[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 15/16 (2006.01)

[12] 发明专利说明书

专利号 ZL 200410033027.7

[45] 授权公告日 2009 年 11 月 11 日

[11] 授权公告号 CN 100559366C

[22] 申请日 2004.3.5

[21] 申请号 200410033027.7

[30] 优先权

[32] 2003. 3. 6 [33] US [31] 60/452，736

[32] 2003.10.24 [33] US [31] 10/693，004

[73] 专利权人 微软公司

地址 美国华盛顿州

[72] 发明人 G·C·亨特 G·奥斯莱德

B·塔巴拉 K·格里利希

R·蒙辛

[56] 参考文献

CN1392502A 2003.1.22

JP5－233655A 1993.9.10

US6349307B1 2002.2.19

CN1368694A 2002.9.11

CN1375685A 2002.10.23

CN1295292A 2001.5.16

CN1306644A 2001.8.1

审查员 孙泽竑

[74] 专利代理机构 上海专利商标事务所有限公司

代理人 张政权

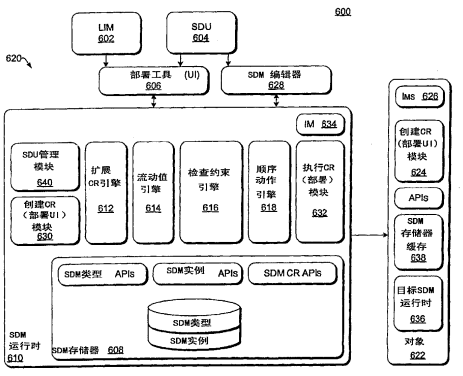
权利要求书 4 页 说明书 101 页 附图 22 页

[54] 发明名称

用于设计、开发和管理一个或多个分布式计算系统的方法和系统

[57] 摘要

描述了一种在分布式计算系统上设计、部署和管理分布式应用程序的体系结构和方法。



1. 一种用于设计、开发和管理一个或多个分布式计算系统的方法，所述方法包括：

创建一个或多个系统定义文档，所述系统定义文档定义了所述一个或多个分布式计算系统的环境需求，定义了有关部署所述一个或多个分布式计算系统的系统部署知识，还定义了管理所述一个或多个分布式计算系统的系统管理知识；

存储所述一个或多个系统定义文档；

通过将存储在所述系统定义文档中的环境需求和所述一个或多个分布式计算系统的环境进行比较来验证所述一个或多个分布式计算系统；

通过使用所述系统定义文档中的系统部署知识来将所述一个或多个分布式计算系统部署在一个或多个环境中；以及

通过使用所述系统定义文档中的系统管理知识来对所述一个或多个分布式计算系统进行修改。

2. 如权利要求1所述的方法，其特征在于，所述创建一个或多个系统定义文档的步骤进一步包括使用开发工具来定义所述分布式计算系统的软件和硬件组件。

3. 如权利要求1所述的方法，其特征在于，所述使用所述系统定义文档中的系统部署知识进一步包括动态分配和配置所述软件和硬件资源。

4. 如权利要求1所述的方法，其特征在于，定义所述一个或多个分布式计算系统的环境需求的步骤包括定义有关其上要部署所述分布式计算系统的计算设备的硬件需求、有关其上要部署所述分布式计算系统的计算设备的软件需求以及有关其上要部署所述分布式计算系统的计算设备的其他需求。

5. 如权利要求1所述的方法，其特征在于，将系统定义文档中存储的环境需求和所述一个或多个分布式计算系统的环境进行比较的步骤进一步包括如果所述环境需求不能满足所述分布式计算系统的需求就对所述一个或多个系统定义文档进行修改。

6. 如权利要求1所述的方法，其特征在于，用控制器从所述系统定义文档中

提取有关部署所述一个或多个分布式计算系统的所述系统部署知识。

7. 如权利要求1所述的方法，其特征在于，在一个或多个环境中部署一个或多个分布式计算系统的步骤进一步包括生成一个部署期间所涉及软件资源的记录。

8. 如权利要求1所述的方法，其特征在于，有关管理所述一个或多个分布式计算系统的系统管理知识包括下述项中的至少一个项：配置、修补、升级、维护任务、健康或性能检测。

9. 如权利要求1所述的方法，其特征在于，对所述一个或多个分布式计算系统进行修改的步骤进一步包括维护对系统所做的任何修改的记录。

10. 如权利要求9所述的方法，其特征在于，用控制器来维护对系统所做的任何修改的记录。

11. 如权利要求1所述的方法，其特征在于，其中，所述创建一个或多个系统定义文档的步骤进一步包括：

创建描述所述系统中存在的实体的一个或多个定义；

创建多个关系来识别系统中所述实体之间的链接；

通过定义至少一组允许的关系来创建系统的约束；

使用所述关系中的约束来限制可以被链接的定义；以及

在分布式计算系统中加载所述一个或多个定义。

12. 如权利要求11所述的方法，其特征在于，创建一个或多个定义的步骤包括创建至少一个抽象定义，该抽象定义与分布式应用程序的模板相关，且包括与分布式应用程序的特定执行相关联的至少一个具体定义。

13. 如权利要求11所述的方法，其特征在于，创建一个或多个定义的步骤包括创建一个描述在所述分布式计算系统中使用的应用程序的一部分的系统定义。

14. 如权利要求11所述的方法，其特征在于，创建一个或多个定义的步骤包括：

创建描述应用程序的自包含独立可部署部分的至少一个系统定义；

创建至少一个终端定义，用于与所述分布式计算系统关联的所有形式的通信建模；以及

创建描述与所述分布式计算系统相关的行为的至少一个资源定义。

15. 如权利要求11所述的方法，其特征在于，创建多个关系以识别实体之间的链接的步骤包括：

创建识别所述定义之间可用通信交互的至少一个通信关系；

创建描述一个特定定义包含其它所述定义的成员的能力的至少一个包含关系；

创建示出在特定定义内包含的成员的至少一个代理关系；

创建描述在所述定义之间的依赖性的至少一个寄宿关系；

创建确认在参数流和结构排序中使用的所述定义之间的附加依赖性的至少一个参考关系。

16. 如权利要求11所述的方法，其特征在于，所述一个或多个定义和所述关系由一个或多个开发、部署和管理工具使用。

17. 如权利要求 15 所述的方法，其特征在于，进一步包括在所述分布式计算系统中加载至少一个通信关系。

18. 一种用于设计、开发和管理一个或多个分布式计算系统的系统，所述系统含有一个或多个系统定义文档，所述系统定义文档定义了所述一个或多个分布式计算系统的环境需求，定义了有关部署所述一个或多个分布式计算系统的系统部署知识，还定义了管理所述一个或多个分布式计算系统的系统管理知识，所述系统包括：

用于存储所述一个或多个系统定义文档的装置；

用于通过将存储在所述系统定义文档中的环境需求和所述一个或多个分布式计算系统的环境进行比较来验证所述一个或多个分布式计算系统的装置；

用于通过使用所述系统定义文档中的系统部署知识来将所述一个或多个分布式计算系统部署在一个或多个环境中的装置；以及

用于通过使用所述系统定义文档中的系统管理知识来对所述一个或多个分布式计算系统进行修改的装置。

19. 如权利要求18所述的系统，其特征在于，所述一个或多个系统定义文档进一步包括所述分布式计算系统的软件和硬件组件的定义。

20. 如权利要求18所述的方法，其特征在于，所述使用所述系统定义文档中的系统部署知识能进一步动态分配和配置所述软件和硬件资源。

21. 如权利要求18所述的系统，其特征在于，定义环境需求包括进一步定

义其上要部署所述系统的计算设备的至少一个硬件需求、其上要部署所述系统的计算设备的至少一个软件需求以及有关其上要部署所述系统的计算设备的至少一个其他需求。

22. 如权利要求18所述的系统，其特征在于，用于通过将存储在所述系统定义文档中的环境需求和所述一个或多个分布式计算系统的环境进行比较来验证所述一个或多个分布式计算系统的装置进一步用于如果所述环境需求不能满足所述分布式计算系统的需求就对所述一个或多个系统定义文档进行修改。

23. 如权利要求18所述的系统，其特征在于，由控制器提取所述系统部署知识。

24. 如权利要求23所述的系统，其特征在于，由控制器进一步生成一个部署期间所涉及软件资源的记录。

25. 如权利要求24所述的系统，其特征在于，所述系统管理知识包括对所述一个或多个分布式计算系统进行配置、修补、升级、执行维护任务或进行健康和性能检测中所用的知识。

26. 如权利要求24所述的系统，其特征在于，由控制器对系统所做的任何修改的记录进行维护。

27. 如权利要求24所述的系统，其特征在于，控制器维护对系统所做的任何修改的记录。

用于设计、开发和管理一个或多个分布式计算系统的方法和系统

相关申请

本申请要求了美国临时申请No.60/452,736的优先权，该申请提交于2003年3月6日，名为“分布式计算系统的体系结构以及分布式应用程序的自动设计、部署和管理”，在此引入作为参考。

本专利申请与如下美国专利申请相关（将这些申请的全部内容引入于此，以作为参考）：

◆ 美国专利申请序列号No.10/382,942，提交于2003年3月6日，名为“VIRTUAL NETWORK TOPOLOGY GENERATION (虚拟网络拓扑产生)”。

◆ 美国专利申请序列号No.09/695,812，提交于2000年10月24日，名为“SYSTEM AND METHOD FOR DISTRIBUTED MANAGEMENT OF SHARED COMPUTERS (共享式计算机的分布式管理系统和方法)”。

◆ 美国专利申请序列号No.09/695,813，提交于2000年10月24日，名为“SYSTEM AND METHOD FOR LOGICAL MODELING OF DISTRIBUTED COMPUTER SYSTEMS (分布式计算机系统的逻辑建模系统和方法)”。

◆ 美国专利申请序列号No.09/695,820，其申请于2000年10月24日，名为“SYSTEM AND METHOD FOR RESTRICTING DATA TRANSFERS AND MANAGING SOFTWARD COMPONENTS OF DISTRIBUTED COMPUTERS (分布式计算机中限制数据传输和管理软件组件的系统和方法)”。

◆ 美国专利申请序列号No.09/695,821，提交于2000年10月24日，名为“USING PACKET FILTERS AND NETWORK VIRTUALIZATION TO RESTRICT NETWORK COMMUNICATIONS (利用包滤波器和网络虚拟化来

限制网络通信)”。

◆ 美国专利申请序列号No.09/696,707，提交于2000年10月24日，名为“SYSTEM AND METHOD FOR DESIGNING A LOGICAL MODEL OF DISTRIBUTED COMPUTER SYSTEM AND DEPLOYING PHYSICAL RESOURCES ACCORDING TO THE LOGICAL MODEL (分布式计算机中设计逻辑模型并根据该逻辑模型部署物理资源的系统和方法)”。

◆ 美国专利申请序列号No.09/696,752，提交于2000年10月24日，名为“SYSTEM AND METHOD PROVIDING AUTOMATIC POLICY ENFORCEMENT IN A MULTICOMPUTER SERVICE APPLICATION (在多计算机服务应用程序中提供自动策略实施的系统和方法)”。

技术领域

本发明涉及分布式计算系统的体系结构。

背景技术

在过去的几年中，对因特网的应用迅速扩大并在不断地增长。人们变得非常适应万维网（或简称“Web”）上提供的多种服务，例如电子邮件、在线购物、搜集新闻和信息、收听音乐、观看视频剪辑、寻找工作等等。为了与基于因特网服务的日益增长的需求齐头并进，专用于宿主站点的计算机系统出现了巨大的增长，以提供这些站点的后端服务，并存储与这些站点相关联的数据。

分布式计算机系统的一种类型是数据中心（例如因特网数据中心（IDC）或企业数据中心（EDC）），其是一个特别设计的集合体，容纳了许多用于寄宿基于网络的服务的计算机。数据中心，也被称为“Webfarms”或“server farms”，一般在空调控制（climate-controlled）且安全的大楼（physically secure buildings）中容纳有成千上万的计算机。数据中心一般提供可靠的因特网访问、可靠的电力供应和安全的操作环境。

今天，大型的数据中心非常复杂并经常用于寄宿多种应用程序。例如，一些站点可操纵数千台计算机，并用于寄宿许多分布式应用程序。这些分布式应用程序通常都具备复杂的网络规格，其要求操作者将计算机与某网络交换机进行物理连接，并手工地调整数据中心内部的布线格局以支持复杂的应用程序。其结果是，构建物理网络拓扑结构以符合应用程序需求的任务极其麻烦，耗时的处

理易于导致人为错误。因此，需要对物理计算系统中分布式应用程序的设计和部署技术进行改进。

发明内容

描述了一种用于设计、部署和管理分布式计算系统中的分布式应用程序的体系结构和方法。

附图说明

在附图中用相同数字标记指代相似特征。

图1示出了网络设置的示例。

图2是示出了使用SDM定义模型的示例性体系结构的框图。

图3示出了在SDM模式设计规范内限定的示例性定义结构。

图4示出了在SDM模式设计规范内限定的示例性关系结构。

图5示出了将WEB应用程序映射到WEB服务器宿主上的示例。

图6示出了使用SDM运行时的示例性体系结构。

图7示出了示例性分层设定。

图8示出了示例性SDM文档。

图9示出了基本定义和成员。

图10示出了示例性成员。

图11示出了示例性设定值和数值列表。

图12示出了根据某些实施例的SDM应用程序的示例性生命周期。

图13示出了从WEB应用程序映射到WEB服务器宿主上的示例映射。

图14示出了示例性内嵌数据类型分层结构。

图15示出了抽象对象定义的隐含扩展示例。

图16示出了抽象关系的隐含扩展示例。

图17示出了变更请求的示例。

图18示出了将新定义载入运行时的过程示例。

图19示出了执行变更请求的示例。

图20示出了连接成员的示例。

图21示出了关于连接的示例性结构。

图22示出了提供实例空间（instance space）概述的UML图示例。

图23示出了能够用来执行此处所描述技术的总体计算机环境。

详细说明

以下所公开的内容描述了用于设计及实现具有大规模应用服务的分布式计算系统的体系结构的多方面内容。所公开的内容包括对系统定义模型（SDM）和SDM运行时环境的详述，其中系统定义模型（SDM）也称作服务定义模型。所公开的内容进一步包括设计特征，例如如何建立各种数据中心组件的模型。

在此处使用的术语“接线”也称为“连接”、“通信”或“通信关系”。同时，术语“系统”也称为“模块”，而术语“资源空间”也称为“资源”。另外，术语“应用程序空间”也称为“应用程序”，而术语“实例空间”也称为“实例”。此外，术语“类”也可以称为“抽象定义”，术语“端口”也称为“终端”，而术语“类型”也可以称为“定义”。

图1示出了网络设备100示例。在设备100中，多（x）个计算装置102（1）、102（2）、...、102（x）与网络106相连接。网络106表示多种常规网络拓扑结构和类型（包括有线和/或无线网络）的任一种，使用多种常规网络协议（包括公共协议和/或专有协议）的任一种。网络106可以包括，例如，局域网（LAN）、广域网（WAN）、因特网的一部分等等。设备100表示广义的多种网络体系的任一种，包括，例如数据中心（如因特网数据中心（IDC））、办公室或商务设备、家庭设备、教育或科研设施、零售或销售设备、数据存储设备等等。

计算装置102可以是多种传统计算装置的任一种，包括桌面PC、工作站、大型机、服务器、因特网设备、游戏控制台、便携式计算、移动电话、个人数字助理（PDA）等等。一个或多个装置102可以是相同类型的装置，或者也可以是不同类型的装置。另外，即使多个装置是相同类型的装置，所述多个装置也可能配置不同（例如，两个装置102可以均为服务器，但可具有不同的硬件配置，如不同的处理器、不同容量的RAM、不同尺寸的硬盘等等）。

一个或多个计算装置102在加入设备100后也可进行再配置。例如，一个特定的计算装置102可能操作了一段时间（如大约几分钟、几小时、几天、几个月等等）以执行某个功能，然后管理者决定需要执行不同的功能（如，从服务器变为工作站计算机，从Web服务器变为本地文件服务器等等）。

图2是一个框图，它示出了使用系统定义模型的示例性体系结构200。将SDM

设计为在系统的整个生命周期上被使用。系统是一组可以共同工作以完成常见功能的相关软件和/或硬件资源。这类系统的一个示例就是应用程序，其涉及一组可以被计算装置运行或执行的指令，以执行各种功能。应用程序的示例包括：诸如游戏之类的娱乐应用程序，诸如字处理软件之类的生产应用程序，诸如电子百科全书之类的参考应用程序，诸如可用于web服务或财政分析的分布式应用程序等等。所述系统的其他示例是工作平台，在其上可部署应用程序（或其他工作平台）。工作平台指的是在其上可以部署应用程序（或其他工作平台）的软件和/或硬件资源。正如下面所详细描述的，这样的工作平台可以分层。

