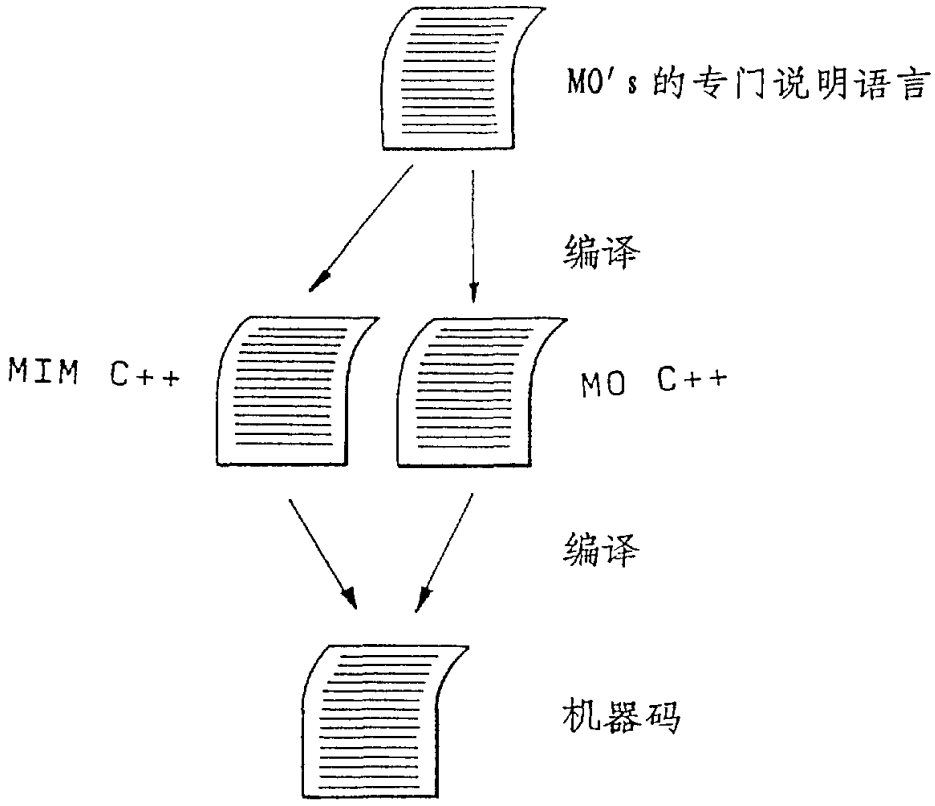


图 24

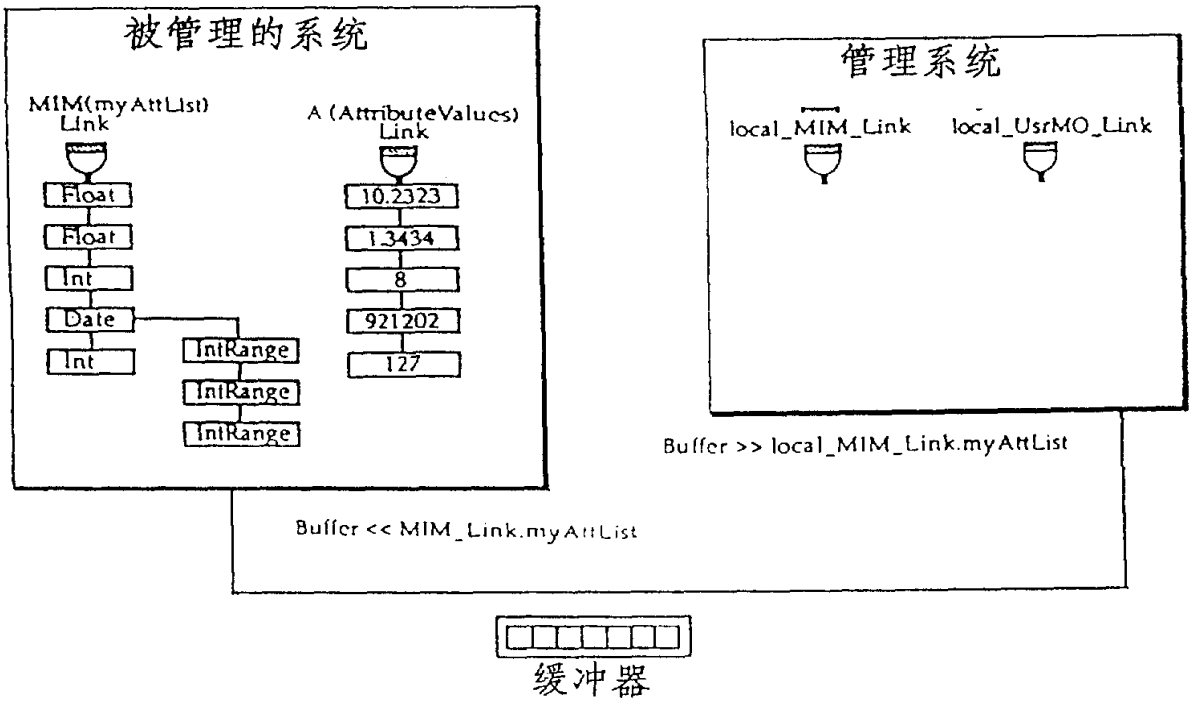

被管理的系统
MIM(myAttList) Link
A (AttributeValues) Link
Float
Float
Int
Date
Int
IntRange
IntRange
IntRange
10.2323
1.3434
8
921202
127
管理系统
local_MIM_Link
local_UsrMO_Link
Buffer >> local_MIM_Link.myAttList
Buffer << MIM_Link.myAttList
缓冲器

图 25

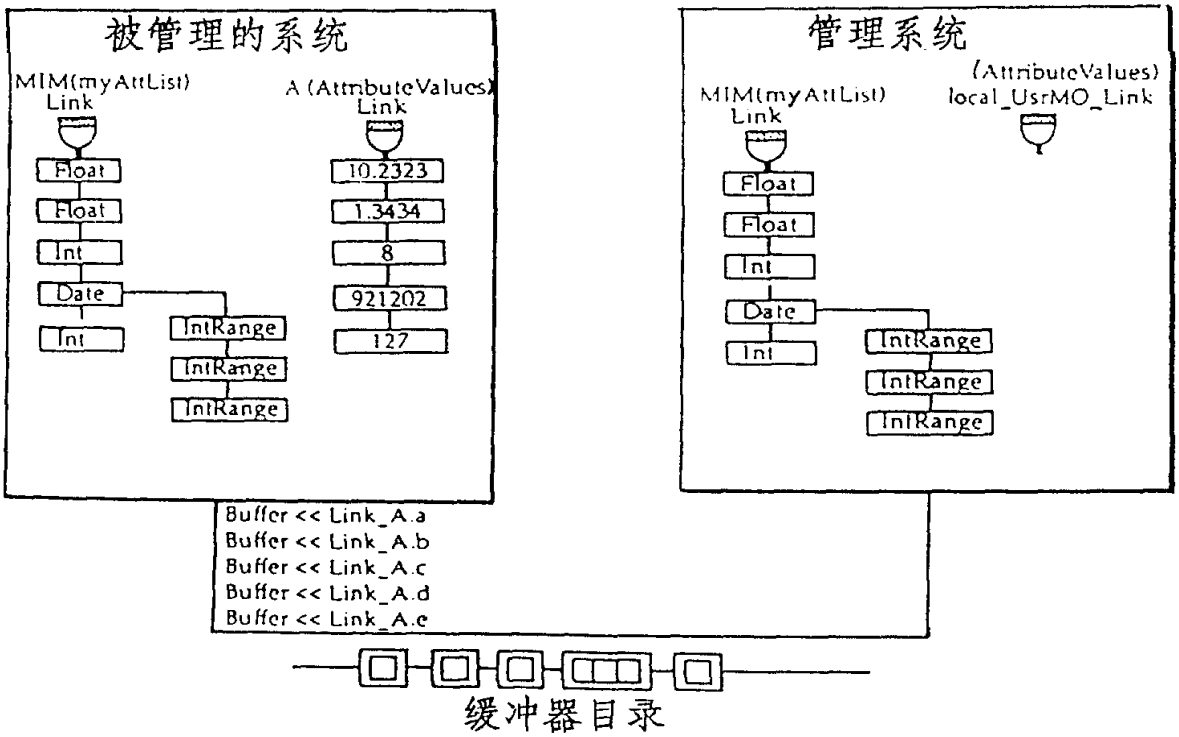

被管理的系统
MIM(myAttList) Link
A (AttributeValues) Link
Float
Float
Int
Date
Int
IntRange
IntRange
IntRange
10.2323
1.3434
8
921202
127
管理系统
(AttributeValues) local_UsrMO_Link
MIM(myAttList) Link
Float
Float
Int
Date
Int
IntRange
IntRange
IntRange
Buffer << Link_A.a
Buffer << Link_A.b
Buffer << Link_A.c
Buffer << Link_A.d
Buffer << Link_A.e
缓冲器目录