系统的生命周期一般包括三个基本阶段（也被成为级）：设计或开发阶段，接下来是部署或安装阶段，接下来是操作或管理阶段。由于该模型应用于系统生命周期的所有三个阶段，其可视为是系统生命周期中各个阶段的整合点，并推进各个阶段。另外，通过使用模型，知识（knowledge）可以在这些阶段之间传输，例如涉及系统管理的知识（例如将该知识反馈到设计和开发组）（例如，因此允许设计和开发组修改系统，由此来创建更高级的版本或提高当前版本的性能）；系统的结构、配置需求和操作行为的知识；从桌面到数据中心的操作平台知识；由端用户监测的服务水平知识；等等。

通常来讲，在设计阶段中，影响SDM的开发工具用于定义由通信软件和硬件组件所组成的系统。系统定义包含部署和操作分布式系统所必需的全部信息，包括需求资源、结构、操作特征、策略等等。在部署阶段中，将系统定义用于自动部署系统和动态分配并配置要求的软件及硬件（例如服务器、存储器和网络）资源。相同的系统定义可用于部署不同的宿主环境和不同的规模。在管理阶段中，操作系统中的SDM服务提供用于管理系统的系统层视图。这使得新的管理工具从系统前景角度来驱动资源分配、结构管理、升级和自动处理。

体系结构200使用SDM定义模型以及在SDM定义模型内定义功能操作的模式。定义模型包括各种不同种类的数据结构，这些数据结构共同地被称为“定义（definitions）”。SDM的功能经由诸如应用程序接口（API）之类的一个或多个平台服务得以示出。

在系统的设计阶段中，开发系统202产生一个包含系统定义的文档，如SDM文档204。开发系统202可以是多个开发系统中的任一个，例如由Microsoft® Corporation of Redmond，Washington提供的Visual Studio®开发系统。SDM文档

204定义了所有涉及系统部署和管理的信息（此处也称为知识）。在部署系统或管理系统时所必须或所用的任何知识均包括在SDM文档204中。虽然此处描述的是单个文档，但应当知道作为选择知识也能被传播开并留存于多个文档中。

SDM文档204包括一个或多个系统约束（也称作需求），其中部署和/或运行该系统的环境必须满足。该环境自身也用SDM文档描述。这样的环境可以是单个计算装置、或者是计算装置的集合（例如数据中心）、应用程序宿主等等。不同的系统能够安装到不同的环境中。例如，数据中心可以包括五十台计算装置，并且一个系统可以被部署到五台这样的计算装置中，同时另一个系统可以被部署到三十五台这类计算装置中。这些需求可以是不同的形式，例如：与其上部署有所述系统的一个或多个计算装置相关的硬件需求（例如，最小处理器速度、最少量的存储器、最少量的空闲硬盘驱动空间、最少量的可用网络带宽、特定的可用安全机构等等）；与其上部署有所述系统的一个或多个计算装置相关的软件需求（例如，特定的操作系统、也必须安装的一个或多个其它应用程序、与如何部署一个特定的系统和/或操作系统相关的规范、使用中特定类型的安全或加密）；与其上部署有所述系统的一个或多个计算装置相关的其它需求（例如，特定的可用安全密钥、必须执行的数据中心策略、使用的验证、环境拓扑等等）。

需求也能够用于其它方向——也即，环境可以在要安装的系统配置上具有约束或需求（例如完成环境标准或策略）。这些可以是由环境操作者建立的“显示”需求，例如系统必须具有的特定设置或配置、系统必须提供或支持的特定功能、系统必须支持的特定安全机构等等。这些也可以是由于环境的特定配置所引起的“隐式”需求。例如，如果环境中的宿主计算装置正在使用特定类型的文件系统，那么它将不能够执行使用该文件系统的一些操作（尽管它能够使用其它文件系统来执行那些相同的操作）。

在系统设计和开发阶段中，SDM文档204能够用来验证用于一个或多个特定环境的系统。这是一种双向验证：为环境验证系统，为系统验证环境。对于系统而言，能够通过将SDM文档204内确认的需求与环境相比较并确定是否所有的需求均被环境满足来验证环境。对于环境而言，能够通过将SDM文档204内为环境确认的需求与系统相比较并确定是否所有的需求均被系统满足来验证系统。如果所有的需求均被环境和系统满足，那么设计者或开发者知道该系统能够在

该环境中部署和运行。然而，如果并不是所有的需求均被环境和/或系统满足时，那么设计者或开发者可选地被告知不满足的需求，从而告知设计者或开发者应当对SDM文档204（并且相应对系统）和/或环境进行什么改变，以便使得系统在该环境中部署和运行。

与在SDM文档204内包括的系统部署相关的知识描述了系统如何在一个或多个环境中部署。该SDM文档204对控制器206来说是可用的，它包括部署模块208和管理模块210。在某些实施例中，SDM文档204以及安装系统所需的所有系统文件（例如二进制、数据、程序库等）被一起打包成一个称为SDU（系统定义单元）的单个容器（例如单个文件）。控制器206可以是图1中的一个或多个计算装置。例如，图1的单个装置102可以是特定数据中心的控制器，或者，所述控制器功能可以分布在多个装置102上。

部署模块208包括用于在一个或多个环境中部署系统的服务程序。在图2中，部署（或在其上部署）系统的环境是一个或多个目标装置212。系统也可以部署到控制器206上。部署模块208的这些服务程序包括一个或多个功能，其中该功能能够在环境中被调用或被请求以便来安装或部署一个或多个系统。

在不同环境中用于部署的不同知识可以被包含在SDM文档204中。该部署知识描述了需要对环境进行的任何变化（例如系统注册变化；需要建立的文件夹、目录或文件；需要设定到特定值的计算装置的其它设定或部署参数；等等），以及在环境中需要复制到一个或多个计算装置的某些文件（例如程序和/或数据文件）和需要对那些文件执行的任何操作（例如一些文件可能需要被解压缩和/或解密）。在许多执行程序中，SDM文档204中的部署知识包括，例如与目前可知的系统典型安装或安装程序相类似的信息。

在部署过程中，控制器206产生在部署中涉及的软件和硬件资源的记录或存储，并产生它们之间关系。该记录或存储能够在管理阶段中依次被控制器206使用。

一旦管理模块210在一个或多个环境中安装，它就包括了用于管理系统的服务程序。这些管理模块210的服务程序包括能够在环境中被调用或被请求来管理系统的一个或多个功能。与包括在SDM文档204内的系统管理相关的知识描述了在一个或多个环境中如何管理系统。

在不同环境中管理系统的不同知识可以包括在SDM文档204中。该管理知识

包括在系统管理或操作中所用的任何知识。管理包括例如配置（和可选的相继再配置）、修补及升级、维护任务（例如备份）、健康或性能检测等等。

通过管理模块210来进行对已部署的系统的改变。管理模块210的服务程序包括能够被调用或被请求来对环境中部署的一个或多个系统进行改变的一个或多个功能。通过对管理模块210进行如此的改变，能够实现几种好处。一种好处是控制器206能够保留已经进行了改变的记录。控制器206可以为系统保留一份SDM文档204并且在该SDM文档204中记录对系统进行的任何改变。或者，控制器206可以保留对系统所进行改变的单独记录。

这种被控制器206保留的变化记录能够简化后续操作，例如解决系统和/或环境的问题，或者当由于硬件差错不得不重新安装系统时（允许系统被重新安装并返回以在出错时所具有的相同参数/设定运行）。通过具有经由控制器206所进行的这些改变并且通过具有保留记录的控制器206，一些人为错误能够从环境中去除（例如，如果管理员进行了改变认为已将该变化记入书中，但忘记了这样做，那么将不会有改变的记录——该问题通过具有保留记录的控制器206得以解决）。

而且，通过经由控制器206对系统进行改变，以及经由处理器206部署系统，控制器206能够作为关于环境、在环境中部署的系统以及它们之间交互的知识仓库。与环境和/或在环境中部署的系统相关的知识能够容易地从控制器206获得。通过使得环境中受控装置反应在中间控制器206内存储的状态有效，该知识能够用来确保受控环境的一致性。

应当指出在一些情况下，可以对系统和/或环境进行改变而不经由控制器206。例如，计算装置可能被意外关闭或出差错。在这些情况下，在控制器206内进行尝试来反应这种变化。这些变化可以在控制器206内自动被反应（例如系统可以运行上述尝试来检测装置差错并且使用管理模块210的服务程序来通知控制器206这种差错）或者可以在控制器206内手动经反应（例如管理员可以使用管理模块210的服务程序来通知控制器206这种差错）。可选地，所进行的变化能够倒转来将系统和/或部分环境带回到具有如控制器206所记录的系统所需状态的程序行。

SDM文档204从而能够被视为“活的”文档——它能够在整个系统生命周期内基于环境变化和/或系统变化而不断变化。

系统定义模型（SDM）

系统定义模型（SDM）是一种用来创建系统定义的建立模型技术。系统是一套一起工作来完成共同功能的相关软件和/或硬件资源。实例系统包括多端在线商业应用、网络服务、网络商务站点、企业数据中心。SDM为应用体系结构、网络体系结构、数据中心体系结构或其他开发者提供工具和环境，从而以抽象方式来设计分布式计算机应用程序和数据中心。SDM定义代表系统功能单元的一组元素并且最终通过物理计算机资源和软件来实现这些功能单位。SDM也定义与要管理系统的操作者或其他个人相关的元素。另外，SDM捕获与开发、部署和操作相关的数据。与SDM元素相联系的是这样一种模式，该模式定义了由组件代表的功能操作如何去规定。

系统由资源、终端、关系和子系统组成。这些项目中每一个的定义在SDM文档中说明。SDM文档是包含一个或多个定义系统、资源、终端和关系的XML文档。资源可以是硬件资源或软件资源。终端代表在系统中交叉的通信。关系定义系统、资源和终端之间的联系。子系统能够作为完整的系统处理并且一般是更大系统的一部分。

系统定义捕获动态系统的基础结构。它能够被视为所有其它信息在其上添加的框架。该结构一般在开发过程中被结构体系和开发者规定，并且一般常常不进行改变。除了结构外，SDM可以包含部署信息、安装过程、配置模式、事件和装置、自动任务、健康模型、操作策略等等。在分布系统生命周期内，其它信息能够被操作人员、卖主和/或管理系统添加。

SDM模式设计说明

在分布系统（例如模型系统）中，SDM被设计用来支持对组件的结构、交互作用和变化的描述。“定义”描述了存在于系统中的实体并且“关系”确认不同实体之间的链接。定义和关系进一步被定义来捕捉与SDM相关的语义信息。如图3所示，SDM定义302包括三个分定义：资源定义304、系统定义306和终端定义308。

如图4所示，SDM关系402包括五个分关系：包含关系404、代理关系406、连接关系408、寄宿关系410、参考关系412。连接关系408也可以称作“通信关系”。与定义和关系相关的进一步详细内容在下面提供。

SDM包括提供系统部件共同分类、为宽范围应用程序提供工具支持及在设

计时为定义检查提供基础的“抽象定义”。一组抽象定义为服务设计提供全面基础。“具体定义”表示实际应用部件或数据中心设计。具体定义通过选择抽象定义并且提供执行功能而产生，其中该执行功能定义具体定义的成员和特性设定值。使用这些具体定义的集合来产生分布式应用程序。

SDM也包括在允许的组关系的基础上的建模限制的“约束”，其中关系实例能够参与允许的组关系。约束对描述需求是有用的，这些需求依赖于在关系中涉及的对象结构。例如，约束可以用于确定位于通信协议每端的参与者是否在使用兼容的安全设定。

流可以被视为定义和/或资源的一部分。该流通过传送操作者设定到系统或其它使用这些设定的组件去，用于在运行时控制应用程序行为。

抽象定义和关系

抽象定义限定多个模块，这些模块在设计时检查应用程序结构并且在运行时部署和管理应用程序。这些模块代表存在于模型系统中的实体。例如，抽象定义能够对文件和文件夹、WEB服务器结构、或SQL服务器内的数据库进行建模。

抽象关系对能够在抽象定义之间发生的交互作用建模。关系是二元的或定向的，确认参与关系表现的实例的定义。关系提供了实体之间相互关联的方法，因此允许约束、结构以及实体之间通信链路的建模。

约束被定义使用来限制它们参与的关系。约束进一步被关系使用来限制能够被链接的定义。在关系内这些约束能够把定义和参与者设定作为目标。

抽象定义空间被分为三类：组件、终端和资源。抽象系统定义描述应用程序的自包含独立部署部件。这些定义代表了通过明确的通信通道交互的应用程序部件，其中的通信通道能够横穿过程和机器界线。抽象终端定义描述了组件可以示出的通信终端。这些抽象终端定义能够对系统知道的所有形式通信建模，从而在设计时来验证系统连通性并且在运行时实现连接。抽象资源定义描述了在组件内限制的行为。资源定义可以具有对其它资源定义的很强依赖性。这些依赖性可以包括需求特定的安装顺序和经由不同通信机构启动运行时交互作用。

抽象定义包括陈述设置的能力。在一种实施例中，这些设置为多个使用XML模式来限定设置定义的命名值（name-value）对。设置可以为动态的或静态的。

静态设置在部署过程中设定。动态设置可以在部署后改变。负责将设置值应用到运行系统的代码在SDM运行时内寄宿。

SDM模型支持在抽象定义上的继承。衍生定义能够延伸由它的母体陈述的性能并且能够为它的母体性能设定值。衍生定义能够参与进入将它的母体视为参与者的关系中。

如上所述，关系分为五类：通信（或连接）、包含、代理、寄宿和参考。通信关系捕捉在抽象终端定义之间的潜在通信相互作用。通信关系的存在表明示出确认的定义的端点的组件能够相互通信。连接的实际建立受终端上的约束和终端的示出的支配。

包含关系描述了抽象定义包含其它抽象定义成员的能力。更具体的，两个抽象定义A和B之间的包含关系允许执行A的具体定义去包含执行B的定义成员。约束关系对当开发者建立应用程序时发生的固有嵌套结构建模。通过包含另一个定义的成员，母体能够控制生命周期和所包含定义的可见性。在运行时空间内所有定义实例作为其它定义实例成员存在，形成完整地相连接的实例组。从而该组包含关系描述了在运行时空间内发生的允许的包含模式。

代理关系可选择性地陈述所包含的成员。例如，代理能够从组件定义内陈述终端成员。通过从内部组件代理终端，外部组件陈述出使用特定协议通信而不在协议后陈述装置的能力。

寄宿和参考关系代表两种形式的依赖关系。寄宿关系被用来捕捉与如何在特定宿主上建立定义实例相关的知识。该寄宿关系允许开发者以不依赖于特定宿主操作的方式生成他们自己的定义。该关系也允许单个定义在多个宿主类型上部署而不重写客户定义。该寄宿关系描述了在生成具体定义实例前存在的抽象定义之间的最初依赖性。每个实例在寄宿关系内作为客户进行参与，从而致使寄宿关系在实例空间内形成连接树。参考关系捕捉用于参数流和结构排序的附加依赖性。

具体定义和关系

具体定义从抽象定义中创建。具体关系从抽象关系中创建。抽象定义和抽象关系的组合限定了用于对目标系统建模的模式。具体定义使用抽象定义空间的子集来生成一个或多个抽象定义的可再使用结构。该抽象定义空间能够比作用于数据库的模式。在该类比中，该具体定义空间代表在数据库中用于一组行

列的可再使用模板。该具体定义相对抽象定义空间生效的方式与数据库中行列相对于模式约束生效的方式相同，例如外部关键词等等。开发者能够从抽象定义知识中推断具体定义知识。从而，与抽象定义相联系的工具能够用从抽象定义衍生而来许多执行工具进行操作。例如，知道抽象网络服务的工具能够随着任何部署入数据中心的网络服务来进行操作而不需要来自开发者的额外信息。

每个具体定义都提供了用于特定抽象定义的执行工具，该抽象定义包括设定模式扩展、设置值、定义及关系成员的声明以及对定义能够参与的关系的约束。具体定义的行为遵从抽象定义的定义。具体来讲，抽象组件定义变为组件定义，抽象终端定义变为终端定义，以及抽象资源定义变为资源定义。

每个具体定义都提供了用于特定抽象关系的执行工具，该特定抽象关系包括设置模式和设置值、相同关系类别的嵌套成员（例如寄宿、包含或通信）、以及对能够参与关系的定义的约束。

具体寄宿关系限定了能够将一个具体定义成员映射到另一个具体定义上的一组寄宿关系。例如，具体寄宿关系能够确认在WEB应用程序和将要部署的IIS宿主之间的捆绑。对于一个具体限定能够存在一个以上的寄宿关系，从而允许开发者为具体拓扑限定部署。

具体定义能够声明其它具体或抽象定义成员——称作“定义成员”。这些定义成员然后从在定义成员之间定义关系的“关系成员”中被参考。定义成员包括对特定定义实例的参考。设置流能够提供用于定义的值或者能够约束当创建定义时使用的结构参数。当声明定义成员时，使用者（例如开发者）能够决定定义成员是否在外部组件创建同时被创建（称作“值语义”）或者定义成员是否由在后面时间发生的明确的新操作被创建（称作“参考语义”）。

当关系成员被创建时，它们限定定义成员将参与的关系。如果定义成员包含在具体定义内，那么包含关系成员在定义成员和用于外部定义的该参考之间被声明。如果定义成员被代理，那么代理关系成员将在定义成员和嵌套定义成员之间限定。通信关系成员可以在处于定义成员上的终端之间说明，并且依赖关系成员（参考和寄宿）可以在定义成员或嵌套定义成员之间声明。

关系约束收缩了特定定义将参与的所述组关系。关系约束确认处于特定关系及处于关系另一端的参与者上的约束。

实例空间

在SDM运行时内存储的实例空间确认模型系统的当前状态。SDM运行时包含已经创建的实例和这些实例之间关系的记录。每个实例具有将每个版本与变更请求相链接的相关版本历史。变更请求是创建新实例的过程。变更请求定义了用于定义的一组创建、升级和删除请求以及与现存实例具体成员相联系的关系。所述根作为特定情况来处理。

变更请求由运行时扩展、相对一个或多个约束进行验证并且随后被构建。扩展过程确认定义和关系实例，该关系实例作为包含定义的结构请求部分被隐式构建。作为扩展过程的一部分，在所有关系上评估设置流。验证步骤检查所有需要关系存在以及这些关系满足必要约束。最后，构建过程确定每个实例的部署、升级或删除的合适排序。构建过程然后以正确的顺序将每个实例传送到实例管理员去执行适宜的操作。

数据中心能够利用多个软件组件加以创建。在多个软件组件之间配置一个或多个连接。这些软件组件中的一些可以起应用程序层宿主的作用。在宿主层内的实例组件定义包括IIS、SQL、AD、EXCHANGE、DNS和Biztalk。

网络/OS/存储层支持数据中心网络和平台结构。该层也支持网络安全模型构造、操作系统平台构造以及一个或多个具有操作系统平台的存储装置的联合。在网络/OS/存储层内的实例组件定义包括VLAN、视窗、过滤器和存储器。

硬件层确认存在于数据中心内的系统定义以及在这些系统之间存在的物理连接。为了满足由特定组件所需的关系，所述组件被绑定于具有匹配性能的宿主组件。该过程称作“逻辑放置”。在部署时，客户组件实例被放置在宿主组件实例位置上。该过程称作“物理放置”。

图5示出了将WEB应用程序502映射到WEB服务器宿主504上的实例。由虚线506标明的边界将应用程序层从宿主层分开。WEB应用程序502包含远程服务终端508、WEB服务终端510和WEB使用者接口终端512，所有这些终端都映射于WEB服务器504中的WEB站点522。另外，在WEB应用程序502内的虚拟文件夹516映射到WEB站点522。WEB应用程序502也包括映射于WEB服务器504内消费者终端524上的消费者终端514。WEB应用程序502的组合体（Assembly）518映射于WEB服务器504的运行时部分528。在WEB应用程序502内的内容文件夹520映射于WEB服务器504内的WEB根文件夹526。

管理分布系统改变的过程与SDM模型相联系。分布系统改变由变更请求驱

动，该变更请求在实例内的动作相对目标系统被分布和执行之前经过一个或多个处理步骤。

图6示出了使用SDM运行时的实例体系机构600。体系机构600是使用SDM运行时610的图2中体系机构200的实例以及在下面“实例实现”部分讲述的SDM的实例实现。SDM运行时610包含一组组件和过程，用于接受及确认SDM文件、载入SDU（多个系统定义单元——它们是一个或多个SDM文件及它们的相关文件的程序包）、创建及执行SDM变更请求、以及部署以SDM为基础的系统为目标环境。运行时功能性允许使用SDM描述的系统被限定和验证、部署到一组计算装置上并被管理。

下面是图6中组件如何一起工作的功能性简述。操作者或管理员能够描述能被部署的应用程序的环境，例如数据中心的拓扑。操作者或管理员生成描述环境的SDM文件，该文件称作“逻辑下部结构”（LIM）602，或者称作数据中心描述或数据中心模型。该SDM文件能够利用各种不同开发系统中的任一个来产生，比如像由Microsoft® Corporation of Redmond，Washington提供的Visual Studio®开发系统等。

另外，应用程序开发者能够使用各种不同开发系统中的任一个来设计和开发他们的应用程序，比如像Visual Studio®开发系统等。当开发者定义应用程序组件以及这些组件如何相互联系时，开发者能够相对数据中心描述602来验证应用程序描述。这也称作“设计时验证”。

一旦应用程序完成，开发者在SDM内保存描述并且请求该应用程序为部署打包成SDU 604。该SDU包括应用程序SDM以及二进制的应用和其它用于安装应用程序的参考文件。

LIM 602和SDU 604被馈送至用于部署的控制器装置620的部署工具606。部署工具606包括用户接口（UI）以使得操作者能够载入所需的SDU604。部署工具606与创建CR模块630一起工作以便在SDU604内根据SDM中的信息安装与SDU604相关联的应用程序。另外，SDM定义和来自SDU604的实例装入SDM运行时610的存储器608内。SDU在SDM 610运行时内由SDU管理模块640管理，这使得SDU的合适部分对运行时610和一个或多个目标622的其它组件可用。

操作者也能够指定他或她想对目标622（例如目标计算装置）采取什么动作，其中目标622上部署了应用程序。操作者能够通过部署文件来做这件事，这一点

在此处也称为变更请求（CR）。CR通过一个或多个引擎612、614、616和618运行。总体上，扩展CR引擎612扩展CR来确认所有相关联的组件以及它们的连接和动作，流动值引擎614流动用于组件的数值（例如连接字符串），检查约束引擎616检查环境和应用程序之间的约束，顺序动作引擎618为CR的所有必要动作指定顺序。

为了启动系统的改变（包括部署应用程序）或者验证模型，操作者或者过程提交一个CR。该CR包含操作者想在运行时610内对实例执行的一组动作。这些动作可以是例如创建动作、升级动作、和/或删除动作。

除了使用者或操作者启动变更请求外，也可以有扩展/自动生成的变更请求，这些变更请求作为扩展过程的一部分产生，将在下面详细讲述。不管它们的资源，一旦被完全扩展和检查，变更请求就通过传送动作到目标622而被执行，例如：发现、安装、卸载和改变目标实例。

将CR作为完成或失败动作的原子组来对待。例如，当检测有效性时这允许检查约束引擎616考虑所有的动作。

在设计时确认内，CR将由SDM编辑器628创建并且将包含在SDM文件内每个SDM组件中的一个或最小值。创建实例命令中的所述CR将流经扩展引擎612、流动值引擎614和检查约束引擎616。在这三个阶段发现的错误将通过他或她使用的部署系统返回给用户。

在部署过程中，操作者将与由部署工具606代表的UI一起创建CR。该CR将在SDM运行时610内流经所有引擎612、614、616和618，并且适当的动作和信息将由CR模块632传送到适当的一个或多个目标622，在此处该实例被执行（例如应用程序被安装）。用于特定安装的适当的一个或多个目标622一般是那些应用程序将被安装的一个或多个目标。

当开始处理CR时，在定义分解阶段，创建CR模块630分解变更请求中所参考的所有定义和成员。该变更请求将假定这些已经由运行时610载入；如果它们不存在，创建CR模块630就启动载入/编辑动作。创建CR模块630也执行路径分解阶段，其中对现存实例的参考和在变更请求内由创建动作定义的实例被分解。

由扩展引擎612执行的扩展为一个过程，该过程中给定一个变更请求，填充所有所需去执行请求的保留动作。总的来说，这些动作为用于定义和关系实例的构建和解构动作。操作者能够为所有构建或解构实例所需的动作可选地提供

细节，或者作为选择，可以将部分过程自动化：例如操作者通过确认成员动作（例如参考成员）来提供他或她所想改变的关键信息，并且将其余动作加载在嵌套成员（例如参考和数值成员）和关系上。作为另一个示例，自动扩展也能够参考外部资源管理者，这些外部资源管理者可以基于选择有可用资源的设备进行部署决定、定位接近其需要数据的应用程序等等。

扩展引擎612也执行“自动写入”。在自动写入过程中，引擎612分析规模不变的组件分组和在SDM内指定的复合组件并且确定当达到所需水平时组件应当如何分组和互联。

扩展引擎612也执行数值成员扩展、参考成员扩展（发现）和关系扩展。

数值成员扩展指所有非参考定义成员的确认。这些成员的基数被指出并且由于知道了所有的所需参数，对于每个成员，生成请求被添加到母体正在被创建的那些成员的变更请求上。如果该变更请求包含解构操作，那么为所有它们的包含实例添加解构操作。

参考成员扩展指参考成员（作为非参考定义成员的相反面）。通常，不对参考成员的基数进行限定，并且它们可以具有需要数值的部署时间设定以便构建实例。因此扩展参考成员（例如参考成员）的过程能够比运行时处在提供位置需求更多关于实例的信息。

与参考成员扩展相关的是被称为发现的过程，该过程为用于发现已经部署的实例的过程。发现一般是由环境操作者启动的动作。例如，在安装请求过程中，扩展引擎612确定实例是否已经存在，如果存在则确定存在什么，如果不存在则创建它。控制器装置620上的实例管理者（IM）634与目标装置622上的实例管理者626相通以便启动发现过程。该发现过程从目标装置622将与实例相关的数据返回给控制器装置620。

发现过程填充参考定义成员作为构建或升级动作的一部分。一般仅仅带有支持发现的对象管理者（实例管理者也进行发现）的参考成员参与该过程。

当新的实例被发现时，进行检查使得实例已经不在使用实例特定密钥值的SDM数据库内存在。一旦知道那是一个新的实例，根据被发现的成员定义来分类该实例。如果该实例不与成员匹配或者存在不明确的匹配，那么让成员参考为空并且该实例作为脱机和不完全标记。

关系扩展指一旦知道将构建的所有定义实例，创建将定义实例捆绑在一起

的关系实例。如果定义实例被解构，所有参考定义实例的关系实例被删除。

为了创建关系，成员空间用来确认应当在实例之间存在的关系的构造。定义成员具有基数比关系中一个拓扑更多的地方从基础关系定义中推断出。例如，对于通信关系能够进行“自动连线”，并且对于寄宿关系基于与寄宿关系相关联的算法来选取宿主。

在流动阶段，流动值引擎614评估经过所有关系实例的流。流动值引擎614可以将升级请求添加到用于实例的变更请求上，其中这些实例被任何改变参数流动影响。作为变更请求的结果，引擎614通过确定具有升级设定的实例组来计算流动。对于这些中每一个，依赖于修改设定的任何输出设定流动被计算并且将目标节点添加到改变的实例组上。该过程持续到该组为空或者该组包含一个循环为止。

在流动阶段后，执行重复检测过程。重复检测可以由图6所示的引擎中的一个（例如流动值引擎614或者检查约束引擎616）来执行，或者可选地由图6中未示出的其它引擎（例如重复检测引擎可以包括在SDM运行时610内）执行。重复检测过程相对在SDM数据存储器中已经存在的实例与扩展实例相匹配。例如，该过程检测是否另一个应用程序已经安装了共享文件。当已经存在的实例被检测到时，依赖于现存实例的版本可以采取几个动作中的一个：安装可能失败；可能参考计算实例；可能升级实例；或者可能并行地执行安装。

检查约束引擎616执行约束计算阶段，在该阶段中检查模型内所有约束来察看在变更请求已经处理后它们是否仍为正确的。

在检查约束引擎616完成约束计算阶段后，可以获得一完整的动作列表。因此，顺序动作引擎618能够使用在组件之间的关系来确定正确的改变排序。可以使用任何一种算法进行该确定。

一旦确定排序的顺序动作引擎618完成，就通过分配机器特定的动作排序组的分组来进行部署。一旦这些动作已经由机器排序和分组，就把动作以及带有实例信息的SDM运行时存储器608必要部分的拷贝送到目标计算设备622。SDM能够在目标装置处暂时存储在存储器缓存638内。

目标计算设备包括与SDM运行时610相通的SDM运行时的目标部分636。目标计算设备622也包括包含执行引擎624的代理并且能够与目标装置上的适当实例管理器（IM）626相通以便在目标上进行改变，比如创建、升级和删除动作。

每个动作被作为原子调用送到实例管理器626，并且实例管理器626返回状态信息并且对于一些动作也返回数据（例如对于发现）。一旦在目标622上完成了所有的动作，目标的代理就返回任何差错和状态到控制器装置620。控制器610然后使用该信息升级SDM运行时存储器608。

如上所述，基于受影响的关系，通过中断变更请求进入可分配的部件来执行改变。一旦完成了所有部件（或者在一个或多个已经失败后），就在运行时610内对照所述结果，并且将总结返回给操作者。在失败的情况下，所有动作能够被“退回”并且系统返回到启动改变前所处的状态。

在某些实施例中，在上述设计时验证过程中，SDM编辑器628接收SDM文件，创建测试CR，通过扩展、流动值运行测试CR并且检查SDM运行时的约束引擎，并将任何错误返回到开发系统去。该过程在设计时内为开发者提供用于部署的SDM验证。

SDM运行时610和/或控制器装置620的公共接口贯穿对象模型（API）程序库。该程序库为管理代码对象模型并且允许执行下面内容：

● 在运行时内管理SDM－SDM文件能够载入运行时。SDM为不变的并且一次载入一个（例如，一个SDM文件能够载入而并非仅仅部分文件能够载入（例如单个定义、分类或从SDM文件映射中的单个））。能够从运行时删除SDM并且能够生成在运行时内用于SDM的XML文档。

● 管理由运行时所知的SDU。

● 管理SDU定义－（在运行时内从载入的SDM中）发现并考虑SDM元素。没有提供来创作新SDM的公共API（例如这是在SDM不变元素上的只读对象模型）。这包括SDM、SDU、确认、版本、分类、定义、捆绑/映射和版本策略。

● 管理SDM实例－发现并考虑组件、终端、资源和关系实例。在实例空间内每个实例能够由GUID、固定路径或阵列基础路径来进行标识。这些路径为字符串并且可以是相对的。这些标识符包括相对路径允许实例被发现并在文档例如变更请求文档中被参考。

● 操作实例－对SDM实例进行改变，包括创建、改变拓扑、升级、改变设定和删除。在提供原子单元更新的变更请求范围内进行实例改变，使得任何错误或者约束破坏导致整个请求失败。由于当提交请求时实例必须具有宿主，因而实例请求也允许实例暂时存在而不与宿主捆绑。它也允许执行影响单个组件

安装或设定的许多操作并且使得安装或设定更新延迟直到提交为止，从而使得单个更新在组件上进行。SDM模型检查优先于或在变更请求提交时间执行并且该提交将在任何模型或违反约束上失败。

● 载入变更请求—变更请求是一个文档，例如XML文件，代表了一组实例空间操作。该文档能够利用相对路径成为可再利用的“脚本”用于创建或删除应用程序实例。

● 发现和考虑变更请求—包括获得安装/升级任务和所有的错误信息，并且重试受请求影响的组件的安装/升级。

● 在数据库中从变更请求中生成变更请求文档。这些文档是稍微可移动的。

● 预订关于变更请求任务的事件，例如升级的进程、记录或更新的状态。这些事件预订的生命周期受到装载客户库过程的生命周期限制（例如这些为常规的CLR事件）。

这些运行时引擎在SDM模型上执行推理以及执行由API加上表面的功能。该程序库与运行时引擎相通作为带有相当粗略调用的网络服务，例如载入SDM、创建组件实例和获得整个SDM（用于考虑SDM实体）。该网络服务的许多参数的格式为具有与SDM文件相同模式的XML。引擎也可以对许可执行检查。