图 26

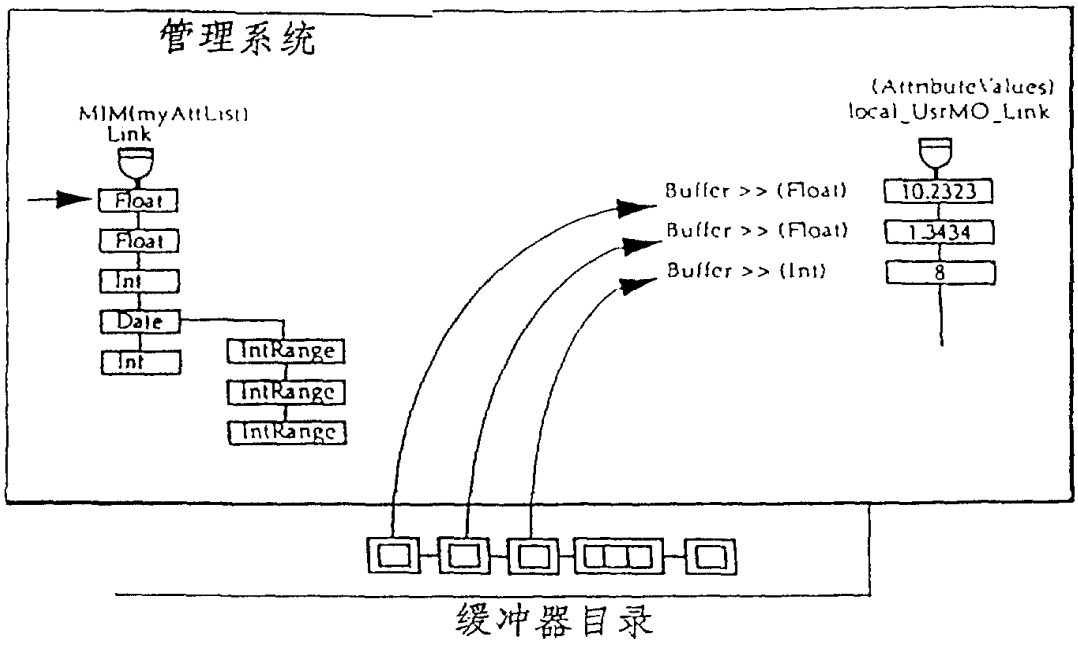

管理系统
MIM(myAttList)
Link
Float
Float
Int
Date
Int
IntRange
IntRange
IntRange
(AttributeValues)
local_UsrMO_Link
Buffer >> (Float)
Buffer >> (Float)
Buffer >> (Int)
10.2323
1.3434
8
缓冲器目录

图 27

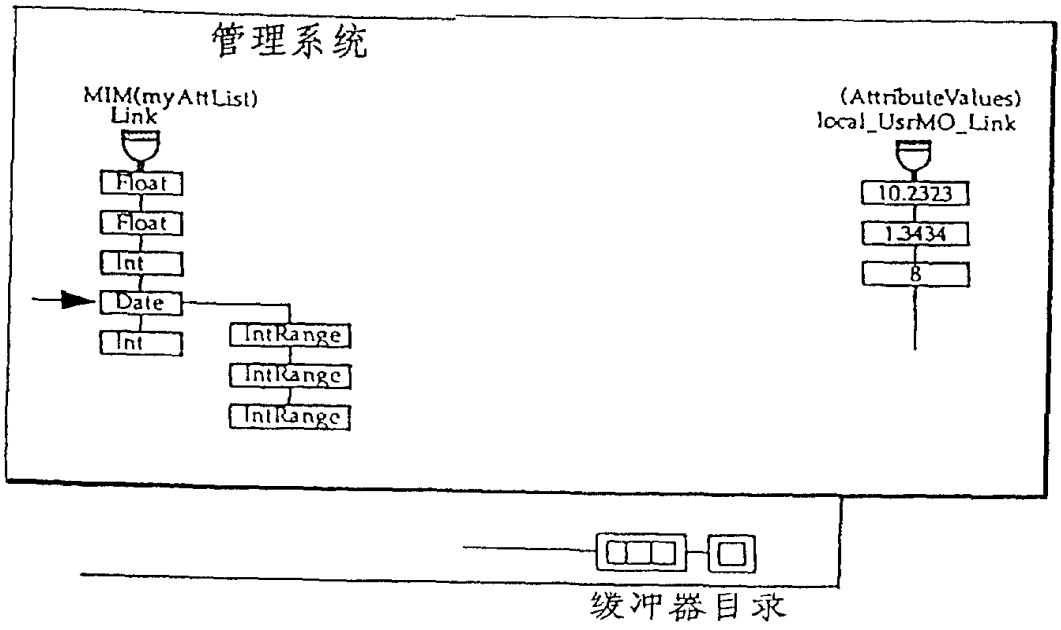

管理系统
MIM(myAttList)
Link
Float
Float
Int
Date
Int
IntRange
IntRange
IntRange
(AttributeValues)
local_UsrMO_Link
10.2323
1.3434
8
缓冲器目录

图 28

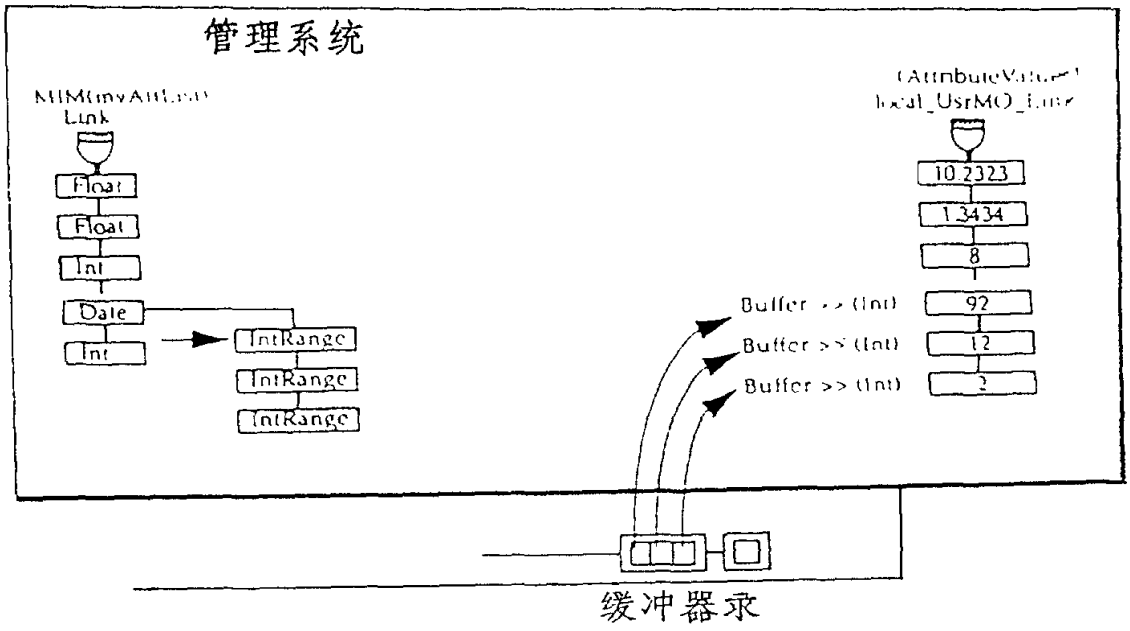

管理系统
Float
Float
Int
Date
Int
IntRange
IntRange
IntRange
10.2323
1.3434
8
92
12
2
缓冲器录

图 29

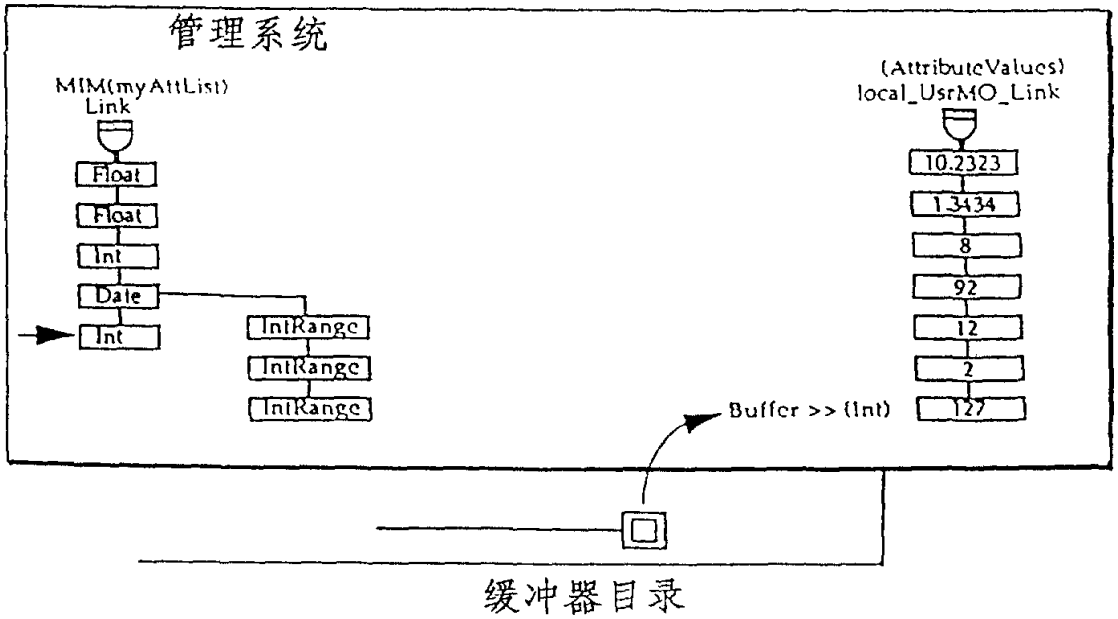

管理系统
MIM(myAttList)
Link
Float
Float
Int
Date
Int
IntRange
IntRange
IntRange
(AttributeValues)
local_UsrMO_Link
10.2323
1.3434
8
92
12
2
127
Buffer >> (Int)
缓冲器目录

图 30

```
1  OBJECT TYPE Subscriber IS
2
3  ATTRIBUTES
4    Number: NumberType;
5    AdmState: admStateType;
6    OpState: opStateType;
7    UsageState: usageStateType;
8    Line: REFERENCE LineDriver;
9  METHODS
10   LockRequest ();
11   Seise () RETURN Boolean;
12   Release ();
13
14 PERSISTENT PROPERTIES
15   PRIMARY KEY Number;
16   IMPLICIT UsageState;
17
18 MANAGED OBJECT PROPERTIES
19   READ-WRITE Line;
20   READ Number, AdmState, OpState,
     UsageState;
21   ACTIONS LockRequest;
22 END;
```

图 31

```
23    TYPE opStateType IS
24      ENUM disabled := 0, enabled := 1
25    END;
26
27    TYPE usageStateType IS
28      ENUM idle := 0, active := 1, busy := 2
29    END;
30
31    TYPE admStateType IS
32      ENUM locked := 0, unlocked := 1,
33        shuttingDown := 2
34    END;
35
36    TYPE NumberType IS
37        ARRAY OF BCDCodedDigits
38    END;
39
40    TYPE BCDCodedDigits IS
41        NATURAL RANGE 0..9
42    END;
```

图 32

```
43    OBJECT TYPE Subscriber IS
44
45    ATTRIBUTES
46      Number: NumberType;
47      AdmState: admStateType;
48      OpState: opStateType;
49      UsageState: usageStateType;
50      Line: REFERENCE LineDriver;
51    METHODS
52      LockRequest ();
53      Seise () RETURN Boolean;
54      Release ();
55
56    PERSISTENT PROPERTIES
57      PRIMARY KEY Number;
58      IMPLICIT UsageState;
59
60    MANAGED OBJECT PROPERTIES
61*     READ-WRITE Line;
62*     READ Number, AdmState, OpState,
63*       UsageState;
64*     ACTIONS LockRequest;
65      PROHIBIT CREATE, DELETE;
66    END;
```

图 33

```
67   OBJECT TYPE Subscriber IS
68
69   ATTRIBUTES
70     Number: NumberType;
71     AdmState: admStateType;
72     OpState: opStateType;
73     UsageState: usageStateType;
74     Line: REFERENCE LineDriver;
75   METHODS
76     LockRequest ();
77     Seise() RETURN Boolean;
78     Release();
79   PRE-CONDITIONS
80       SET Line TO NULL ONLY IF AdmState = locked;
81   PERSISTENT PROPERTIES
82     PRIMARY KEY Number;
83     IMPLICIT UsageState;
84
85   MANAGED OBJECT PROPERTIES
86     READ-WRITE Line;
87     READ Number, AdmState, OpState,
88       UsageState;
89     ACTIONS LockRequest;
90     PROHIBIT CREATE, DELETE;
91   END;
```

图 34

```
92   OBJECT TYPE Subscriber IS
93
94   ATTRIBUTES
95     Number: NumberType;
96     AdmState: admStateType;
97     OpState: opStateType;
98     UsageState: usageStateType;
99     Line: REFERENCE LineDriver INVERSE OF Subsc;
100  METHODS
101    LockRequest ();
102    Seise () RETURN Boolean;
103    Release ();
104  PRE-CONDITIONS
105    SET Line TO NULL ONLY IF AdmState=locked;
106  POST-CONDITIONS
107    NOT (Line = NULL AND AdmState = unlocked);
108  PERSISTENT PROPERTIES
109    PRIMARY KEY Number;
110    IMPLICIT UsageState;
111
112  MANAGED OBJECT PROPERTIES
113    READ-WRITE Line;
114    READ Number, AdmState, OpState,
115      UsageState;
116    ACTIONS LockRequest;
117    PROHIBIT CREATE,DELETE;
118  PARTY TO LineAndSubscriber;
119  END;
```

图 35

```
120
121    PRE-CONDITIONS
122      SET Line TO NULL ONLY IF AdmState = locked;
123    POST-CONDITIONS
124      NOT Line = NULL AND AdmState = unlocked;
125        RULE CHECK CONSISTENCY;
126    PERSISTENT PROPERTIES
127
128    END;
```

图 36

```
129
130   PRE-CONDITIONS
131     SET Line TO NULL ONLY IF AdmState = locked;
132   POST-CONDITIONS
133     NOT Line = NULL AND AdmState = unlocked;
134       RULE MAINTAIN CONSISTENCY;
135   PERSISTENT PROPERTIES
136
137   END;
```

图 37

```
138   OBJECT TYPE LineDriver IS
139
140   ATTRIBUTES
141     Cicuit: CircuitType;
142     AdmState: admStateType;
143     OpState: opStateType;
144     UsageState: usageStateType;
145     Subsc: REFERENCE Subscriber INVERSE OF Line;
146   METHODS
147     LockRequest();
148     Seise() RETURN Boolean;
149     Release();
150   PRE-CONDITIONS
151     SET Subsc TO NULL ONLY IF AdmState = locked;
152   POST-CONDITIONS
153     NOT (Subsc = NULL AND AdmState = unlocked);
154   PERSISTENT PROPERTIES
155     PRIMARY KEY Circuit;
156     IMPLICIT UsageState;
157
158   MANAGED OBJECT PROPERTIES
159     READ-WRITE Subsc;
160     READ Number, AdmState, OpState,
161       UsageState;
162     ACTIONS LockRequest;
163   PARTY TO LineAndSubscriber;
164   END;
```

图 38

```
165  DEPENDENCY SCHEMA LineAndSubscriber IS
166    FOR ALL LineDriver (1), Subscriber (s);
167    RELATIONS 1.Subsc = s;
168
169    POST-CONDITIONS
170      NOT (1.AdmState = unlocked AND
171       s.AdmState = locked);
172      RULE MAINTAIN CONSISTENCY;
173  END;
```

图 39

```
174
175  POST-CONDITIONS
176    WHEN LockRequest
177      AdmState = locked OR
178      AdmState = shuttingDown;
179
```

图 40

```
180
181  POST-CONDITIONS
182    WHEN CREATE
183      Line/= NULL;
184
```

图 41

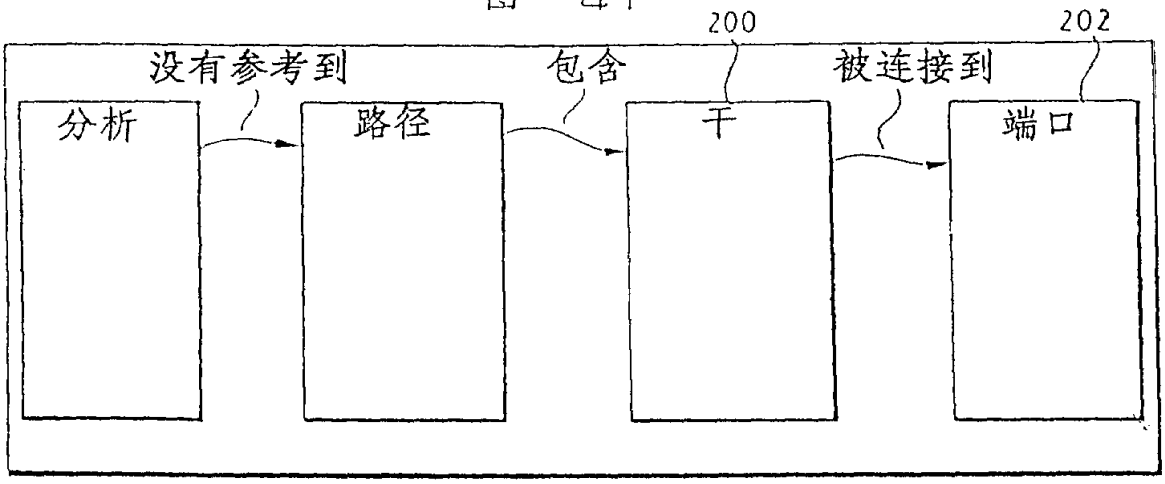

200
202
没有参考到
包含
被连接到
分析
路径
干
端口

图 42

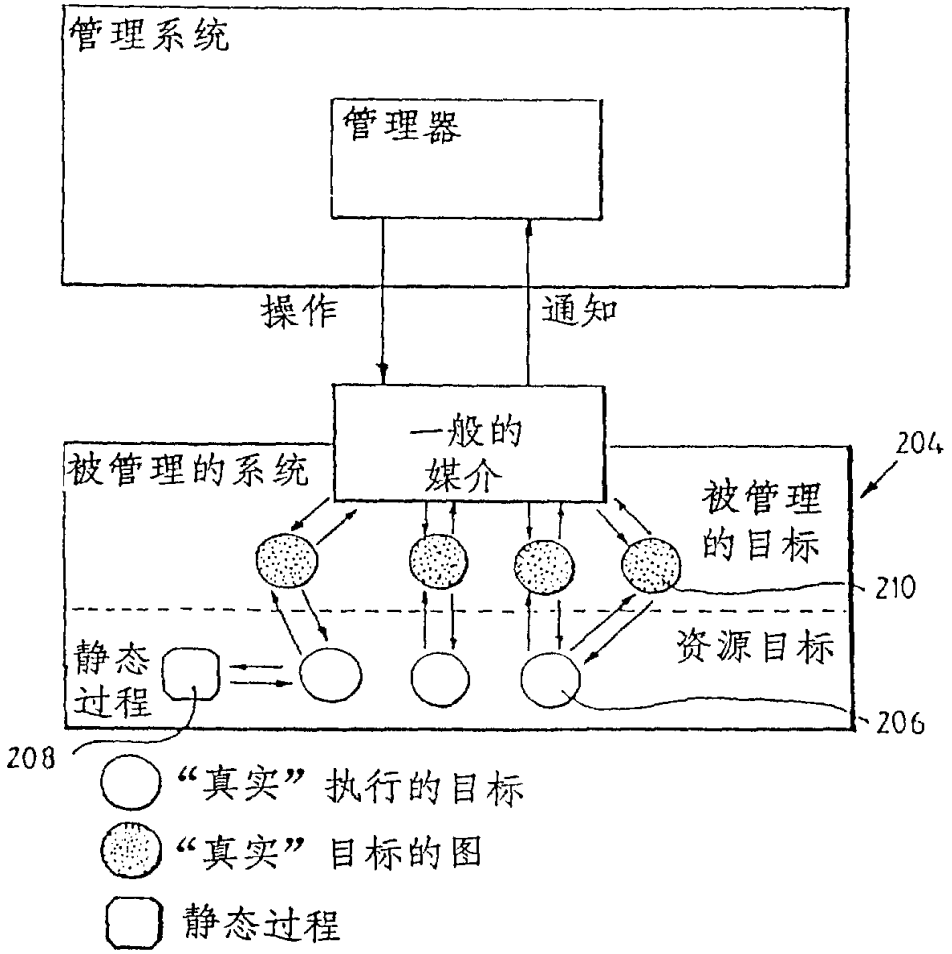

管理系统
管理器
操作
通知
一般的
媒介
被管理的系统
204
被管理
的目标
210
静态
过程
资源目标
206
208
“真实”执行的目标
“真实”目标的图
静态过程