控制器装置620能够使用实例管理器（IM），这些实例管理器可以在模型内与任何定义或关系相关联。IM可以执行一个或多个下述的任务：

- 支持实例部署。
- 一旦已经被部署支持实例的验证（审核）。
- 支持发现在运行时内没有被部署的已经部署过的实例。
- 支持设置值流动。
- 支持约束的计算。
- 支持变更请求的扩展。
- 支持通过API作为CLR分类向用户提供实例。

对于部署，在控制器装置620上的实例管理器（IM）插件与分类宿主关系相关联并且与在开发系统内使用的插件分开，其中该开发系统为分类提供设计经验并且在SUU604内生成相联系的边界和设置模式。实例管理器作为CLR分类供应给SDM运行时610（例如在dll集合内），其中CLR分类执行实例管理器接口或从抽象分类继承而来。SDM实例管理器也称作实例管理器（IM）插件，为控制

器装置620提供下面的功能：

● 生成文件和命令（任务）在它们的宿主上安装、卸载或重新安装组件实例－当变更请求产生新的组件实例、删除组件实例或需要卸载和重新安装的组件改变时，正是实例管理器获得用于实例、宿主实例、与组件相关联的定义以及与SDU604内那些定义相关联的二进制的设置，并且生成用于在目标服务器上执行安装或卸载的文件和命令，准备好用于手动执行或通过部署引擎分派。

● 当组件实例的设定改变或者当从它的终端中一个看改变时（例如由于通信关系拓扑改变或者可见终端设定改变），生成文件和命令（例如任务）来更新组件实例。

● 将在组件实例终端上的可见终端实例映射到在组件实例上的设定－在SDM内组件实例具有终端实例，作为一些通信关系拓扑的结果，这些终端实例能够看到其它终端实例。其它终端实例的细节被映射到组件实例能够在运行时得到的设定上，通常使得两者之间相捆绑。例如，WEB站点可以具有数据库客户终端实例，因此与数据库建立通信关系。当正确建立时，它的数据库客户终端能够看到单个数据库服务器终端实例和在那个服务器终端上的设定。该信息被实例管理器用来在客户终端的名义下为服务器将连接字符串放入结构文件内。最终结果是代码仅仅从它的结构设置中为数据库读出连接字符串。

● 生成文件和命令（任务）来审核组件实例－审核确定存在、正确的设定。这也可以应用程序到宿主实例设定上去。

● 对于任何任务都将报告状态－IM将翻译已捕捉的部分或完整的输出，并且提供成功、失败或不完整的任务状态，并且可选地提供未完成的进程（%或最后响应）、失败细节（错误信息）和任何状态人工可读的记录。通过返回实例管理器来解释任务输出，在保持人工可读的过程中，实例管理器可以自由地获得其任务记录结构化的信息（例如，作为XML或者甚至SOAP），而不用试图不得不为诊断产生足够的记录。

● 实例管理器也可以提供在宿主和它们的客户机之间进行约束检查的代码。安装者可以使用公共约束语言，例如基于XML、XPath和XPuery。

分层

SDM模型提供应用程序开发者、软件基础结构和数据中心体系结构之间关注事物的分离。这些组中每一个都集中在特定服务上并且具有不同组的关注事

物。例如，开发者主要关注结构和它们所使用的宿主之间的连接性，例如SQL、IIS和CLR。宿主结构设计者主要关注网络拓扑和OS结构。开发网络拓扑、OS结构和存储映射的体系结构主要关注在数据中心中存在的硬件。

SDM使得系统的功能组成跨越水平和垂直轴。沿着水平轴的组成由系统和子系统处理。沿着垂直轴的组成由“层”处理。应用程序、服务、网络拓扑和硬件在分布式系统中实现功能，但是一般单独限定并且由不同的组或结构拥有。分层由在宿主上限定一组约束的组件完成，反之亦然。

为了支持这种对关注事物的分离，SDM陈述出分层的概念。分层是指：利用寄宿关系来将分层依赖的应用程序捆绑到服务上，而不声明作为应用程序包含结构一部分的那些服务。分层允许由不同的个体以不同次数和不同水平来部署系统。

图7示出了实例分层设定。在图7中示出了四层：层702、层704、层706和层708。虽然在图7中示出了四层，但是层的实际数量是可以改变的，而且可以大于或小于四。另外，不同层的内容在不同实施例中也可以改变。图7中可以看到，不同层位于其它层上面和/或下面（例如层706在层704上但在层708下面）。

在层内的不同系统和子系统可以互相作用，并且也可以与不同层的系统和子系统相互作用。例如，层708内的子系统710可以与层708内的子系统712以及层706内的子系统714相互作用。另外，每个层可以视为下一个更高层的环境。例如层706为层708内的系统和子系统的环境，而层704为层706内的系统和子系统的环境。每个层702、704、706和708具有它自己相联系的SDM文档。

不同层702、704、706和708可以表示不同内容。在某些实施例中，层702为硬件层、层704为网络拓扑和操作系统层、层706为应用程序宿主层、以及层708为应用程序层。硬件层代表分层系统在其上建造的物理设备（例如计算设备）（例如图1中设备102）。网络拓扑和操作系统层代表计算设备（例如图1中的网络设置100）的网络拓扑以及在那些计算设备上安装的操作系统。应用程序宿主层代表在可以寄宿其它应用程序的计算设备（例如SQL服务器、IIS等等）上安装的应用程序。应用程序层代表在不寄宿其它应用程序（例如，诸如游戏之类的娱乐应用程序、诸如文字处理器之类的生产应用程序、诸如电子词典之类的参考应用程序、诸如可以用于网络服务或财务分析的应用程序等这类分布应用程序等等）的计算设备上安装的应用程序。

SDM实现示例

以下详细描述了定义SDM元素的模式的实施例。

1定义

Term	Definition
Change Request	A declarative document that describes a set of changes to a modeled system
Fully qualified change request	A change request that has passed through the model evaluation stages and is now ready to be executed against the target system
Abstract type	A type that is used to define the settings required to act on a modeled system object
Concrete type	A reusable definition of a modeled system object that contains definitions for member types and relationships.
Relationship	An sdm object that is used to describe the interaction between modeled system elements
System Definition Model (SDM) Document	An xml document that contains definitions for abstract objects, concrete types and relationships
Software Distribution Unit (SDU)	The combination of a set of SDM documents and the associated binary information (files) required to deploy those types to an SDM managed system
SDM Layer	A layer is a set of abstract objects that are specific to the objects modeled in that layer. For example the application layer types may include Web application and Database, while the operating system layer may include types for file systems and network devices. Some types will not be assigned to layers and will instead be usable across a range of layers.
SDM Instance space	A set of concrete type and relationship instances that represent the modeled system

2 体系结构综述

该系统定义模型（SDM）被设计成能支持分布式系统（模型系统）中对结构、交互和组件变化的描述。

SDM基于对象关系模型。我们使用对象来描述存在于系统中的实体并确定它们之间的链接。SDM进一步限定对象和关系来捕获对SDM来说重要的语义。具体来讲，我们将对象分为系统、终端和资源，我们将关系分为传递、包含、寄宿、代理和参考。

我们使用抽象定义来提供系统部分的一般分类，在设计时允许支持大量应用程序的工具，并提供类型检测的基础。我们期望该组抽象定义提供用于系统设计的综合基础，并期望它们随时间变化缓慢。

我们构造代表实际请求或数据中心设计的具体对象定义。我们采用抽象对象定义，并提供定义具体成员类型的工具，还设定其特性的值。然后从这些定义的集合中构造系统。

约束用于在所允许的、实例能够加入的关系组之上为限制建模。我们使用约束来捕获依赖于关系中所涉及的对象构造的细粒需求。例如，约束可以用来验证位于使用兼容安全设定通信协议的每个终端的参与者。

为了对目标系统产生影响，SDM使用被称为变更请求的所需变更的说明性描述。SDM定义过程，其用于展开、确认和执行作为SDM执行模型一部分的变更请求。

实例空间即捕获期望也捕获应用程序管理的当前状态。我们跟踪实例空间的变化，并将其与引起变化的变更请求相关联。

如下的UML图捕获sdm模型中各对象之间的宽交互。为了简化起见，这些交互中的一些已经在基本类型之间进行限定，其中此处实际交互存在于衍生类型之间，并且该结果更为具体化。例如，通信关系可能仅仅参考抽象端点定义。

Sdm文档包含描述文档、文档中定义的管理器、输入参考其他文档和定义组重要语句的声明。

图8示出了文档实例。

所有的sdm定义均从一般基本定义中得出，可能包含如图9所示的成员。定义与成员之间的关系可以比下图所示出的更复杂。

参考图10所示，根据其参考的定义的种类来对成员进行划分。

参考设定定义来设定声明。如图11所示设定值和值列表提供用于设置的值。

2．1 SDM应用程序的生命周期

根据某些实施例的SDM应用程序的生命周期实例如图12所示。

将该应用程序设计并执行于visual studio环境中（块1202）。开发者执行组件并将其与混合组件相结合。该应用程序通过SDM文件描述。为了验证，该应用程序将被部署在特定数据中心内，开发者将应用程序绑定到同样在SDM文件中描述的数据中心表示上（块1204）。该表示将包括用于应用程序组件宿主和对应

用程序构造进行约束的定义。如果绑定失败，则开发者可修订他们的应用程序设计。

一旦开发者满意他们的应用程序，他们就可以设计并公布应用程序，因而现在的应用程序有一个较强的名字和版本与之相关联（块1206）。应用程序公布的形式称为软件分配单元（SDU）。操作者从开发者处取得SDU并将应用程序装入SDM运行时（块1208）。在装入应用程序的过程中，操作者选择他们想要绑定到应用程序上的数据中心的模型。当操作者选择部署一个应用程序时，他们向应用程序提供部署时间参数并确定应用程序的规模（块1210）。这些通过使用变更请求来完成。

一旦应用程序展开，操作者可以与运行时交互以确定应用程序的结构和应用程序每部分的设定（块1212）。运行时也可以校验应用程序的实际结构是否与记录于运行时中的期望结构相匹配。操作者可以通过服从变更请求来移除一个已部署的应用程序（块1214）。操作者也可以像移除一个服务包那样重新运行针对正在运行的应用程序的个别变更。在块1216，正在运行的应用程序的结构可以像对网络前端那样通过增加或移除已部署的应用程序来改变。应用程序也可以通过安装一个或多个应用程序组件的较新版本来升级。

2．2 抽象对象和关系定义

抽象对象定义定义了我们所需要的构件块，以便在设计时检查应用程序结构，并在运行时部署和管理应用程序。这些构件块表示存在于模型系统中的视图。例如，我们利用抽象对象定义来建立文件和目录的模型，在网页服务器或数据库中的结构在SQL server内部。

我们使用抽象关系定义来建立可在抽象对象定义之间发生的交互的模型。关系是二进制且定向的，其识别定义实例的对象定义，该实例参与关系的表示。关系提供了将对象约束在一起的途径，由此我们可以建立对象之间的包含、结构和通信链接。

然后约束被对象使用以约束关系，他们参与并通过关系来约束可被链接的对象。这些约束既把定义作为目标也把关系中参与者的设定作为目标。这就允许约束将关系中的参与者限定成实例，所述实例源自于特殊定义，并且允许约束要求该实例具有落入特定范围内的设定值。

我们将对象定义划分为三类：系统、终端和资源。

抽象系统定义用于描述应用程序的自包含独立可部署部分。这些定义表示通过被恰当地定义了的各通信信道交互的应用程序的各部分，该通信信道可以横穿过程并形成边界。

抽象终端定义用于描述系统可示出的通信终端。这些用于建立系统可知道的通信的所有形式的模型，以在设计时检验系统连通性并在运行时启动连接。

抽象资源定义描述包含于系统中的行为。资源定义可能与其他资源定义有较强的依赖关系。这些依赖关系可通过无文档通信机制包括需要特殊安装顺序和开始运行时交互。

所有的抽象对象定义共享示出设置的能力。这些设定是简单的命名值对，其使用XML模式来定义设置的类型。设置可以是动态或静态的，如果其静态的则仅可以在部署过程中进行设定，如果其是动态的则他们可以在部署后改变。在SDM运行时，对运行系统应用设定值负责的代码被寄宿在SDM运行时内。

SDM支持经由抽象对象定义的继承。衍生定义能够扩展由其母体陈述的性能并且能够为它的母体性能设定数值。衍生定义能够参与将它的母体视为参与者的任何一种关系。

关系定义分为五类：通信、包含、代理、寄宿和参考。

通信关系用于捕获在抽象终端定义之间潜在的通信交互作用。通信关系的存在表明陈述验证定义的终端组件能够相互通信。链接的实际建立受终端上的约束和终端的示出。

包含关系描述了抽象定义包含其它抽象定义元件的能力。更具体的，两个抽象定义A和B之间的包含关系允许执行A的具体对象定义去包含执行B的具体对象定义的成员。

我们使用约束来为开发者建立应用程序时所产生的固有嵌套结构建模。通过包含元件对象，母体能够控制生命周期和所包含对象的可见性。在运行时间空间内所有对象实例作为其它对象实例元件存在，形成完整地相连接的实例组。从而该组包含关系描述了在实例空间内产生的所允许的包含模式。

代理关系用于可选地陈述所包含元件；特别地，我们使用代理从组件定义内陈述终端元件。通过从分系统代理终端，外部系统陈述出使用特定协议进行通信的能力而不陈述出协议后执行装置。

寄宿和参考关系代表两种形式的依赖关系。寄宿关系描述了在生成具体对

象实例前应当存在的抽象定义之间的基本依赖性。每个实例正好在一个寄宿关系内应当作为客户参与，从而致使寄宿关系在实例空间内也形成完整的连接树。参考关系捕获用于参数流程和结构排序的附加依赖性。

2.3具体对象和关系定义

我们从抽象对象定义中构造具体对象定义以及从抽象关系定义中构造具体关系定义。

抽象对象定义和抽象关系定义的组合限定了用于对目标系统建模的模式。具体对象定义的作用是基于一个或多个抽象定义使用抽象定义空间的子集来生成可重复使用的结构。作为一个简单的类比，抽象定义空间能够比作为数据库的模式；那么具体对象定义将代表在数据库中用于一组行列的可再使用模板。当具体对象实例生成时，仅仅在数据库中生成这些行列。为了执行设计时间验证，我们可以相对于抽象定义空间来验证具体对象定义，所采用的方式与我们相对模式约束（例如外来密钥等等）在数据库中验证行列的方式相同。

每个具体对象定义提供用于特定抽象对象定义的执行程序。该执行程序包括设定模式扩展，设定数值，对象成员、关系成员、约束成员和流动成员的说明。具体对象的行为跟着抽象对象的定义：抽象系统定义变为具体系统定义，抽象终端定义变为具体终端定义，以及抽象资源定义变为具体资源定义。

每个具体关系定义提供用于特定抽象关系定义的执行程序。该执行程序包括设定说明和设定值、相同关系分类（寄宿、包含、通信等）的嵌套成员、以及对能够参与关系的类型的约束。

具体寄宿关系用来限定从一个具体对象的成员到另一个具体对象的映射。例如，具体寄宿关系可以用于确认在web应用和IIS宿主之间的绑定，其中具体寄宿关系将部署到该IIS宿主上。对于给定类型可以存在一个以上的具体寄宿关系，允许开发者为特定拓扑限定不同的部署结构。

2.4成员（MEMBERS）

具体类型可以说明其它具体或抽象对象的成员——我们称这些为对象成员。这些成员随后参考限定对象成员之间关系的关系成员。

对象成员用来创建特定对象定义的实例。设定流动能够用来为对象提供数值。当声明对象成员时，用户可以决定对象成员是在创建外部系统的同时创建（值语义）还是通过在某一后面时间发生的显示新操作创建（参考语义）。

关系成员限定当对象成员被创建时对象成员将参与的关系。如果对象成员被它的母体包含，那么约束关系成员将在类型成员和外部类型之间进行说明。如果对象成员被代理，那么代理关系成员将在对象成员和嵌套对象成员之间进行限定。通信关系成员可以在对象成员上的终端之间声明，依赖关系成员（参考和寄宿）可以在对象成员或嵌套对象成员之间声明。

关系约束用于限定特定对象期望参与的关系组。它们确认在特定关系和在所述关系另一端的参与者上的约束。

2.5实例空间(IINSTANCE SPACE)

存储在SDM运行时内的实例空间反映了模型系统的当前状态。该运行时包含已经创建的实例和这些实例之间关系的完整记录。每个实例具有相关联的版本历史，其中每个版本链接到变更请求。