图 43

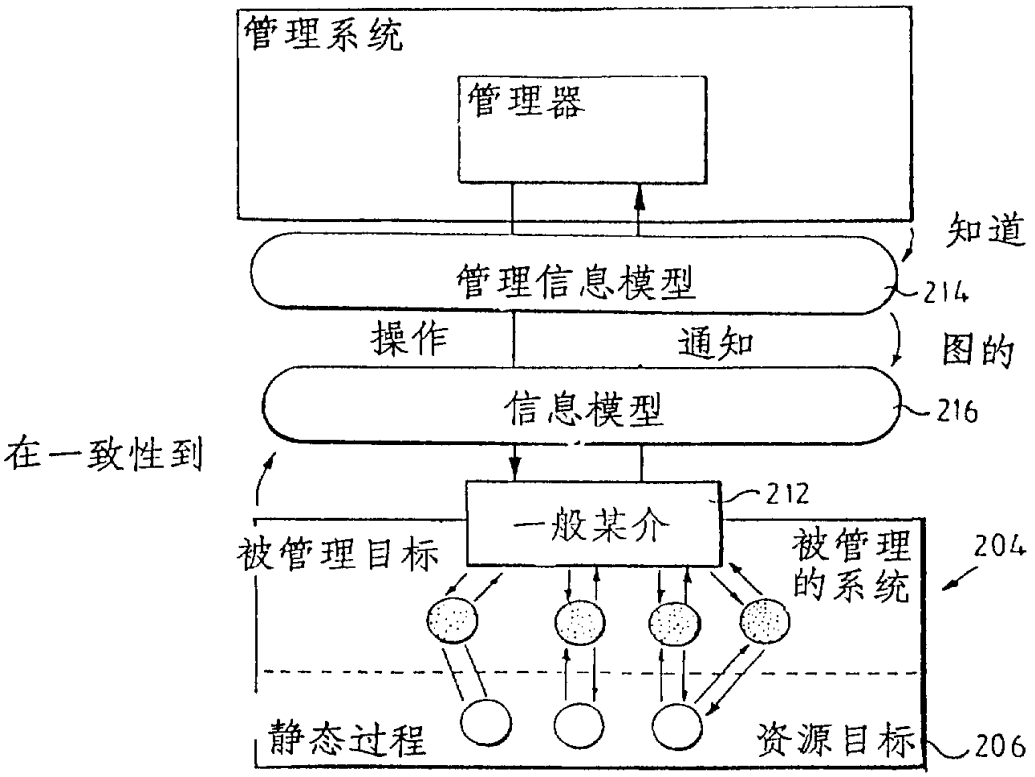

管理系统
管理器
知道
管理信息模型
214
操作
通知
图的
信息模型
216
在一致性到
212
一般某介
被管理目标
被管理的系统
204
静态过程
资源目标
206

图 44

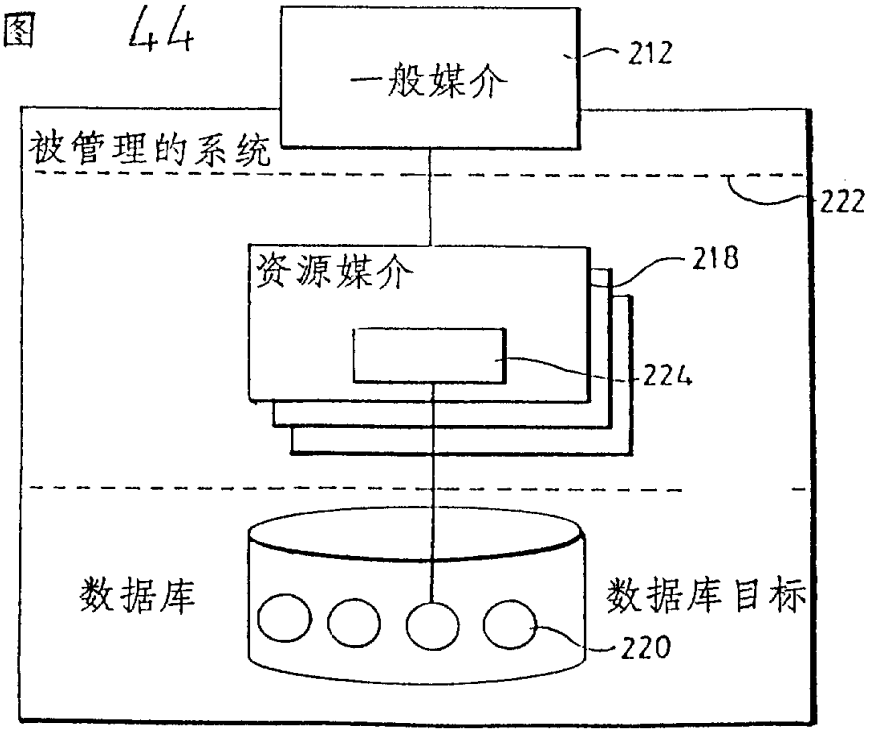

一般媒介
212
被管理的系统
222
资源媒介
218
224
数据库
数据库目标
220

图 45

```
1  OBJECT TYPE AnObject IS
2    BASE BaseObject;
3    ATTRIBUTES
4      attr1 : Atype;
5      attr2 : Integer;
6      attr3 : Integer;
7
8    METHODS
9      m1 (IN anArg : ArgType);
10     m2 (IN anArg : AnotherArgType)
       RETURNS AreturnType;
11
12   PRE-CONDITIONS
13     m1 (aValue) ONLY IF
       attr2 = attr3;
14
15   POST-CONDITIONS
16     attr2 >= attr3;
17
18 END;
```

图 46

```
20 OBJECT TYPE ResourceA IS
21     ATTRIBUTES
22       key : KeyType;
23       admState : AdministrativeState;
24       opState : OperationalState
25       Bref : REFERENCE TO ResourceB
                INVERSE OF Aref;
26     PRIMARY KEY key;
27
28     METHODS
29       allocate() RETURNS Boolean;
30
31     PRE-CONDITIONS
32       deleteObject () ONLY IF
         admState = locked;
33
34     POST-CONDITIONS
35       NOT (admState=unlocked
         AND Bref=NULL);
36
37     PARTY TO ResourceAandB;
38
39 END;
```

图 47

```
40 OBJECT TYPE ResourceB IS
41   ATTRIBUTES
42     key : KeyType;
43     admState : AdministrativeState;
44     opState : OperationalState:
45     Aref : REFERENCE TO ResourceA
              INVERSE OF Bref;
46   PRIMARY KEY key;
47
48   METHODS
49     allocate() RETURNS Boolean;
50
51   PRE-CONDITIONS
52     deleteObject() ONLY IF
       admState=locked;
53
54   PARTY TO ResourceAandB;
55
56 END;
```

图 48

```
57 DEPENDENCY SCHEMA ResourceAandB IS
58   FOR ALL ResourceA(a), ResourceB(b);
59   RELATIONS a.Bref = b;
60
61   POST-CONDITIONS
62     NOT (a.admState = unlocked AND
63          b.admState = locked);
64
65     NOT (a.opState = enabled AND
66            b.opState = disabled);
67
68 END;
```

图 50

```
69 DEPENDENCY SCHEMA ResourceAandB IS
70   FOR ALL ResourceA(a), ResourceB(b);
71   RELATIONS a.Bref = b;
72
73   POST-CONDITIONS
74     NOT (a.admState=unlocked AND
75          b.admState=locked)
76     RULES
77      WHEN COMMIT THEN CHECK CONSISTENCY
78
79     NOT (a.opState=enabled AND
89          b.opState=disabled)
81     RULES
82      WHEN b.opState=disabled THEN CONCLUDE
        a.opState=disabled,
83      WHEN COMMIT AND a.opState=enabled
        THEN CHECK CONSISTENCY
84
85 END;
```

图 49

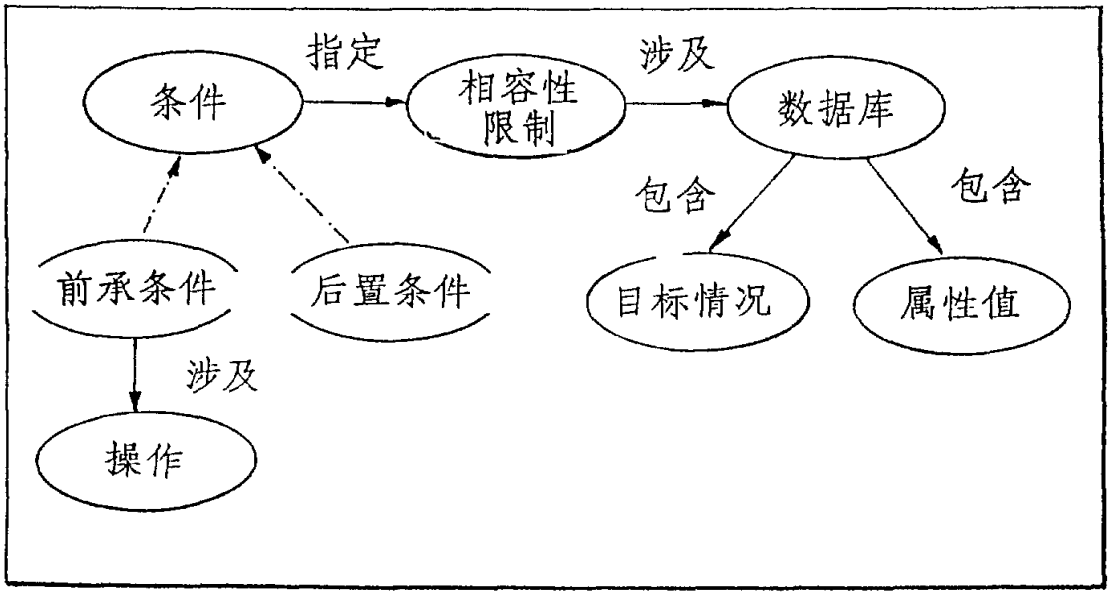


图 51

```
86 OBJECT TYPE ResourceA IS
87   ATTRIBUTES
88     key : KeyType;
89     admState : AdministrativeState;
90     internalOpState : OperationalState;
91     resourceBstate : OperationalState;
92     DERIVED opState=IF (internalOpState=
93     disabled OR resourceBstate=disabled)
94                        THEN disabled
95                        ELSE enabled
96     Bref : REFERENCE TO ResourceB
              INVERSE OF Aref;
97
98   PRIMARY KEY key;
99
100  METHODS
101    allocate() RETURNS Boolean;
102
103  PRE-CONDITIONS
104   deleteObject() ONLY IF admState=locked;
105
106  POST-CONDITIONS
107    NOT (admState=unlocked AND Bref=NULL);
108
109  PARTY TO ResourceAandB;
110END;
```

图 52

```
111DEPENDENCY SCHEMA ResourceAandB IS
112.....
113.....
114   NOT (a,opState = enabled AND
115         b.opState = disabled)
116   RULES
117    WHEN b.opState=NewValue
118    Then CONCLUDE a.resourceBstate=
       NewValue);
119
120END;
```

图 53

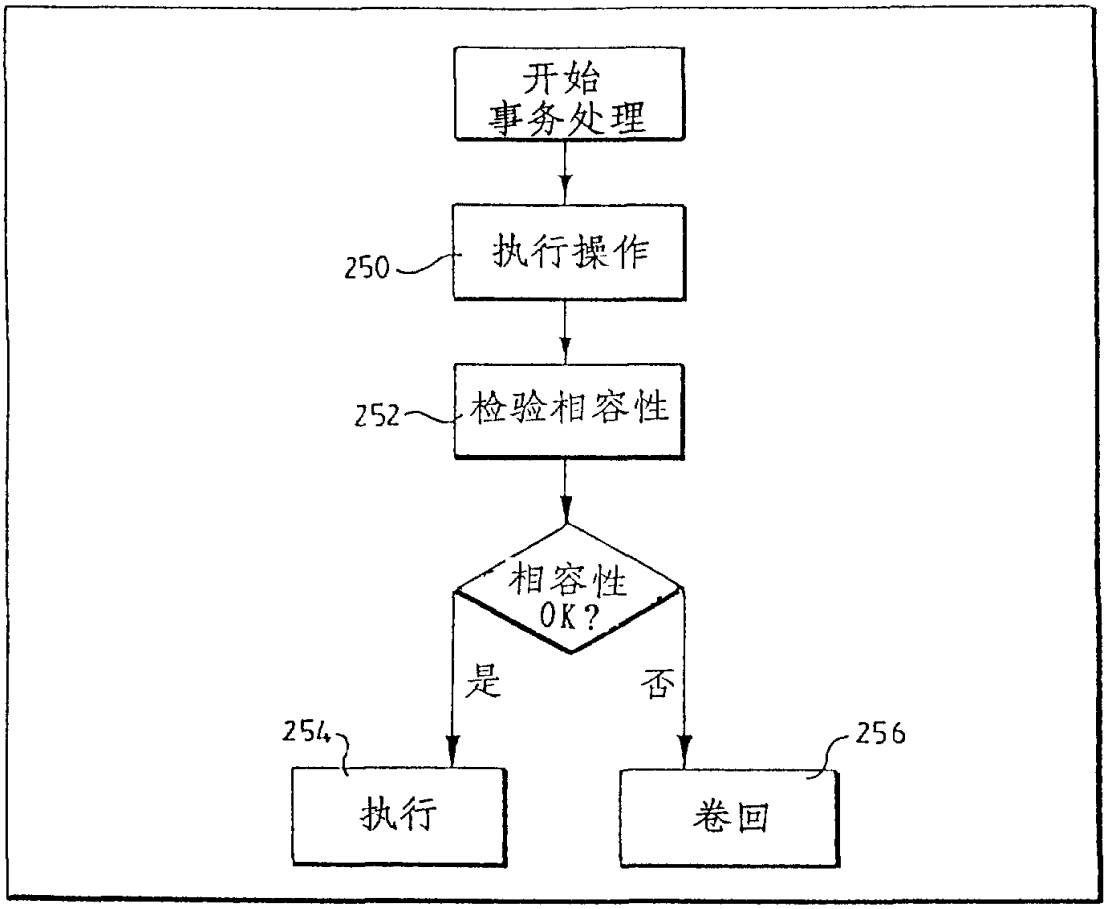


图 54

```
121 ifndef _ResourceA_hh_
122 define _ResourceA_hh_
123
124 include <PredefinedTypes.hh>
125 include "MOstateTypes.hh"
126
127class ResourceB;
128
129// OBJECT TYPE ResourceA
130
131class ResourceA
132(
133public
134   static ResourceA* open(Mode,const
135   KeyType&, DbTransaction*transaction=NULL);
136   void deleteObject ();
137   virtual Boolean checkConsistency
      (ErrorMessage&);
138   Boolean allocate ();
139
140   KeyType get Key ();
141   AdministrativeState getAdmState();
142   void setAdmState (const
      AdministrtiveState);
143   OperationalState getOpState();
144   void setOpState(const
      OperationalState);
145   ResourceB* getBref();
146   void setBref(ResourceB*);
147
148private
149   .....
150   .....
151);
152
153 endif
```

图 55

```
155 include "ResourceA.hh"
156
157//////////////////////////
158/////////
159//
160// ResourceA: METHOD checkConsistency
161//
162
163Boolean ResourceA: :checkConsistency
            (ErrorMessage& message)
164(
165   Boolean flag = true;
166
167   flag = !(getAdmState() == unlocked
      && getBreaf() == NULL;
168   if (!flag) (
169      createErrorMessage(1,message);
170   )
171   return flag;
172)
173
174
```

图 56

```
175 ifndef _ResourceB_hh_
176 define _ResourceB_hh_
177
178 include <PredefinedTypes.hh>
179 include "MOstateTypes.hh"
180
181class ResourceA;
182
183// BOJECT TYPE ResourceB
184
185class ResourceB
186(
187public
188   static ResourceB* open(Mode,const KeyType&,
189   DbTransaction*transaction=NULL);
190   void deleteObject();
191   Boolean allocate();
192   virtual Boolean checkConsistency
      (ErrorMessage&);
193
194   KeyType getKey ();
195   AdministrativeState getAdmState();
196   void setAdmState(const
      AdministrativeState);
197   OperationalState getOpState ();
198   void setOpState (const OperationalState);
199   ResourceA* getAref();
200   void setAref(ResourceA*);
201
202private
203   void propagateOpState
      (const OperationalState);
204   void opState(const OperationalState);
205.....
206.....
207);
208
209 endif
```

图 57

```
211 include "ResourceB.hh"
212 include "ResourceA.hh"
213
214//////////////////////
215//
216// ResourceB: METHOD setOpState
217//
218
219void ResourceB: setOpState(const
                    OperationalState newValue)
220(
221   OperationalState oldValue=getOpState();
222
223   opState(newValue);
224   if (newValue==disabled && newValue
          != oldValue)
225   (
226     PropagateOpState (newValue);
227   )
228)
229
230//////////////////////
231//
232// ResourceB: METHOD propagateOpState
233//
234
235void ResourceB: .PropagateOpState
      (const OperationalState
236(
237   getAref () ->setOpState (newValue);
238)
239
```