创建新实例的过程是由变更请求来启动的。对于类型和与现存实例的特定成员相关联的关系，该变更实例限定一组产生、更新和删除请求；根是一种具体的情况。

变更请求由运行时扩展、相对所有约束进行验证并且构建。该扩展过程确认对象和作为包含对象的构建请求的一部分而隐式构建的关系实例，随后设置流在所有关系上经评估。该验证步骤检查所有所需的关系存在以及这些关系满足所有的约束。最后，该构建过程确定部署、更新或删除每个实例的合适排序并且随后按照正确顺序将每个实例送到实例管理器去执行合适的动作。

2.6分层(LAYERING)

SDM模型的目标是允许将应用程序开发者、软件基础结构设计者和数据中心体系结构之间所关注的事物分开。这些组中每一个重点集中在特定服务并且具有不同的依赖性组。

例如，开发者主要关心结构与其依赖的宿主之间的连接性，所述宿主比如像SQL、IIS和CLR。宿主结构设计者关心网络拓扑和OS结构，而开发网络拓扑的体系结构、OS结构和存储映射需要知道在数据中心中存在的硬件。

为了支持这种对关心事物的分离，SDM陈述了分层的概念。分层是使用寄宿关系来将应用程序绑定到它所依赖的服务上而不说明那些服务作为所述应用程序约束结构的一部分。

我们把四层确认成SDM模型的一部分……

应用程序层

● 该应用程序层在约束的上下文关系内支持应用程序的结构。该上下文关系由在宿主层确认的宿主结构来进行限定。

● 在应用程序层内系统定义的实例包括web服务、数据库和Biztalk进度表。

宿主层

● 在软件组件外构建数据中心。构造在组件之间的连接。一些这些组件用作应用程序层的宿主。

● 该层内系统定义的实例—IIS、SQL、AD、EXCHANGE、DNS和Biztalk。

网络/OS/存储层

● 构建数据中心网络和平台。构造网络安全模型和操作系统平台结构。为操作系统结构添加存储器。

● 该层内系统定义的实例—VLAN、Windows、Filter、Storage。

硬件层

该硬件层确认存在于数据中心的机器类型和在这些机器之间存在的物理连接。

图13显示了从层4 web应用程序到层3 web服务器宿主的示例映射。每层外部的方框代表系统、边界的方框代表终端、内部的方框代表资源。我们通过寄宿关系将这些元素中的每一个映射到处于层下的宿主上。

为了满足系统所需的关系，我们将该系统绑定到具有匹配性能的宿主系统上。我们称该过程为布置。在设计时间，我们构建代表可能布置的具体寄宿关系。在部署时，我们例示具体寄宿关系实例来将客户系统实例绑定到宿主系统实例上。

2．7 模型评估(MODEL EVALUATION)

与SDM模型相联系的是用于管理分布式系统变更的明确定义的过程。

每个变更通过说明性变更请求进行驱动，该说明性变更请求在请求内的动作相对目标系统进行分布和执行之前通过几个处理步骤。

3 实现细节

3．1 命名（NAMING）

在SDM中的许多地方我们都需要一个较强的命名系统以定义对象。以下的命名系统允许一个类型的创建者标记该定义，用这样一种方法定义的用户可以

确定其与开发者原始公布的定义是一致的。

以下的表头是SDM命名空间（namespace）标识符的一个实例：

```
<sdm name="FileSystem"
     version="0.1.0.0"
     publicKeyToken="AAAABBBBCCCCDDDD"
     culture="neutral"
     platform="neutral"
     publicKey="aLongKey"
     signature="TheHashOfTheFileContents" >
</sdm>
```

为了参考其他命名空间中的类型需要引入如下命名空间：

```
<import alias="FileSystem" name="FileSystem" version="0.1.0.0" publicKeyToken="AAAABBBBCCCCDDDD"/>
```

然后你可以使用如下别名来指代命名空间中的类型：

```
FileSystem:file
```

3．1．1 身份（Identity）

SDM名称在命名空间中定义并通过命名空间来限定范围。通过名称（name）、版本（version）、语言（language）和公共密钥标记（public key token）来定义命名空间，且它们包含在一个单独的文件中。

恒等式的基本组成包括名称、版本号、语言和公共密钥标记。

```
<xs:attributeGroup name="Identity">
    <xs:attribute name="name" type="simpleName" use="required"/>
    <xs:attribute name="version" type="fourPartVersionType" use="required"/>
    <xs:attribute name="publicKeyToken" type="publicKeyTokenType" use="optional"/>
    <xs:attribute name="culture" type="xs:string" use="optional"/>
    <xs:attribute name="platform" type="xs:string" use="optional"/>
</xs:attributeGroup>
```

Attribute / element	Description
name	The name of the sdm file is a friendly name that a developer can use to reference the contents of the file. The name in combination with the public key token provides a strong name for the file.
version	Version is used to identify the version of the contents of the file. All elements of the file adopt the same version number
publicKeyToken	Public key token is a short name for the public key associated with the file.
culture	The culture of the binaries. Defaults to neutral
platform	The supported platform for the binaries.

基本身份可用于参考一个已存在的身份，或者与签名（signature）和公共密钥一起建立一个新的健壮的身份。使用私有密钥（private key）对文档签名，从而

允许文档的用户通过公有密钥检验其内容。

```
<xs:attributeGroup name="namespaceIdentity">
    <xs:attributeGroup ref="Identity"/>
    <xs:attribute name="signature" type="xs:string" use="optional"/>
    <xs:attribute name="publicKey" type="xs:string" use="optional"/>
</xs:attributeGroup>
```

Attribute / element	Description
signature	The signed hash of the contained sdm type definitions
publicKey	The public key that can be used to check the signature of the file.

公共密钥标记是16字符的十六进制字符串，其标识公共/私有密钥对的公共部分。这不是公共密钥，仅仅是公共密钥的64位无用信息。

```
<xs:simpleType name="publicKeyTokenType">
    <xs:annotation>
        <xs:documentation>Public Key Token: 16 hex digits in size</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
        <xs:pattern value="([0-9]|[a-f]|[A-F]){16}"/>
    </xs:restriction>
</xs:simpleType>
```

3．1．2 版本（Version）

一个文件的版本通过形如N.N.N.N的四部分数字定义，其中0<=N<=65535。根据惯例，该数字参照Major .Minor .Build .Revision的形式。

```
<xs:simpleType name="fourPartVersionType">
    <xs:restriction base="xs:string">
        <xs:pattern value="([0-9]{1,4}|[0-5][0-9]{4}|64[0-9]{3}|655[0-2][0-9]|6553[0-5])(\.([0-9]{1,4}|[0-5][0-9]{4}|64[0-9]{3}|655[0-2][0-9]|6553[0-5])){3}"/>
    </xs:restriction>
</xs:simpleType>
```

3.1.3 简名（Simple names）

简名由阿拉伯数字字符和有限的标点符号构成。该名称必须以非数字字符开始。

```
<xs:simpleType name="simpleName">
    <xs:annotation>
        <xs:documentation>name of a type or member</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
        <xs:pattern value="[a-z,A-Z]{1}([0-9,a-z,A-Z,_])*"/>
    </xs:restriction>
</xs:simpleType>
```

我们计划使符合C#对应标识符的定义；有关部分（2.4.2）已在下面插入。说明可在以下网址找到：

http://devdiv/SpecTool/Documents/Whidbey/VCSharp/Formal%20Language%20Specification/CSharp%20Language%20Specification.doc

注意在SDM模型中并不支持“@”前缀名。

The rules for identifiers given in this section correspond exactly to those recommended by the Unicode Standard Annex 15, except that underscore is allowed as an initial character (as is traditional in the C programming language), Unicode escape sequences are permitted in identifiers, and the "@" character is allowed as a prefix to enable keywords to be used as identifiers.

identifier:
available-identifier
@ ***identifier-or-keyword***

available-identifier:
An ***identifier-or-keyword*** **that is not a** ***keyword***

identifier-or-keyword:
identifier-start-character ***identifier-part-characters***$_{opt}$

identifier-start-character:
letter-character
_ (the underscore character U+005F)

identifier-part-characters:
identifier-part-character

identifier-part-characters *identifier-part-character*

identifier-part-character:
letter-character
decimal-digit-character
connecting-character
combining-character
formatting-character

letter-character:
A Unicode character of classes Lu, Ll, Lt, Lm, Lo, or Nl
A *unicode-escape-sequence* representing a character of classes Lu, Ll, Lt, Lm, Lo, or Nl

combining-character:
A Unicode character of classes Mn or Mc
A *unicode-escape-sequence* representing a character of classes Mn or Mc

decimal-digit-character:
A Unicode character of the class Nd
A *unicode-escape-sequence* representing a character of the class Nd

connecting-character:
A Unicode character of the class Pc
A *unicode-escape-sequence* representing a character of the class Pc

formatting-character:
A Unicode character of the class Cf
A *unicode-escape-sequence* representing a character of the class Cf

For information on the Unicode character classes mentioned above, see *The Unicode Standard, Version 3.0*, section 4.5.

Examples of valid identifiers include "`identifier1`", "`_identifier2`", and "`@if`". An identifier in a conforming program should be in the canonical format defined by Unicode Normalization Form C, as defined by Unicode Standard Annex 15. The behavior when encountering an identifier not in Normalization Form C is implementation-defined; however, a diagnostic is not required.

The prefix "@" enables the use of keywords as identifiers, which is useful when interfacing with other programming languages. The character @ is not actually part of the identifier, so the identifier might be seen in other languages as a normal identifier, without the prefix. An identifier with an @ prefix is called a *verbatim identifier*. Use of the @ prefix for identifiers that are not keywords is permitted, but strongly discouraged as a matter of style.

The example:

```
class @class
{
public static void @static(bool @bool) {
        if (@bool)
                System.Console.WriteLine("true");
        else
                System.Console.WriteLine("false");
}
}
class Class1
```

```
{
        static void M() {
                cl\u0061ss.st\u0061tic(true);
        }
}
```

defines a class named "class" with a static method named "static" that takes a parameter named "bool". Note that since Unicode escapes are not permitted in keywords, the token "cl\u0061ss" is an identifier, and is the same identifier as "@class".

Two identifiers are considered the same if they are identical after the following transformations are applied, in order:

- The prefix "@", if used, is removed.
- Each *unicode-escape-sequence* is transformed into its corresponding Unicode character.
- Any *formatting-characters* are removed.

Identifiers containing two consecutive underscore characters (U+005F) are reserved for use by the implementation. For example, an implementation might provide extended keywords that begin with two underscores.

3.1.4 保留名（Reserved names）

以下是保留名列表，当为SDM模型中的对象建立名称时为防止用户使用。在一定环境中，将保留某些名称。

Context	Name
Abstract and concrete definitions	**this**
Abstract and concrete hosting Relationship definitions	**guest, host**
Abstract and concrete containment relationship definitions	**parent, member**
Abstract and concrete communication relationship definitions	**client, server**
Abstract and concrete reference relationship definitions	**source, dependent**
Abstract and concrete delegation relationships definitions	**proxy, delegate**

因为与CLR结合，所以将这些名称保留。

C# keywords				
AddHandler	AddressOf	Alias	And	Ansi
As	Assembly	Auto	Base	Boolean
ByRef	Byte	ByVal	Call	Case
Catch	CBool	CByte	CChar	CDate
CDec	CDbl	Char	CInt	Class
CLng	CObj	Const	CShort	CSng
CStr	CType	Date	Decimal	Declare
Default	Delegate	Dim	Do	Double
Each	Else	ElseIf	End	Enum
Erase	Error	Event	Exit	ExternalSource
False	Finalize	Finally	Float	For
Friend	Function	Get	GetType	Goto
Handles	If	Implements	Imports	In
Inherits	Integer	Interface	Is	Let
Lib	Like	Long	Loop	Me
Mod	Module	MustInherit	MustOverride	MyBase
MyClass	Namespace	New	Next	Not
Nothing	NotInheritable	NotOverridable	Object	On
Option	Optional	Or	Overloads	Overridable
Overrides	ParamArray	Preserve	Private	Property
Protected	Public	RaiseEvent	ReadOnly	ReDim
Region	REM	RemoveHandler	Resume	Return
Select	Set	Shadows	Shared	Short
Single	Static	Step	Stop	String
Structure	Sub	SyncLock	Then	Throw
To	True	Try	TypeOf	Unicode
Until	volatile	When	While	With
WithEvents	WriteOnly	Xor	eval	extends
instanceof	package	var		

3.1.5 参考其他命名空间(References to other namespaces)

我们允许命名空间通过将其引入当前的命名空间并将该命名空间与别名相关联来参考其他的命名空间。该引入的命名空间通过名称、版本和公共密匙标记引用。版本将在3.16部分描述。

```
<xs:complexType name="import">
    <xs:attribute name="alias" type="simpleName" use="required"/>
    <xs:attributeGroup ref="identity"/>
</xs:complexType>
```

Attribute / element	Description
alias	The alias used to reference the external sdm file within the scope of the current sdm file.

3.1.6 合格路径（Qualified Paths）

合格路径既是相关定义的名称也是在当前命名空间或别名命名空间中定义的管理者。

```
[<alias> :] <simpleName> (. <simpleName>)*
```

在引入阶段定义别名。下面的简名标识类型甚至路径、嵌套类型。

```
<xs:simpleType name="qualifiedName">
    <xs:restriction base="xs:string">
        <xs:pattern value="[a-z,A-Z]{1}([0-9,a-z,A-Z,_])*(:[a-z,A-Z]{1}([0-9,a-z,A-Z,_])*)?(\.[a-z,A-Z]{1}([0-9,a-z,A-
Z,_])*)*"/>
    </xs:restriction>
</xs:simpleType>
```

3.1.7 定义路径和成员路径（Definition and Member Paths）

路径是标识成员或设置的一个名称序列。路径应该以众所周知的名称或成员名来开始，该名称或成员名通过对象或与路径相关联的关系来定义。

```
<simpleName>(.<simpleName>)*

<xs:simpleType name="path">
    <xs:restriction base="xs:string">
        <xs:pattern value="[a-z,A-Z]{1}([0-9,a-z,A-Z,_])*(\.[a-z,A-Z]{1}([0-9,a-z,A-Z,_])*)*"/>
    </xs:restriction>
</xs:simpleType>
```

3.1.8 实例路径（Instance Paths）

实例空间中的路径基于xpaths，此处xpath中的元素名称与符合设定的xpath中的成员名称和属性相符合。

3.1.9 名称解析（Name Resolution）

未以别名开始的名称没有充分资格。这意味着名称所被估价的范围可以改变产生的绑定。一个实例就是嵌套定义。当分析嵌套定义的名称时，在本地范围内的定义将隐藏到宽范围中的定义。

3.2 设置（SETTINGS）

所有的定义都可以陈述设置声明。当从抽象定义中建立具体定义时，或从其他定义的成员中参考定义时，这些设置用于描述提供的值。

为了定义一个设置，首先需要定义使用xsd的设定。

```
<xs:schema>
    <xs:simpleType name="registryValueType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="binary"/>
            <xs:enumeration value="integer"/>
            <xs:enumeration value="long"/>
            <xs:enumeration value="expandString"/>
            <xs:enumeration value="multiString"/>
            <xs:enumeration value="string"/>
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

然后可以说明一个设定，其使用定义并包括一组用于定义设定行为的属性。

```
<settingDeclaration name="valueType" type="registryValueType" access="readwrite" dynamic="false" required="true"/>
```

一旦具有了设置声明，就可以提供设置的值。

```
<settingValue name="valueType" fixed="true">long</settingValue>
```

3.2.1 设置定义（Setting Definitions）

我们使用XSD模式来限定由设置声明所使用的设置定义。我们支持使用来自模式的简单和复杂类型，尽管其它模式元素可以存在来支持那些类型的定义。

设置定义部分应当包含包括有命名空间声明和命名空间输入的完整的xml模式。我们将检测在xsd模式内的输入除了xsd模式命名空间外与sdm文件内的输入相匹配。这意味着所有参考类型应当在另一个sdm文件内进行限定；该模式不能参考在其它任意的xsd文件内限定的类型。

```
<xs:complexType name="settingDefinitions">
    <xs:sequence>
        <xs:element ref="xs:schema"/>
    </xs:sequence>
    <xs:attribute name="manager" type="qualifiedName" use="optional"/>

    <xs:attribute name="clrNamespace" type="xs:string" use="optional"/>
</xs:complexType>
```

Attribute / element	Description
xs:schema	A schema in the http://www.w3.org/2001/XMLSchma namespace.
manager	The clr assembly that contains helper classes for this schema
clrNamespace	The clr namespace in which these classes are defined. All setting type will resolve to a clr based on their mapping through CLR serialization.

设定可从三个独立的命名空间加以解析：

a） SDM命名空间一当涉及系统、资源、终端、关系、包含或流类型中

的设定类型时。

b） CLR命名空间－当涉及使用clr强类型类的设定时和当设定类型在其他设定类型中建立时。

c） XSD命名空间－当设定类型使用其他设定类型建立时。

为了使其工作，我们需要对声明设定进行多种约束：

a） 每个CLR、SDM和XSD命名空间中的所有设定必须在相同的分组中。也就是说，如果两个设定都在一个命名空间中，则在所有的三个命名空间中它们都必须在一起。

b） XSD模式定义中的引入命名空间应该与SDM文件中的引入命名空间及关联帮助组合体中的引入命名空间相匹配。

c） 除XSD之外，XSD模式中所有的引入域名空间均应在SDM文件中定义。

来自引入SDM文档的XSD类型可以使用Qname来访问：

```
<alias>:<type-name>
```

因此，举例来说，如果Foo.sdm引入Bar.sdm，则Bar.sdm的设定类型可能参考Foo.sdm的SettingType元素，正如下面这个示例举例说明的那样。

```
<!--Foo.sdm-->
...
<import alias="bar" location="Bar.sdm".../>

<settingTypes>
        ...
        <xs:simpleType name="D">
                <xs:restriction base="bar:B".../>
        </xs:simpleType>
        ...
</settingTypes>

<!--Bar.sdm-->
...
<settingTypes>
        ...
        <xs:simpleType name="B">
                <xs:restriction base="xs:string".../>
        </xs:simpleType>
        ...
</settingTypes>
```

3.2.2 内嵌简单数据类型（Built in simple data types）

SDM支持有限的一组嵌入数据类型，该嵌入数据类型为XSD和C＃域名空间的交集。这些类型由SDM运行时天生支持并在下表内进行限定。除了这些类型外，用户自由构造和使用在xsd和cls类型之间的他们自己的映射。

Type	Description	XSD Type	C# Type
string	A string is a sequence of Unicode characters	string	string
integer	64-bit signed integral type	long	long
float	Single-precision floating point type	float	float
double	Double-precision floating point type	double	double
boolean	a boolean value is either true or false	boolean	bool
any	The base type of all other types	anyType	object
Date	A simple date	date	dateTime

这些类型在C＃和xsd类型空间内可以流动的这些类型的相同起源。例如，字符串数值可以流动到在字符串上限定了约束的xsd类型，并且任何值都可以流动到接受类型＝“any”的设置。

3.2.2.1 XSD内嵌类型

图14示出了示例嵌入数据类型分层结构。

3.2.2.2 C＃数据类型

Type	Description	Example
object	The ultimate base type of all other types	`object o = null;`
string	String type; a string is a sequence of Unicode characters	`string s = "hello";`
sbyte	8-bit signed integral type	`sbyte val = 12;`
short	16-bit signed integral type	`short val = 12;`
int	32-bit signed integral type	`int val = 12;`
long	64-bit signed integral type	`long val1 = 12;` `long val2 = 34L;`
byte	8-bit unsigned integral type	`byte val1 = 12;`
ushort	16-bit unsigned integral type	`ushort val1 = 12;`
uint	32-bit unsigned integral type	`uint val1 = 12;` `uint val2 = 34U;`

ulong	64-bit unsigned integral type	`ulong val1 = 12;`
		`ulong val2 = 34U;`
		`ulong val3 = 56L;`
		`ulong val4 = 78UL;`

float	Single-precision floating point type	`float val = 1.23F;`
double	Double-precision floating point type	`double val1 = 1.23;` `double val2 = 4.56D;`
bool	Boolean type; a bool value is either true or false	`bool val1 = true;` `bool val2 = false;`
char	Character type; a char value is a Unicode character	`char val = 'h';`
decimal	Precise decimal type with 28 significant digits	`decimal val = 1.23M;`

3.2.2.3 支持的转换

以下是在xsd类型和cls类型之间存在的转换。

XML Schema (XSD) type	.NET Framework type
anyURI	System.Uri
base64Binary	System.Byte[]
Boolean	System.Boolean
Byte	System.SByte

Date	System.DateTime
dateTime	System.DateTime
decimal	System.Decimal
Double	System.Double
duration	System.TimeSpan
ENTITIES	System.String[]
ENTITY	System.String
Float	System.Single
gDay	System.DateTime
gMonthDay	System.DateTime
gYear	System.DateTime
gYearMonth	System.DateTime
hexBinary	System.Byte[]
ID	System.String
IDREF	System.String
IDREFS	System.String[]
int	System.Int32
integer	System.Decimal
language	System.String
long	System.Int64
month	System.DateTime
Name	System.String
NCName	System.String
negativeInteger	System.Decimal
NMTOKEN	System.String
NMTOKENS	System.String[]
nonNegativeInteger	System.Decimal
nonPositiveInteger	System.Decimal
normalizedString	System.String
NOTATION	System.String
positiveInteger	System.Decimal
QName	System.Xml.XmlQualifiedName
short	System.Int16
string	System.String
time	System.DateTime
timePeriod	System.DateTime
token	System.String
unsignedByte	System.Byte
unsignedInt	System.UInt32
unsignedLong	System.UInt64
unsignedShort	System.UInt16

3.2.3 设置声明(Setting declaration)

设置声明部分使用从前一部分来的设置定义以建立命名的设置。属性用于提供每个设置更进一步的信息。

```
<xs:complexType name="SettingDeclaration">
    <xs:complexContent>
        <xs:extension base="Member">
            <xs:attribute name="List" type="xs:boolean"/>
            <xs:attributeGroup ref="settingsAttributes"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
settingsAttributes	The set of attributes that can be applied to an individual setting declaration
list	This attribute is used to indicate that this setting is a list of values rather than a single value.

3.2.4 列表支持(List support)

为了支持多值设定操作，我们支持设定值的简单列表。作为设定说明的列表是相同定义值的序列。列表可以流向其他的列表，该列表可以流动到它们可以重置或合并的其他列表。当给列表添加数值时我们不支持复制部分因为通过使用设定流动这可以更灵活地完成，并且我们不保证任何形式的排序。

列表说明包括将属性列表设为“真（true）”：

```
<settingDeclaration name="roles" type="xs:string" list="true"/>
```

然后利用settingValueList来提供值。当提供列表时，用户可以指定是否重置前一值或与前一值合并。

```
<settingValueList name="roles" fix="true" replace="true">
    <value>staticContent</value>
    <value>aspPages</value>
</settingValueList>
```

SDM支持列表值的简单操作。当来自流成员的路径以设定说明为目标时，结果行为依赖于路径任一终点的定义。

Source	Destination	Result
Element	List	replace = false – element is added to the list
		replace = true – list with single element

List	List	replace = false – source and destination lists are merged replace = true – source list
List	Element	The sdm cannot resolve which element to select from the list so this combination is **not supported**.

3.2.5 设定属性(Setting Attributes)

设定属性由运行时使用，用于描述特殊设定的行为。

```
<xs:attributeGroup name="settingsAttributes">
    <xs:attribute name="access">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="readwrite"/>
                <xs:enumeration value="readonly"/>
                <xs:enumeration value="writeonly"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="secure" type="xs:boolean" />
    <xs:attribute name="deploymentTime" type="xs:boolean"/>
    <xs:attribute name="required" type="xs:boolean"/>
    <xs:attribute name="dynamic" type="xs:boolean"/>
    <xs:attribute name="keyValue" type="xs:boolean"/>
    <xs:attribute name="nillable" type="xs:boolean" />
</xs:attributeGroup>
```

<table>
<tr><th>Attribute Name</th><th>Description</th><th>Default</th></tr>
<tr><td>access</td><td>The access attribute of a setting specifies whether reading and writing the setting's value is permitted; providing SDM runtime access control and display/editing rules to designers.

<table>
<tr><th>Attribute Value</th><th>Meaning in SDM Runtime</th><th>Display/Editing Rules</th></tr>
<tr><td>readwrite</td><td>Indicates that a setting value can be read and written.</td><td>Value is displayed and can be edited.</td></tr>
<tr><td>readonly</td><td>Indicates that a setting value can be read, but not written. This value cannot be a target for flow.

In general a readonly setting will be one that is calculated or provided by the real world instance.

Example: A value that represents the number of connections on a server</td><td>Value is displayed, but cannot be edited.</td></tr>
<tr><td>writeonly</td><td>Indicates that a setting value can be written, but not read. This value cannot be a source for flow.

Example: a password for a service account</td><td>Value is masked, but can be edited.</td></tr>
</table>
</td><td>readWrite</td></tr>
<tr><td>deploymentTime</td><td>A value that can only be provided as part of the deployment process for an instance. Design time constraints cannot be evaluated against this value. A definition or member cannot supply a fixed value for this setting.</td><td>False</td></tr>
<tr><td>required</td><td>If required is true, a value should be provided for this setting before an instance is deployed. This setting cannot be readonly.</td><td>False</td></tr>
</table>

dynamic	If **dynamic** is true, the instance manager supports changes to this value after an instance has been deployed.	True
keyValue	**Keyvalue** is used to Indicate that a setting is unique within its hosting scope. The instance manager will use this setting in paths to identity object instances and to detect collisions with existing instances.	False
Secure	The **secure** attribute of a setting specifies whether the value of a setting should be encrypted (**true** or **false**) when stored to an SDM document. Also indicates whether tools should log this value during manipulations such as installs.	False
nillable	The **nillable** attribute of a setting indicates whether a setting value is valid if it carries the namespace-qualified attribute **xsi:nil** from namespace http://www.w3.org/2001/XMLSchema-instance.	false

3.2.6 设置值（Setting Values）

依赖于设置作为单个值或列表，用于设定的值可以使用设置值元素或设置值列表元素来提供。

3.2.6.1 设置值

设置值用于提供特殊设置说明的值。该值应当与声明相关联的定义相匹配。如果该值被说明为固定的，则该提供的值将用于所有衍生的定义或参考成员，这取决于值被固定所在的点。一旦值被固定，它不可以再被覆盖。

```
<xs:complexType name="settingValue">
    <xs:complexContent>
        <xs:restriction base="xs:anyType">
            <xs:attribute name="name" type="simpleName" use="required"/>
            <xs:attribute name="fixed" type="xs:boolean" use="optional" default="false"/>
            <xs:attribute ref="xsi:nil" use="optional" default="false"/>
        </xs:restriction>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
name	The name of the setting declaration that this value will apply to.
fixed	The **fixed** attribute controls whether the value provided can subsequently be overridden by new values A **fixed** value of **true** indicates that the value provided for a setting

	cannot be overridden If the value of **fixed** is **true** on a concrete definition member's setting value, it is fixed in all deployments of that member. Otherwise, if the value of fixed is false it is override-able in each deployment of that member. If the value of **fixed** is **true** on a concrete definition's setting value, it is fixed for all members (i.e., uses) of that concrete definition. Otherwise, if the value of **fixed** is **false**, it may vary with each member (i.e., use) of that concrete definition. If the value of **fixed** is **true** on an abstract definition's setting value, it is fixed for concrete definitions that implement or abstract objects that extend that abstract definition. Otherwise, if the value of **fixed** is **false**, it may be overridden by a concrete definition, in a derived abstract definition or in a member declaration.
Nil	Finally, a setting value is considered valid without content if it has the attribute **xsi:nil** with the value **true**. An element so labeled should be empty, but can carry attributes if permitted by the setting definition .

3.2.6.2 设置值列表

设置值用于给作为列表而声明的设定提供一个或多个值。当声明这些值时，用户能够决定合并先前的值或者重写全部的先前值。

```
<xs:complexType name="settingValueList">
    <xs:sequence>
        <xs:element name="value" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
                <xs:complexContent>
                    <xs:restriction base="xs:anyType">
                    </xs:restriction>
                </xs:complexContent>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
    <xs:attribute name="name" type="simpleName" use="required"/>
    <xs:attribute name="fixed" type="xs:boolean" use="optional" default="false"/>
    <xs:attribute name="replace" type="xs:boolean" use="optional" default="false"/>
</xs:complexType>
```

Attribute / element	Description
name	The name of the setting declaration that this value will apply to.

fixed	The **fixed** attribute controls whether the value provided can subsequently be overridden by new values A **fixed** value of **true** indicates that the value provided for a setting cannot be overridden If the value of **fixed** is **true** on a concrete definition member's setting value, it is fixed in all deployments of that member. Otherwise, if the value of fixed is false it is override-able in each deployment of that member. If the value of **fixed** is **true** on a concrete definition's setting value, it is fixed for all members (i.e., uses) of that concrete definition. Otherwise, if the value of **fixed** is **false**, it may vary with each member (i.e., use) of that concrete definition. If the value of **fixed** is **true** on an abstract definition's setting value, it is fixed for concrete definitions that implement or abstract objects that extend that abstract definition. Otherwise, if the value of **fixed** is **false**, it may be overridden by a concrete definition, in a derived abstract definition or in a member declaration.
replace	The replace attribute is used to indicate whether the new value for a setting should replace or merge with any previous non-fixed values for a setting. If replace is true then all previous values will be ignored. If replace is false, then the new and the previous values will be combined into a single list. Duplicates will NOT be detected during the merge process.

3.2.7 设置继承（Settings Inheritance）

设置继承意味着衍生定义隐式包括所有来自基础定义的设定说明。设置继承的一些重要方面为：

设置继承是可传递的。如果C从B衍生而来并且B从A衍生而来那么C继承在B内以及在A内声明的设置。

衍生定义可以为它从它所扩展的基础定义继承的那些内容添加新的设置声明，但是它不能删除所继承设置的定义。

3．2．8 类型转换（Type Conversions）

我们支持在嵌入类型之间的无损转换。其它类型转换需要流动以便执行合适的转换。

3.3 属性（ATTRIBUTES）

许多SDM中的对象可以被定义属性来捕获中心行为被正交的行为。我们使

用一般属性模型定义如下：

3.4 定义和成员（DEFINITIONS AND MEMBERS）

3.4.1 定义（Definition）

定义是基础，对象、关系、约束和流动定义均由定义衍生而来。所有定义可包括设定模式和设计表面数据。每一个定义通过简名确定并参考管理器。该管理器负责提供对用于特定定义的SDM运行时的扩展支持。

设定模式限定了可以在该定义实例上找到的数值。DesignData元素用来包含在设计表面上显示和边界该定义的特定数据。

```
<xs:complexType name="Definition">
    <xs:sequence>
        <xs:element name="Description" type="Description" minOccurs="0"/>
        <xs:element name="DesignData" type="DesignData" minOccurs="0"/>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="SettingDeclaration" type="SettingDeclaration"/>
            <xs:element name="SettingValue" type="SettingValue"/>
            <xs:element name="SettingValueList" type="SettingValueList"/>
        </xs:choice>
    </xs:sequence>
    <xs:attribute name="Name" type="SimpleName" use="required"/>
    <xs:attribute name="Manager" type="QualifiedName" use="optional"/>
    <xs:attribute name="ClrClassName" type="xs:string" use="optional"/>
</xs:complexType>
```

Attribute / element	Description
SettingDeclaration	The declaration of a setting
SettingValue	A value for a setting on the defintion or its base definition. A value can be provided once for a setting declaration within an definition.
SettingValueList	A list of values for a writable list setting on the definition or its base definition.
DesignData	Design surface specific data
Name	A name for this definition that is unique within the scope of the containing sdm file.
Manager	A reference to the manager declaration for this definition. See section:3.10 for a definition of the manager.

ClrClassName	The name of the clr class that supports this definition in the runtime. The class should exist in the assembly identified by the manager. The manager attribute should be present if this attribute is present.
Description	A text description of the definition.

3.4.2 成员（Member）

成员用于确认在运行时存在的定义实例。所有成员在类型范围内通过可以为它们所参考的定义提供设定的唯一命名来进行确认，并且可以包含设计表面特定数据。