图 58

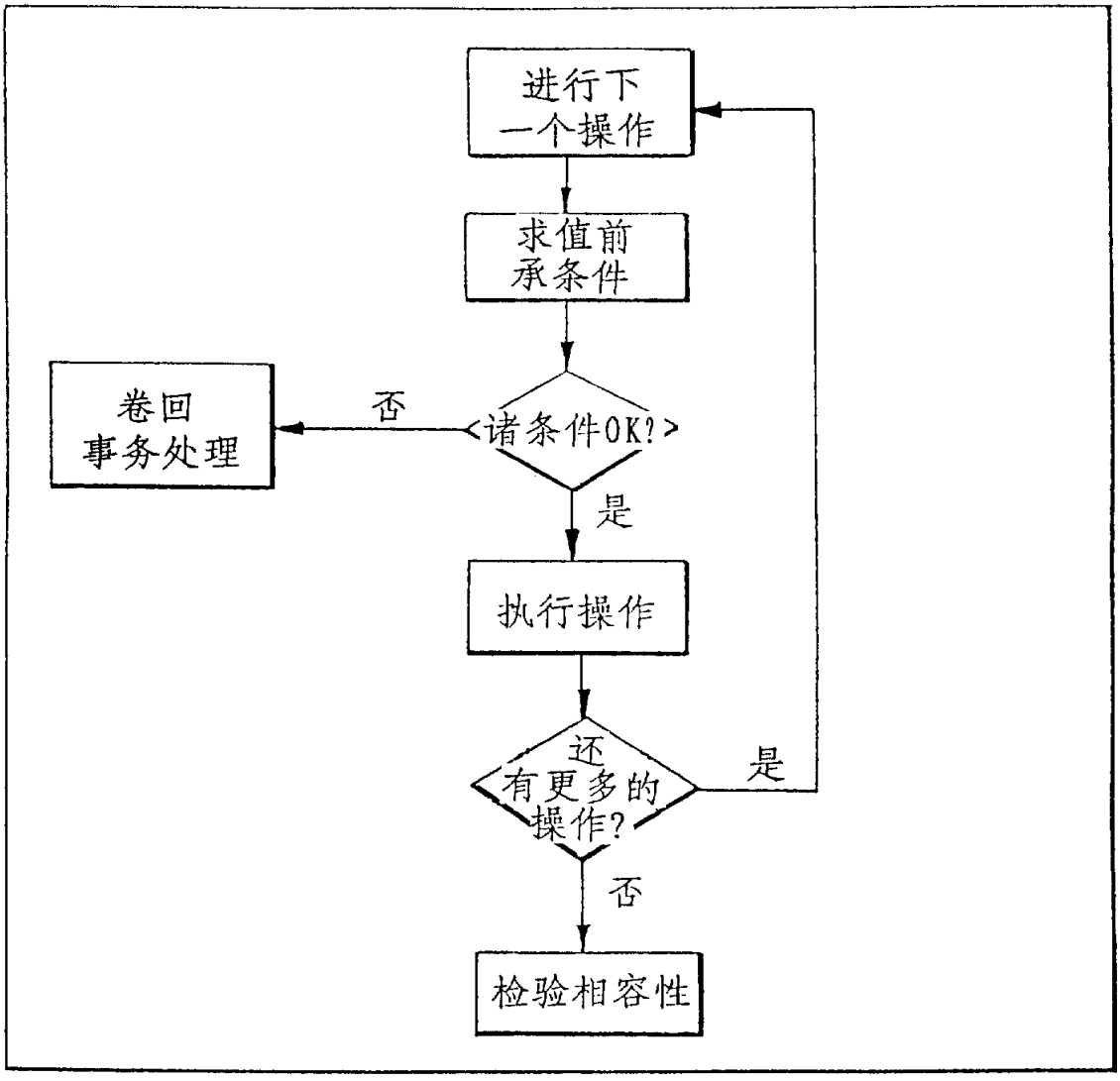


图 59

```
240 ifndef _ResourceA_hh_
241 define _ResourceA_hh_
242
243 include <PredefinedTypes.hh>
244 include "MOstateTypes.hh"
245
246class ResourceB;
247
248// OBJECT TYPE ResourceA
249
250class ResourceA
251(
252public
253   static ResourceA* open(Mode,const
254   KeyType&, DbTransaction* transaction=NULL);
      =NULL);
255   void deleteObject ();
256   virtual Boolean checkConsistency
      (ErrorMessage&);
257   Boolean allocate ();
258
259   KeyType getKey();
260   AdministrativeState getAdmState();
261   void setAdmState(const
      AdministrativeState);
262   OperationalState getOpState)(;
263   void setOpState(const
      OperationalState),
264   ResourceB* getBref();
265   void setBref(ResourceB*);
266
267private
268   void deleteObjectCondition();
269   void reallyDeleteObject();
270);
271
272 endif
```

图 60

```
274 include "ResourceA.hh"
275
276//////////////////////////
277//
278// ResourceA. METHOD remove
279//
280
281void ResourceA: :deleteObject()
282
283  deleteObjectcondition();
284  reakkyDeleteObject();
285)
286
287//////////////////////////
288//
289// ResourceA: METHOD
       deleteObjectCondition
290//
291
292void ResourceA: :deleteObjectCondition()
293(
294  AdministrativeState admStateValue=
     admStateDatabaseValue();
295
296  if (admStateValue != locked)
297  (
298    throw ErrorMessage(2,admStateValue);
299  )
300)
```

图 61

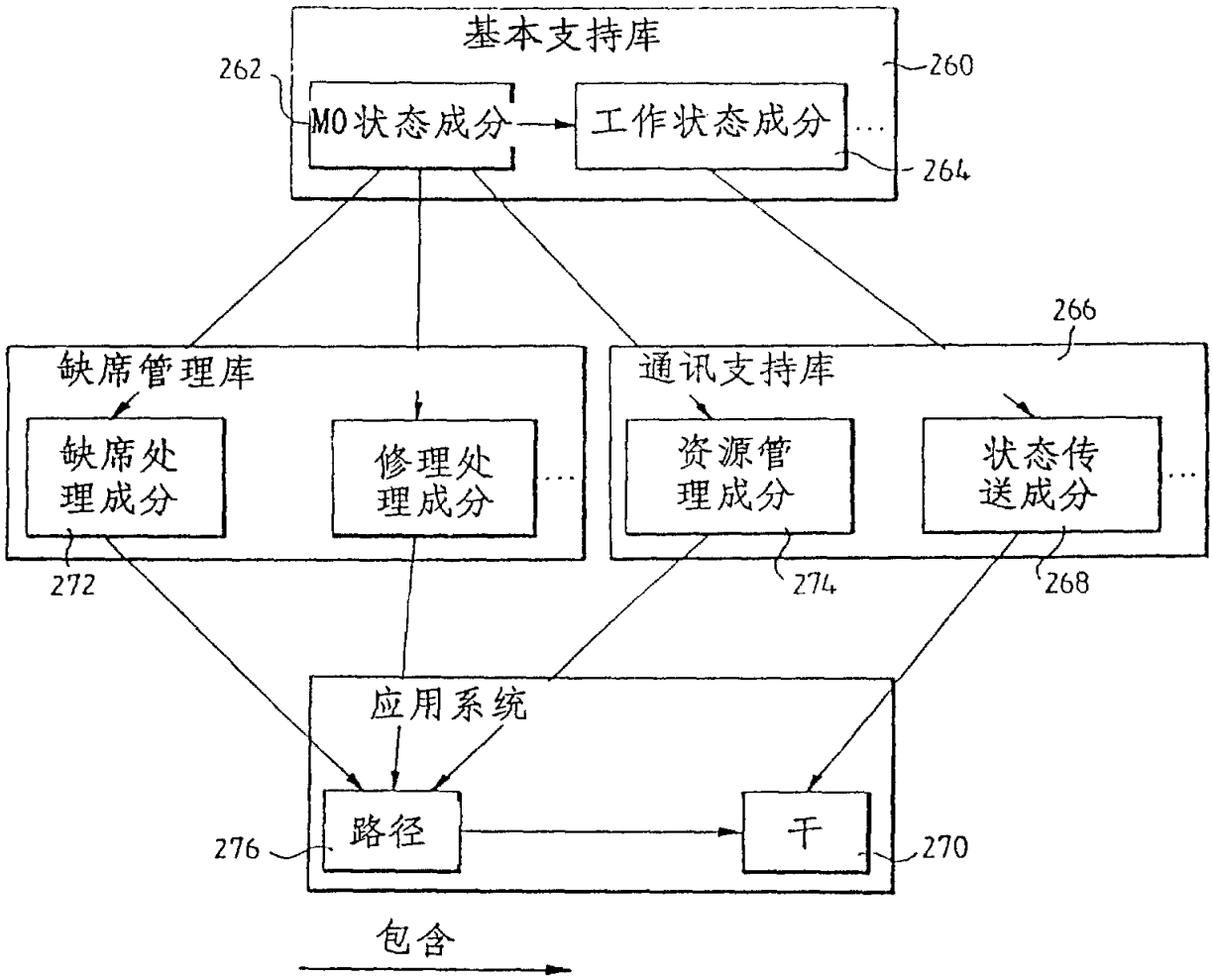

基本支持库
262
MO状态成分
工作状态成分
260
264
缺席管理库
缺席处理成分
修理处理成分
272
通讯支持库
资源管理成分
状态传送成分
266
274
268
应用系统
路径
干
276
270
包含