```
<xs:complexType name="Member">
    <xs:sequence>
        <xs:element name="Description" type="Description" minOccurs="0"/>
        <xs:element name="DesignData" type="DesignData" minOccurs="0"/>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="SettingValue" type="SettingValue"/>
            <xs:element name="SettingValueList" type="SettingValueList"/>
        </xs:choice>
    </xs:sequence>
    <xs:attribute name="Name" type="SimpleName" use="required"/>
    <xs:attribute name="Definition" type="QualifiedName" use="required"/>
</xs:complexType>
```

Attribute / element	Description
Description	A description of the members
DesignData	Design surface specific information about the member
SettingValue	Values for settings that correspond to writeable settings on the referenced type. If these values are marked as fixed then they should be used when an instance is created for the member, if they are not fixed, then the values can be overridden by deployment or flowed parameters.
SettingValueList	A list of values for a writable list setting on the referenced type.
Name	A unique name for the member within the scope of the containing type.
Definition	The name of the definition that this member references.

3．5 设置流（SETTINGS FLOW）

设置流用于在对象定义的成员之间和在关系中参与者之间传递参数。作为流的一部分，用户可以使用转换来组合或分离设置值并计算新的设置值。

所有设置流成员使用流定义来执行转换。流定义在SDM文件中说明。如下是url语法分析的流类型。

```
<FlowDefinition name="UrlToComponents">
    <SettingDeclaration name="url" type="url" access="writeonly"/>
    <SettingDeclaration name="protocol" type="xs:string" access="readonly"/>
    <SettingDeclaration name="server" type="xs:string" access="readonly"/>
    <SettingDeclaration name="path" type="xs:string" access="readonly"/>
    <SettingDeclaration name="file" type="xs:string" access="readonly"/>
</FlowDefinition>
```

然后，流动成员在对象或关系内被声明。流成员为流定义提供输入并将流

的输出引导入目标设置。

```
<Flow name="deconstructUrl"     type="UrlToComponents">
    <Input name="url" path="webservice.url"/>
    <Output name="server" path="webservice.server"/>
</Flow>
```

3.5.1 流定义(Flow definition)

我们使用流定义来限定我们期望在一组设定值中应用的特殊转换。流定义示出定义输入设置（只写设置）和输出设置（只读设置）的设置模式，用于设计表面特殊信息的DesignData部分，如输入接口，用于限定转换，并且描述用于在浏览sdm文件时使用。流定义在其所限定的命名空间内由命名确定。当该定义评估流时，该定义也确认支持运行时的管理器。

我们期望在需要直接转换的地方运行时包括几个标准流设置来简化流元素的构建。实例可以包括拷贝、合并和字符串替换。由于流定义可以被参数化我们也期望有基于结构参数来执行不同操作的一个或多个简单的转换。

```
<xs:complexType name="FlowDefinition">
    <xs:complexContent>
        <xs:extension base="Definition"/>
    </xs:complexContent>
</xs:complexType>
```

3．5．2 流成员（Flow member）

每个流元素确认一个或多个资源节点、一个或多个目标节点、一些静态设定和流定义。当该流被评估时，资源数据从资源节点收集，与来自流动元素的设定相结合，并传送到流动定义用于转换。该输出数据传送到目标节点。

```
<xs:complexType name="FlowMember">
    <xs:complexContent>
        <xs:extension base="Member">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="Input" type="SettingTarget"/>
                <xs:element name="Output" type="OutputPath"/>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
input	A list of paths to setting values that are used as input to the flow. Each input should identify a write only setting on the flow definition.
output	A list of paths to settings that will be set as a result of this flow. Each output should identify a read only setting on the flow definition.

3.5.3 设置目标（Setting target）

设置目标在成员或嵌套成员内确认到设定值的路径，该成员或嵌套成员与在限定流动的上下文内的公知命名相关。公知命名的实例包括在定义或参考声明内的this、在寄宿关系声明内的host和guest、或在约束说明内限定的target。设定目标也确认与流定义相关的设置，该相关流定义将用作源值或由路径确认的设定的目标设置。

```
<xs:complexType name="SettingTarget">
    <xs:attribute name="Name" type="SimpleName"/>
    <xs:attribute name="Path" type="Path"/>
</xs:complexType>
```

Attribute / element	Description
Name	The name of the setting on the flow or constraint definition that is the source or destination of the setting identified by the path.
Path	Path to the source or destination setting relative to the context in which the flow is defined. The path should identify a single setting – this implies that the maximum cardinality of all the members on the path should be one or zero.

输出路径对settingTarget的变量，该settingTarget支持用于固定和替换目标值的语义。

```
<xs:complexType name="OutputPath">
    <xs:complexContent>
        <xs:extension base="SettingTarget">
            <xs:attribute name="Fix" type="xs:boolean" use="optional"/>
            <xs:attribute name="Replace" type="xs:boolean" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
Fix	This will declare the target value to be fixed. This means that any attempts to subsequently modify the value will result in an error.
Replace	When a list is the target of the path, we can choose to either merge or replace the current contents of the list. The default behavior is to merge with the current contents.

3.6 设置约束（SETTINGS CONSTRAINTS）

约束被用于确认对定义成员设定数值或对关系内参与者的限定。这些限定在实例空间内同时在设计时间和部署时间进行评估。

所有设定约束使用约束定义来评估设定值。该约束定义使用设定声明来确认它所约束的数值。下面的约束定义实现采用两个参数和一个操作符的简单比较功能，然后评估该约束并且最终返回成功或错误。

```
<ConstraintDefinition name="SimpleOperatorComparison">
    <SettingDeclaration name="LHS" type="xs:any" access="writeonly"/>
    <SettingDeclaration name="operator" type="operator" access="writeonly"/>
    <SettingDeclaration name="RHS" type="xs:any" access="writeonly"/>
</ConstraintDefinition>
```

约束成员然后用于提供数值到用于评估的约束类型。

```
<Constraint name="constraintSecurityMode" definition="SimpleOperatorComparison">
    <Input name="LHS" path="webservice.securityMode"/>
    <SettingValue name="operator">==</settingValue>
    <SettingValue name="RHS">basicAuth</settingValue>

    </Constraint >
```

3．6．1 约束定义（Constraint definition）

约束定义限定根据一组输入值动作的约束。该约束可以被参数化来选择自定义行为或来支持使用参数来限定它的行为的简单的约束引擎。我们期望为简单的参数值约束和一组复杂的约束来写出一组标准约束定义，以用来支持在抽象对象之间的已知关系。

```
<xs:complexType name="ConstraintDefinition">
    <xs:complexContent>
        <xs:extension base="Definition"/>
    </xs:complexContent>
</xs:complexType>
```

3．6．2 约束成员（Constraint Member）

约束成员为特定约束定义确认一组输入值。该成员可以确认用于设定的静态值，并且可以使用输入语句来将约束设置绑定到路径上。

```
<xs:complexType name="ConstraintMember">
    <xs:complexContent>
        <xs:extension base="Member">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="Input" type="SettingTarget"/>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
input	A list of inputs to the constraint. An input identifies a path to the source setting value that will be passed to the constraint and constraint setting that will be set as a result. The source setting definition and the constraint setting definition should be compatible.

3．7 系统终端和资源定义（SYSTEM ENDPOINT AND RESOURCE

DEFINITIONS）

该部分描述用于抽象和具体对象定义的模式。

抽象对象定义陈述一组设置声明可以包含它所参与的关系的约束并且在运行时内具有相关的管理器。

下面是用于Web服务器的抽象系统定义。该Web服务器具有两种设定，并且具有需要它包含至少一个 vsite 类型的关系约束。

```
<AbstractSystemDefinition name="WebServer"
                          clrClassName="micorosft.sdm.IISSupportClass"
                    manager="IISSupportCode">

 <SettingDeclaration name="serverName" type="xs:string"/>
 <SettingDeclaration name="category" type="xs:string"/>

 <RelationshipConstraint name="containsVsites"
                         relationship="containmentRelationship"
                         myRole="parent"
                         targetType="vsite"
                         minOccurs="1"
                         maxOccurs="unbounded" />

</ AbstractSystemDefinition >
```

Vsite是一种包含服务器绑定信息的抽象终端定义。

```
<AbstractEndpointDefinition name="vsite">
    <SettingDeclaration name="ipAddress" type="ipaddress" required="true"/>
    <SettingDeclaration name="Endpoint" type="xs:int"/>
    <SettingDeclaration name="domainName" type="xs:int"/>
    <SettingDeclaration name="securityModel" type="securityModelEnum"/>
</AbstractEndpointDefinition>
```

用于前端Web服务器的具体系统定义把Web服务器分类视为静态上下文，包含可以代表1和100之间终端实例的单个byReference终端成员。该用于终端的具体终端定义嵌套在系统定义内，并且它限定用于vsite的ip Endpoint为Endpoint 80。

```
<SystemDefinition name="FrontendWebServer" implements="WebServer" >
    <SettingValue name="category" fixed="true">staticContentOnly</settingValue>
    <Port name="contentOnlyVsite" type="port80Vsite" isReference="true" minOccurs="1" maxOccurs="100"/>
    <PortDefinition name="port80Vsite" implements="vsite">
        <SettingValue name="Endpoint" fixed="true">80</settingValue>
    </PortDefinition >
</SystemDefinition>
```

3．7．1 对象定义(Object definition)

抽象和具体对象扩展下面的基础对象定义除了基本类型定义的元素之外它们享有约束对象参与的关系的能力。

```
<xs:complexType name="ObjectDefinition">
    <xs:complexContent>
        <xs:extension base="Definition">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="Flow" type="FlowMember" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="RelationshipConstraintGroup" type="RelationshipConstraintGroup"/>
                <xs:element name="RelationshipConstraint" type="RelationshipConstraint"/>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
Flow	A flow members declaration

RelationshipConstraint	Constraints on the relationships that these types can participate in.
RelationshipConstraintGroup	Constraints on the relationships that these types can participate in.

3．7．2 抽象对象定义(Abstract Object definitions)

抽象对象定义用于限定设计表面示出的构建模块，并且所有的具体对象由它们衍生而来：具体对象定义应当执行抽象对象定义。

抽象对象定义通过添加简单的继承扩展SDM对象：该扩展属性用于为抽象对象定义确认基本对象定义。然后该抽象对象定义从基本对象定义继承设定和关系约束。通过继承，对象定义可以通过添加新的设定和约束来扩展抽象对象定义的设定和约束。

抽象对象定义也可以对它们想参与的关系添加约束。例如，抽象对象定义会需要某种关系的存在，会限定放置在关系的另一端的对象定义，或限定在给定关系内参与的实例的设置。

3．7．2．1 抽象对象定义

所有抽象对象可以确认它们想进行关联的层。如果不提供这个，则假定该对象定义可以在任一层使用。抽象对象定义可以确认它们所扩展的基本对象定义，在这种情况下它们继承那个对象定义的设置和约束，并且可以在基本对象定义参与的关系内替换基础对象定义。

```
<xs:complexType name="AbstractObjectDefinition">
    <xs:complexContent>
        <xs:extension base="ObjectDefinition">
            <xs:attribute name="Layer" type="xs:string" use="optional"/>
            <xs:attribute name="Extends" type="QualifiedName" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
layer	The layer attribute identifies the layer at which this abstract object definition can be used. If it is not provided, the abstract object definition can be used at any layer.
extends	Identifies the abstract object definition that this object definition derives from.

	definition derives from.

3．7．2．2 抽象终端、系统和资源对象定义

在SDM模型内存在抽象对象定义的三种分类，它们为：抽象终端定义、抽象系统定义和抽象资源定义。这些中的每一个都为抽象对象定义的重新命名。

```
<xs:complexType name="AbstractEndpointDefinition">
    <xs:complexContent>
        <xs:extension base="AbstractObjectDefinition"/>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="AbstractSystemDefinition">
    <xs:complexContent>
        <xs:extension base="AbstractObjectDefinition"/>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="AbstractResourceDefinition">
    <xs:complexContent>
        <xs:extension base="AbstractObjectDefinition"/>
    </xs:complexContent>
</xs:complexType>
```

终端定义代表通信终端。在终端上的设定与在绑定过程中它的使用相关。例如，具有客户服务器协议，服务器终端定义可以使用设置模式来确认绑定到终端上所需的设定，客户终端定义可以陈述客户特定连接属性。

系统定义用于表示数据、软件、和硬件元素的集合。实例包括Web服务、数据库和交换机。资源定义被用于捕获可以被视为系统定义一部分的特定元素。

3．7．3 隐式基本定义（Implicit base definitions）

所有不扩展另一个抽象对象定义的抽象对象定义隐式扩展一个终端、系统或资源基本定义，如图15所示。这些基本定义形成每个树的根，这些树可以用在关系和约束说明中。这允许关系或约束用来指示出由根衍生而来的任一个类型可以被用于替换确认的根目标。这些根类型总是抽象的，并且不能直接进行实例化。

这些类型定义包括在模型内控制它们的示例的基本约束；它们可以在

System.sdm中找到。

3．7．4 具体对象定义（Concrete object definitions）

具体对象定义提供用于抽象对象定义的执行程序。该执行程序从对象和关系成员、已执行抽象定义的设置数值、新的设置说明、在成员和成员约束之间的流动中构建。

具体定义也可以包含嵌套定义的说明。这些定义可以在包含定义的范围内被用于成员，并且在定义范围之外在约束内被参考。

3．7．4．1 基本具体对象定义（Base concrete object definition）

基本具体类型扩展对象定义，继承设定说明、设计数据、可选的管理器参考、命名、它可以参与的关系的约束、为抽象定义的设置提供数值的能力和描述在它的设置和它的成员设置之间的流动的能力。该具体定义然后添加确认它所执行的抽象定义的能力，并且几个可选属性添加自定义定义的绑定行为的能力。

```
<xs:complexType name="ConcreteObjectDefinition">
    <xs:complexContent>
        <xs:extension base="ObjectDefinition">
            <xs:attribute name="Implements" type="QualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
Implements	Identifies the abstract type that this concrete type implements.

3．7．4．2 对象成员（Object Member）

对象成员应当参考抽象或具体对象定义。它们可以代表实例数组，在这种情况中它们可以为该数组限定上下边界。如果它们是参考成员，那么将对象实例化的用户应当为该成员显示构建实例。如果它们不是参考成员，那么运行时将在外部对象产生的同时创建实例。

```
<xs:complexType name="ObjectMember">
    <xs:complexContent>
        <xs:extension base="Member">
            <xs:attribute name="MinOccurs" type="xs:int" use="optional"/>
            <xs:attribute name="MaxOccurs" type="xs:int" use="optional"/>
            <xs:attribute name="IsReference" type="xs:boolean" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
MinOccurs	Lower bound on the number of instances associated with this member. If it is zero, then the member is optional. Default is one.
MaxOccurs	Upper bound on the number of instances associated with this member. Should be one or greater. Default is one.
IsReference	If this value is true, then the instance associated with the member should be explicitly created by the operator or referenced in another type. If it is false, then the instance is created when the type is created.

当母体从那些具有不依赖于母体生命周期的成员中销毁时，在SDM模型内我们需要区分当构建和解构母体时所产生的成员。我们为此目的使用IsReference属性。一个简单的类比是一C++声明，该C++声明允许基于new是否被用来产生实例的堆栈基础和堆基础的结构。如果成员被标记为IsReference，那么在操作者部分需要显示新的操作来产生实例以及将它与该成员相联系。

我们如此做有一些理由：

1、当操作者构建系统时我们示出构建IsReference成员的能力。这极大简化了操作者经历。

2、当我们处理sdm文档时，我们具有清楚的边界其中文档的实例空间可以与具体定义空间的实例空间不同。

3．7．4．3 关系成员（Relationship Member）

当关系成员被创建时关系成员确认将在对象成员之间存在的关系。关系实例由操作者显式创建或者由运行时隐式创建。前者的实例为实例之间的寄宿关系，后者为系统之间的通信关系。

```
<xs:complexType name="RelationshipMember">
    <xs:complexContent>
        <xs:extension base="Member">
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

3．7．4．3．1 寄宿成员（Hosting Member）

寄宿成员用于声明两个对象成员间的寄宿关系。对象成员可以是包含定义的直接成员或者有带有定义的成员关系的嵌套成员。应该在被参考的成员和包

含定义间有成员关系链。

```
<xs:complexType name="HostingMember">
    <xs:complexContent>
        <xs:extension base="RelationshipMember">
            <xs:attribute name="GuestMember" type="Path" use="required"/>
            <xs:attribute name="HostMember" type="Path" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
GuestMember	Identifies a member that has a definition compatible with the guest of the relationship. Member can be nested.
HostMember	Identifies a member that has a definition compatible with the host of the relationship. Member can be nested.