图 62

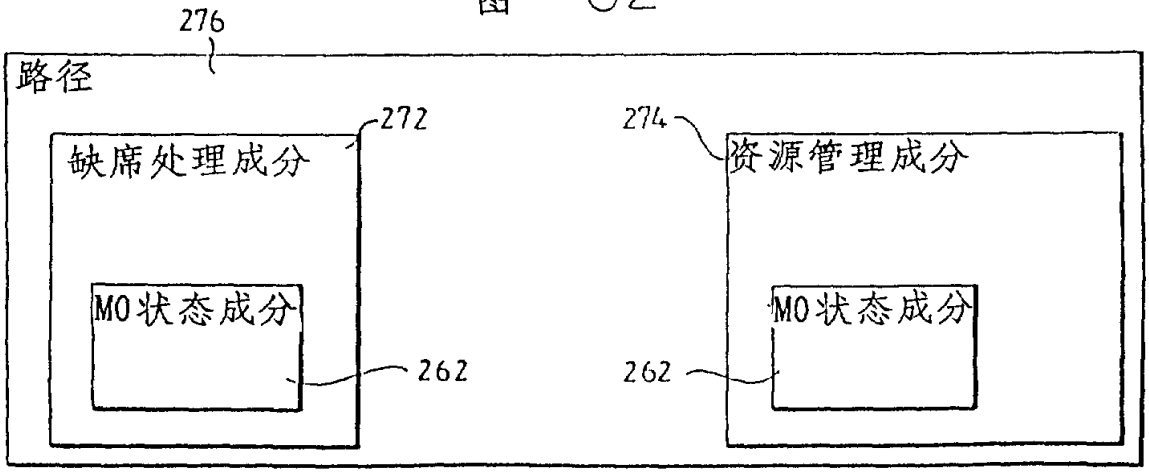

276
路径
272
缺席处理成分
MO状态成分
262
274
资源管理成分
MO状态成分
262

图 63

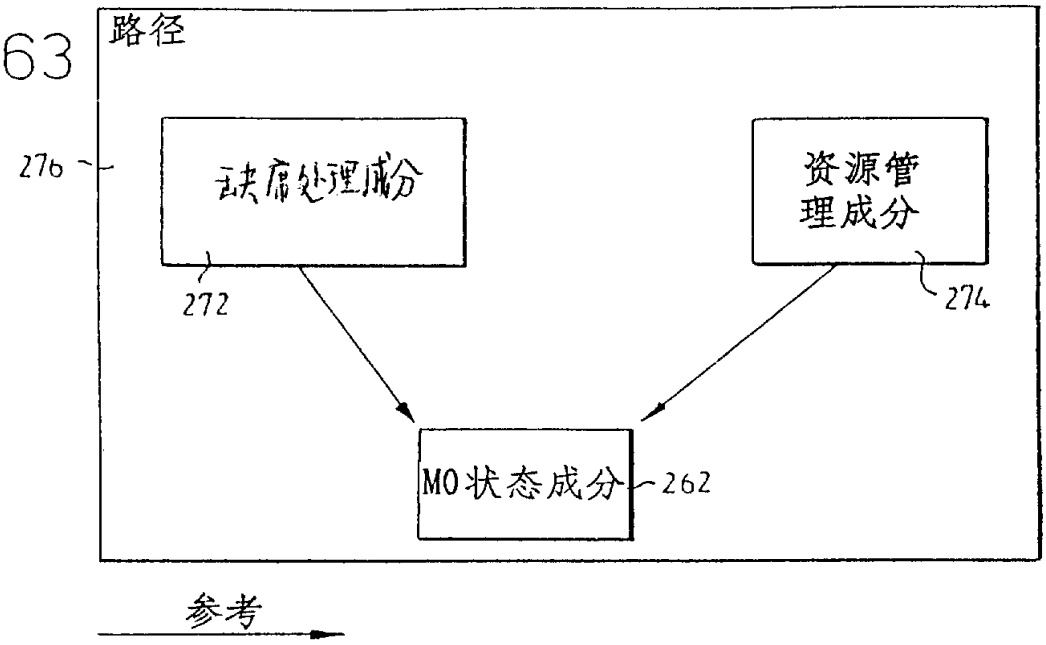

路径
276
缺席处理成分
272
资源管理成分
274
MO状态成分
262
参考

图 64

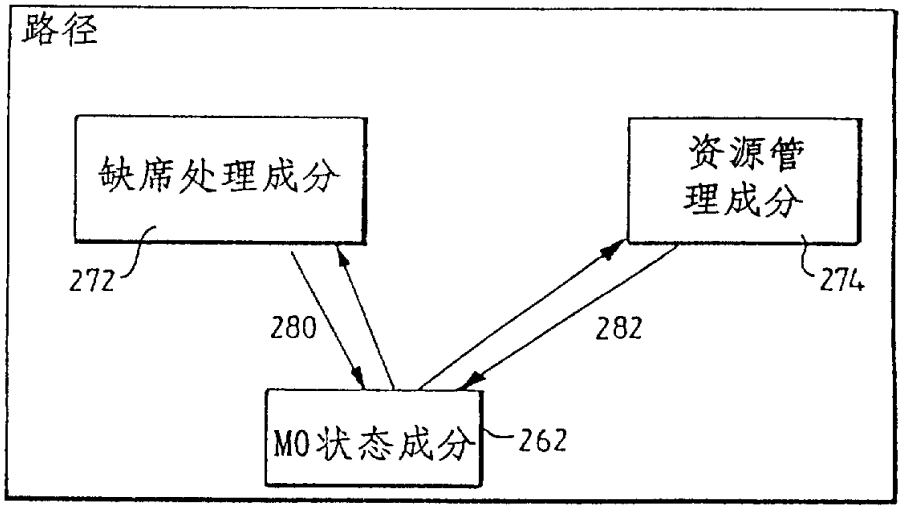

路径
缺席处理成分
272
资源管理成分
274
280
282
MO状态成分
262

图 65

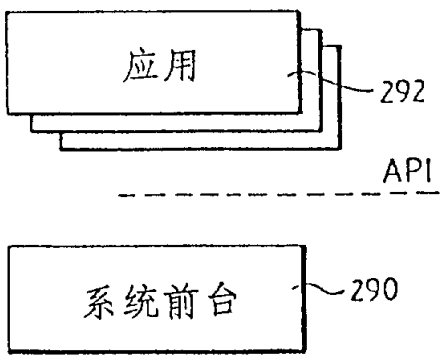

应用
292
API
系统前台
290

图 66

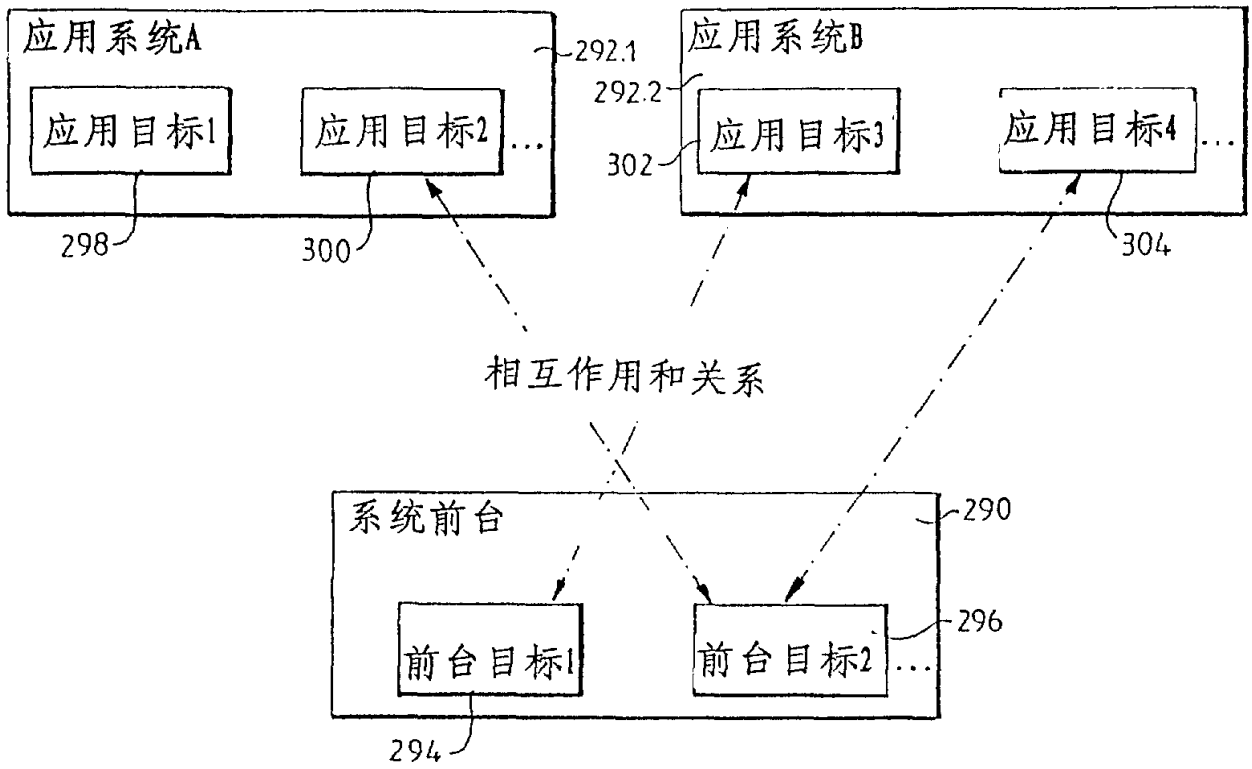

应用系统A
292.1
应用系统B
292.2
应用目标1
应用目标2
...
应用目标3
应用目标4
...
302
298
300
304
相互作用和关系
系统前台
290
前台目标1
前台目标2
...
296
294

图 67

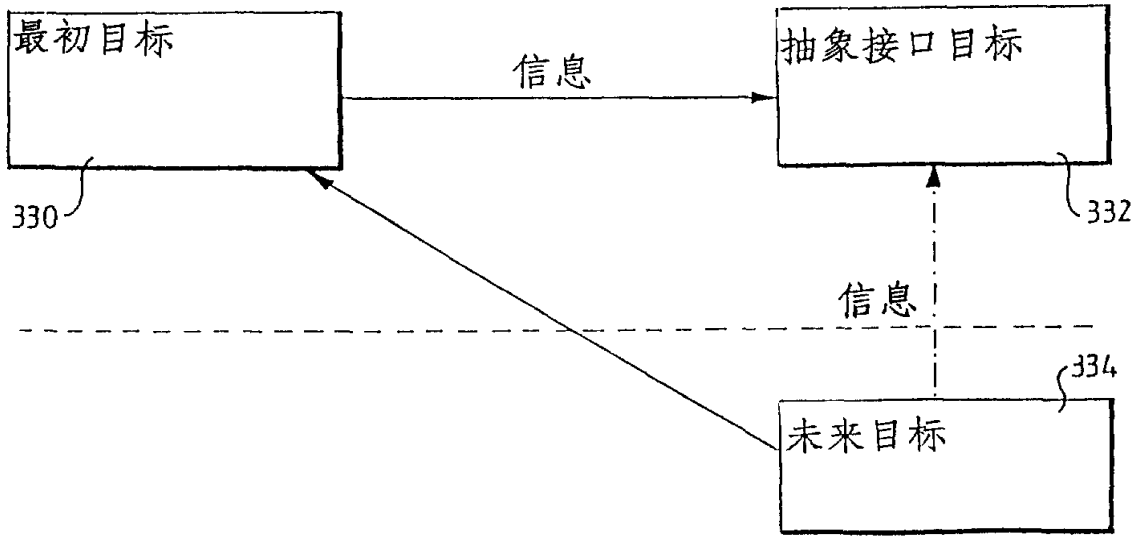

最初目标
信息
抽象接口目标
330
332
信息
334
未来目标

图 68

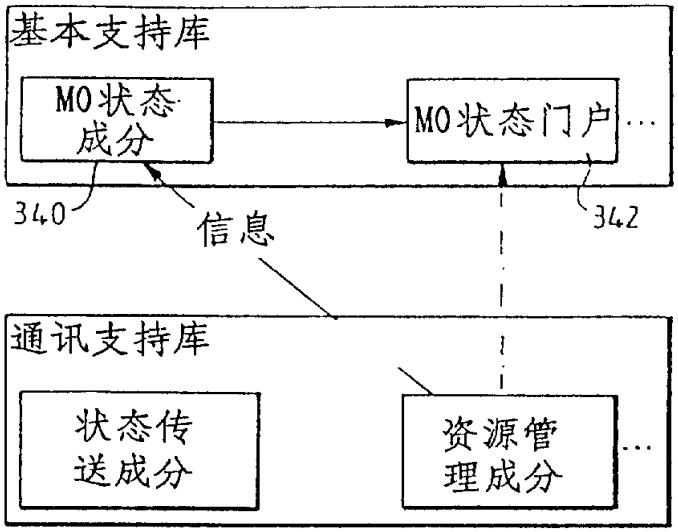

基本支持库
MO状态成分
MO状态门户
340
信息
342
通讯支持库
状态传送成分
资源管理成分

图 69

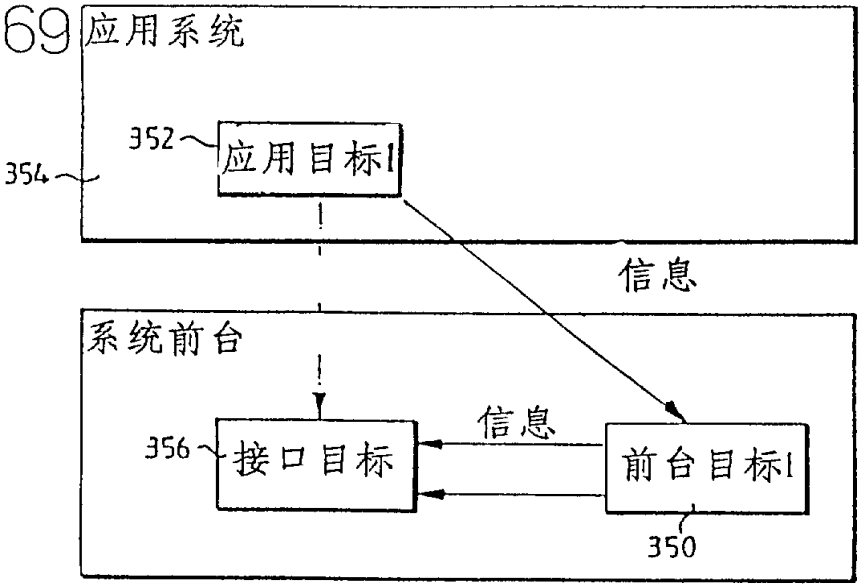

应用系统
352
354
应用目标1
信息
系统前台
信息
356
接口目标
前台目标1
350

图 70

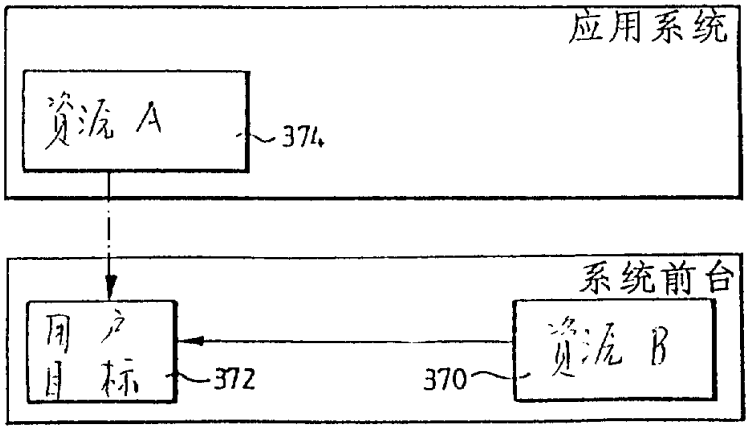

应用系统
资流 A
374
系统前台
用户目标
372
370
资流 B

图 71

```
1  OBJECT TYPE ResourceB IS
2    ATTRIBUTES
3      key : KeyType;
4      admState : AdministrativeState;
5      opState : OperationalState;
6      userRef : REFERENCE TO UserObject
                 INVERSE OF Bref;
7    PRIMARY KEY key;
8
9    METHODS
10     allocate() RETURNS Boolean;
11
12   PRE-CONDITIONS
13     deleteObject() ONLY IF
       admState = locked;
14
15   PARTY TO ResourceAndUser;
16END;
```

图 72

```
17OBJECT TYPE UserObject IS
18   ATTRIBUTES
19      admState : AdministrativeState,
                   PURE VIRTUAL;
20      resourceBstate : OperationalState;
21      Bref : REFERENCE TO ResourceB
               INVERSE OF UserRef;
22
23   PARTY TO ResourceAndUser;
24END;
```

图 73

```
25DEPENDENCY SCHEMA ResourceAndUser IS
26   FOR ALL UserObject(a), ResourceB(b);
27   RELATIONS a.Bref = b;
28
29   POST-CONDITIONS
30     NOT (a.admState = unlocked AND
31             b.admState = locked)
32     RULES
33      WHEN COMMIT THEN CHECK CONSISTENCY;
34
35   PROPAGATIONS
36     WHEN b.opState = NewValue
37     THEN CONCLUDE a.resourceBstate=NewValue;
38
39END;
```

图 74

```
40OBJECT TYPE ResourceA IS
41   BASE UserObject;
42
43   ATTRIBUTES
44     key : KeyType;
45     admState : AdministrativeState;
46     DERIVED opState = IF (InternalOpState=disabled OR
47                           resourceBstate=disabled)
48                       THEN disabled
49                       ELSE enabled;
50
51     internalOpState : OperationalState PRIVATE;
52
53   PRIMARY KEY key;
54
55   METHODS
56     allocate() RETURNS Boolean;
57
58   PRE-CONDITIONS
59     deleteOblect() ONLY IF admState = locked;
60
61   POST-CONDITIONS
62     NOT (admState = unlocked AND Bref = NULL);
63
64END;
```

图 75

```
65 ifndef _UserObject_hh_
66 define _UserObject_hh_
67
68 include <PredefinedTypes.hh>
69 include "MOstateTypes.hh"
70
71class ResourceB
72
73// OBJECT Type Userobject
74
75class Userobject
76(
77public
78   static UserObject* open(Mode,const
79   KeyType&, DbTransaction* transaction=
     NULL);
80   virtual deleteObject() =0;
81   virtual Boolean checkConsistency
     (ErrorMessage&);
82
83   virtual AdministrativeState
     getAdmState() =0;
84   virtual void setAdmState
     (const AdministrativeState)==;
85   OperationalState getResourceBstate();
86   setResourceBstate(const
     OperationalState);
87   ResourceB* getBref();
88   void setBref(ResourceB*);
89
90private:
91.....
92.....
93);
94
95 endif
```

图 76

```
97 include "UserObject.hh"
98 include "ResourceB.hh"
99
100//////////////////////
101//
102// UserObject: METHOD checkConsistency
103//
104
105Boolean UserObject::checkConsistency(
            ErrorMessage& message)
106(
107   Boolean flag = true;
108
109   flag = !(getAdmState()  unlocked &&
110              getBref()->get admState()==locked);
111   if ( !flag) (
112      createErrorMessage(1,message);
113   )
114   return flag;
115)
116
```

图 77

```
117 ifndef __ResourceB_hh_
118 define _ResourceB_hh_
119
120 include <PredefinedTypes.hh>
121 include "MOstateTypes.hh"
122
123class UserObject;
124
125// OBJECT TYPE ResourceB
126
127class ResourceB
128(
129public
130   void deletobject();
131
132   void deleteObject();
133   Boolean allocate();
134   virtual Boolean checkConsistency
      (ErrorMessage&);
135
136   KeyType getKey();
137   AdministrativeState getAdmState();
138   void setAdmState (const
      AdministrativeState);
139   OperationalState getOpState();
140   void setOpState(const OperationalState);
141   UserObject* getUserRef();
142   void setUserRef(UserObject*);
143
144private
145   void propagateOpState
      (const OperationalState);
146   void opState (const OperationalState);
147   .....
148   .....
149);
150
151 endif
```

图 78

```
153 include "ResourceB.hh"
154 include "UserObject.hh"
155
156//////////////////////
157//
158// ResourceB: METHOD setOpState
159//
160
161void ResourceB: :setOpState(
            const OperationalState newValue)
162(
163  OperationalState oldValue = getOpState();
164
165  opState(newValue);
166  if (newValue != oldValue)
167  (
168     propagateOpState(newValue);
169  )
170)
171
172//////////////////////////
173//
174// ResourceB: METHOD PropagateOpState
175//
176
177void ResourceB: :propagateOpState(
              const OperationalState newValue)
178(
179  getUserRef()->resourceBstate(newValue)
180)
181
```

图 79

```
182 ifndef _ResourceA_hh_
183 define _ResourceA_hh_
184
185 include <PredefinedTypes.hh>
186 include "UserObject.hh"
187
188// OBJECT TYPE ResourceA
189
190class ResourceA : public UserObject
191(
192public
193   static ResourceA* open(Mode,
194   const KeyType&, DbTransaction*
      transaction=NULL);
195   void deleteObject ();
196   virtual Boolean checkConsistency
      (ErrorMessage&);
197
198   Boolean allocate();
199
200   KeyType getKey();
201   AdministrativeState getAdmState();
202   void setAdmState (const
      AdministrativeState);
203   OperationalState getOpState();
204
205private
206   OperationalState getInternalOpState();
207   void setInternalOpState (const
      OperationalState);
208   .....
209   .....
210);
211
212 endif
213
```

图 80

```
214 include "ResourceA.hh"
215
216//////////////////////
217//
218// ResourceA: METHOD getOpState
219//
220
2210perationalState ResourceA: :getOpState()
222(
223   if  (get InternalOpState() ==disabled
224         getResourceBstate() ==disabled)
225   (
226    return disabled;
227   )
228   else
229   (
230    return enabled;
231   )
232)
233
```