3．7．4．3．2 通信成员（Communication Member）

通信成员用于说明定义的直接系统成员的终端成员之间的通信关系。

```
<xs:complexType name="CommunicationMember">
    <xs:complexContent>
        <xs:extension base="RelationshipMember">
            <xs:attribute name="ClientMember" type="Path" use="required"/>
            <xs:attribute name="ServerMember" type="Path" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
ClientMember	Identifies an endpoint member that has a definition compatible with the client definition of the relationship. Endpoint member should be a member of an immediate system member of the definition.
ServerMember	Identifies an endpoint member that has a definitioncompatible with the server definition of the relationship. Endpoint member should be a member of an immediate system member of the definition.

3．7．4．3．3 包含成员（Containment Member）

包含成员用于说明类型成员被类型包含。每一个类型成员均可以被包含和被代理。包含成员自动将包含关系的父值设定为关系的this指针。

```
<xs:complexType name="ContainmentMember">
    <xs:complexContent>
        <xs:extension base="RelationshipMember">
            <xs:attribute name="ChildMember" type="Path" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
ChildMember	Identifies an immediate object member that will be contained by the parent.

3．7．4．3．4 代理关系（Delegation Member）

代理关系用于在外部类型上的终端定义成员和外部类型的直接系统成员上的终端定义成员之间设立代理关系。

```
<xs:complexType name="DelegationMember">
    <xs:complexContent>
        <xs:extension base="RelationshipMember">
            <xs:attribute name="ProxyMember" type="Path" use="required"/>
            <xs:attribute name="DelegateMember" type="Path" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
ProxyMember	Proxy identifies an immediate endpoint member on the system that does not have a containment relationship to the system. The definition of the member should match the definition of the proxy on the delegation relationship.
DelegateMember	Delegate identifies an endpoint member on an immediate member of the type. The type of the endpoint member should match the delegate type on the delegation relationship.

3．7．4．3．5 参考成员（Reference Member）

参考成员用于在外部系统的两个立即或嵌套成员之间设立参考关系。

```
<xs:complexType name="ReferenceMember">
    <xs:complexContent>
        <xs:extension base="RelationshipMember">
            <xs:attribute name="DependentMember" type="Path" use="required"/>
            <xs:attribute name="SourceMember" type="Path" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
dependentMember	The object member that depends on the source member. Should match the definition of the dependent object in the reference relationship.
sourceMember	The source object member. Should match the definition of the source object in the reference relationship.

3．7．4．4 终端定义（Endpoint definition）

终端定义通过添加说明嵌套资源类型、资源成员和宿主、包含和参考关系成员的能力来扩展基本对象定义。

```
<xs:complexType name="EndpointDefinition">
    <xs:complexContent>
        <xs:extension base="ConcreteObjectDefinition">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="ResourceDefinition" type="ResourceDefinition"/>
                <xs:element name="Resource" type="ResourceMember"/>
                <xs:element name="Hosting" type="HostingMember"/>
                <xs:element name="Containment" type="ContainmentMember"/>
                <xs:element name="Reference" type="ReferenceMember"/>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
ResourceDefinition	A nested resource definition that can be used by type members within the scope of the outer type definition.
Resource	A resource member declaration that references a resource type.
Hosting	A hosting relationship member declaration.
Containment	A containment relationship member declaration.
Reference	A reference relationship member declaration.

3．7．4．5 服务定义（Service definition）

系统通过添加对下列的支持来扩展基本类型：嵌套终端、系统和资源类型；终端、系统和资源成员和宿主、包含、连接、代理和参考关系。

```
<xs:complexType name="ServiceDefinition">
    <xs:complexContent>
        <xs:extension base="ConcreteObjectDefinition">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="EndpointDefinition" type="EndpointDefinition"/>
                <xs:element name="ServiceDefinition" type="ServiceDefinition"/>
                <xs:element name="ResourceDefinition" type="ResourceDefinition"/>
                <xs:element name="Endpoint" type="EndpointMember"/>
                <xs:element name="Subsystem" type="SubSystem"/>
                <xs:element name="Resource" type="ResourceMember"/>
                <xs:element name="Hosting" type="HostingMember"/>

                <xs:element name="Containment" type="ContainmentMember"/>
                <xs:element name="Connection" type="CommunicationMember"/>
                <xs:element name="Delegation" type="DelegationMember"/>
                <xs:element name="Reference" type="ReferenceMember"/>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
EndpointDefinition	A nested endpoint definition that can be used by members within the scope of the outer service definition
ResourceDefinition	A nested resource definition that can be used by type members within the scope of the outer type definition.
SystemDefinition	A nested system definition that can be used by members within the scope of the outer system definition
Endpoint	An endpoint member declaration that references an endpoint definition.
Subsystem	A subsystem member declaration that references a system definition .
Resource	A resource member declaration that references a resource definition.
Containment	A containment relationship member declaration.
Hosting	A hosting relationship member declaration.
Connection	A connection relationship member declaration.
Delegation	A delegation relationship member declaration.
Reference	A reference relationship member declaration.

3．7．4．6 资源定义（Resource definition）

资源类型可以包含嵌套类型定义、资源成员和宿主、包含和参考关系成员。

```
<xs:complexType name="ResourceDefinition">
    <xs:complexContent>
        <xs:extension base="ConcreteObjectDefinition">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="ResourceDefinition" type="ResourceDefinition"/>
                <xs:element name="Resource" type="ResourceMember"/>

                <xs:element name="Hosting" type="HostingMember"/>
                <xs:element name="Containment" type="ContainmentMember"/>
                <xs:element name="Reference" type="ReferenceMember"/>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
ResourceDefinition	A nested resource definition that can be used by members within the scope of the outer resource definition.
Resource	A resource member declaration that references a resource defintion.
Hosting	A hosting relationship member declaration.
Containment	A containment relationship member declaration.
Reference	A reference relationship member declaration.

3．7．4．7 关系规则（Relationship Rules）

为了对象定义的特殊实例，下表确认了具有每个实例可实行的规则的基数关联。

3．7．4．7．1 系统规则（system Rules）

Definition	Role	System	Endpoint	Resource
System	Parent(0...*)	Contains	Contains	Contains
	Member(1...1)	ContainedBy	Not Allowed	Not Allowed
	Proxy(0...*)	Not Allowed	Not Allowed	Not Allowed
	Delegate(0...*)	Not Allowed	Not Allowed	Not Allowed
	Client(0...*)	Not Allowed	Not Allowed	Not Allowed
	Server(0..*)	Not Allowed	Not Allowed	Not Allowed
	Guest(1..1)	HostedBy	HostedBy (??)	HostedBy
	Host(0..*)	Hosts	Hosts	Hosts
	Source(0..*)	Provides	Not Allowed	Not Allowed
	Dependent(0..*)	Consumes	Not Allowed	Not Allowed

3．7．4．7．2 终端规则（Endpoint Rules）

	Role	System	Endpoint	Resource
Endpoint	Parent(0...*)	Not Allowed	Not Allowed	Contains
	Member(1...1)	ContainedBy	Not Allowed	Not Allowed
	Proxy(0...*)	Not Allowed	DelegatesTo	Not Allowed
	Delegate(0...*)	Not Allowed	Implements	Not Allowed
	Client(0...*)	Not Allowed	ConnectsTo	Not Allowed
	Server(0..*)	Not Allowed	ProvidesService	Not Allowed
	Guest(1..1)	HostedBy	HostedBy	HostedBy
	Host(0..*)	Hosts	Hosts	Hosts
	Source(0..*)	Not Allowed	Provides	Provides
	Dependent(0..*)	Not Allowed	Consumes	Consumes

3．7．4．7．3 资源规则（Resource Rules）

	Role	System	Endpoint	Resource
Resource	Parent(0...*)	Not Allowed	Not Allowed	Contains
	Member(1...1)	ContainedBy	ContainedBy	ContainedBy
	Proxy(0...*)	Not Allowed	Not Allowed	Not Allowed
	Delegate(0...*)	Not Allowed	Not Allowed	Not Allowed
	Client(0...*)	Not Allowed	Not Allowed	Not Allowed
	Server(0..*)	Not Allowed	Not Allowed	Not Allowed
	Guest(1..1)	HostedBy	HostedBy	HostedBy
	Host(0..*)	Hosts	Hosts	Hosts
	Source(0..*)	Not Allowed	Provides	Provides
	Dependent(0..*)	Not Allowed	Consumes	Consumes

3．7．4．7．4 注意

每一个实例应该准确地参与一个包含关系和至少一个寄宿关系。

这意味着：

A）非参考成员应该确定一个包含关系

B）为了构建参考成员应该确认一个包含关系

C）不具有包含关系的参考成员应该仅代表其他成员。

3．8 关系（RELATIONSHIPS）

关系用于确定类型之间可能的交互。它们是二进制且定向的，每个均确定可参与关系的实例的类型。关系也可以包含参与关系的实例的设置并可以横跨关系使设定值流动。

以下是为在类型部分中描述的网页服务器上的webApplication提供的可能的寄宿关系。该关系包含约束，它验证两个系统的安全模式是否兼容，并且包含从vsite到vdir复制的服务器名的设置流成员。

```
<AbstractHostingDefinition name="vsiteHostsVdir" guestType="vdir" hostType="vsite">

    <ObjectConstraint name="checkCompatibility" primaryRole="guest" primaryType="vdir">
        <Constraint name="constrainSecurityModel" type="SimpleOperatorComparison">
            <input name="LHS" path="host.securityModel"/>
            <settingValue name="operator">==</settingValue>
            <input name="RHS" path="guest.securityModel"/>
        </Constraint>
    </ ObjectConstraint >

    <flow type="copy" name="copyServerToVdir">
        <input name="source" path="host.server"/>
        <output name="destination" path="guest.server"/>
    </flow>

</ AbstractHostingDefinition  >
```

关系通过说明关系成员来使用，所述关系成员确认将参与关系的类型成员。

```
<SystemDefinition name="testSystem" implements="ApplicationSpace">
    <resource name="myVdir" type="vdir" isReference="false"/>
    <resource name="myVsite" type="vsite" isReference="false"/>
    <hosting relationship="vsiteHostsVdir" guestMember="myVdir" hostMember="myVsite"/>
</ SystemDefinition >
```

3．8．1 关系定义(Relationship definition)

基本类型定义添加对象约束并流向定义。对象约束是为对象实体提供的关于设置值的语句，该对象实体参与该关系的实体。例如，表示DCOM连接的通信关系可以校验为客户或服务器提供的安全设定是兼容的。在这种情况下，设置之间的精确关系可以作为设计过程的一部分容易地被捕获；关系上有四个阶乘设置的组合，但有效结合数量要小的多。

流给关系开发者以将值从一个实例转发到另一个实例的能力。这允许独立于对象定义的可能交互作用来开发此对象定义，并且为了完整地描述特定实例，它允许实例作为信息的参考点而独立存在，而并非需要关系图的子集。

关系的名字在包含关系的命名空间中应当是唯一的。

```
<xs:complexType name="RelationshipDefinition">
    <xs:complexContent>
        <xs:extension base="Definition">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="ObjectConstraintGroup" type="ObjectConstraintGroup"/>
                <xs:element name="ObjectConstraint" type="ObjectConstraint"/>
                <xs:element name="Flow" type="FlowMember"/>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
ObjectConstraint (group)	Constraints on the instances that participate in this relationship. See section: 3.5.3
Flow	Flow between the instances that participate in this relationship.

3．8．2 抽象关系（Abstract Relationships）

抽象关系是限定两个抽象对象定义的关系。它们表示两个定义之间可能的交互。

```
<xs:complexType name="AbstractRelationshipDefinition">
    <xs:complexContent>
        <xs:extension base="RelationshipDefinition">
            <xs:attribute name="extends" type="QualifiedName" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

3．8．2．1 抽象通信关系（Abstract Communication Relationship）

通信关系用于在两个终端定义之间捕获可能的通信链接。它们用于描述单独展开的软件元素之间的交互。通信关系模式通过添加客户端和服务终端参考来扩展基本关系模式。

```
<xs:complexType name="AbstractCommunicationDefinition">
    <xs:complexContent>
        <xs:extension base="AbstractRelationshipDefinition">
            <xs:attribute name="ClientDefinition" type="QualifiedName" use="required"/>
            <xs:attribute name="ServerDefinition" type="QualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
ClientDefinition	The defintion of the client instance involved in the communication relationship
ServerDefinition	The type of the server instance involved in the relationship

以下抽象类型对的结合对通信关系适用：

Client Type	Server Type
Endpoint	Endpoint

3．8．2．2 抽象寄宿关系（Abstract Hosting Relationship）

寄宿关系用于捕获客户请求宿主按顺序构造的事实。由于可以为客户提供多于一个的可能主机，所以这意味着寄宿关系也负责在主机上构建客户端。因此为了产生对象实例，从客户端到兼容主机应当存在寄宿关系。

例如，寄宿关系可以存在于Webservice对象定义和IIS对象定义之间。在这种情况下，该关系指示假定MyWebservice和MyIIS分别执行网络服务和IIS，在寄宿关系上使用管理器可能在系统MyIIS实例上产生系统MyWebservice实例。我们不知道是否可能产生该关系直到我们已经评估了在系统和关系上同时存在的约束为止。

```
<xs:complexType name="AbstractHostingDefinition">
    <xs:complexContent>
        <xs:extension base="AbstractRelationshipDefinition">
            <xs:attribute name="GuestDefinition" type="QualifiedName" use="required"/>
            <xs:attribute name="HostDefinition" type="QualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
GuestDefinition	Identifies the definition of the guest instance.
HostDefinition	Identifies the definition of the host instance.

以下抽象定义对的结合对寄宿关系适用：

Guest Type	Host Type
Endpoint	Endpoint
Resource	Resource
Resource	System
System	Resource
System	System

3．8．2．3 抽象包含关系（Abstract Containment Relationship）

两个抽象对象之间的包含关系捕获基于父类型的具体类型可以包含基于成员类型的成员的事实。包含暗示着父实例可以控制成员实例的生命周期并可以代理针对成员实例的行为。

```
<xs:complexType name="AbstractContainmentDefinition">
    <xs:complexContent>
        <xs:extension base="AbstractRelationshipDefinition">
            <xs:attribute name="ParentDefinition" type="QualifiedName" use="required"/>
            <xs:attribute name="MemberDefinition" type="QualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
ParentDefinition	Identifies the definition of the instance that will contain the member.
MemberDefinition	Identifies the definition of the instance that will be the contained member

以下抽象定义对的包含对包含关系适用：

Parent Type	Member Type
System	Endpoint
System	Resource
System	System
Endpoint	Resource
Resource	Resource

3．8．2．4 抽象代理关系（Abstract Delegation Relationship）

代理用于从外部系统到包含系统发送行为。我们做这些的方式是通过代理从外部系统的终端到内部系统的终端进行的。这有效地将所有的交互转发到了内部系统上的终端，而该些交互本来是到外部系统。代理可以是连锁式的，允许内部系统更近一步代理其他系统的行为。

代理关系限定了可参与代理的抽象终端定义的对。每一个关系限定了一个可作为代理人执行的抽象终端定义和一个可为其代理行为的抽象终端定义。

```
<xs:complexType name="AbstractDelegationDefinition">
    <xs:complexContent>
        <xs:extension base="AbstractRelationshipDefinition">
            <xs:attribute name="ProxyDefinition" type="QualifiedName" use="required"/>
            <xs:attribute name="DelegateDefinition" type="QualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
ProxyDefinition	Identifies the definition of the outer endpoint that delegates its behavior to the inner endpoint
DelegateDefinition	Identifies the definition of the inner endpoint that provides the required behavior.

以下抽象类型对的结合对代理关系适用：

Proxy Type	Delegate Type
Endpoint	Endpoint

我们允许资源和系统代理以支持层之间的绑定。例如，允许IIS在未部署文件系统的情况下示出部分文件系统。

3．8．2．5 抽象参考关系（Abstract Reference Relationship）

我们使用参考关系来捕获添加到寄宿关系相关性中的实例之间的强相关性。在部署时，这些相关性用于控制构建顺序；在装入和更新时，这些相关性用于控制系统之间的流参数。由于参考关系指示了较强的相关性，因此我们不能允许参考关系横跨系统边界。这意味着，一个系统中的资源不能在其他系统中在资源上具有相关性。这会使系统不在是部署的独立单元。在系统之间存在相关性的地方，我们使用通信关系。通信关系可以随着时间进行改变，而不需要系统的重新装入。

```
<xs:complexType name="AbstractReferenceDefinition">
    <xs:complexContent>
        <xs:extension base="AbstractRelationshipDefinition">
            <xs:attribute name="DependentDefinition" type="QualifiedName" use="required"/>
            <xs:attribute name="SourceDefinition" type="QualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
DependentDefinition	The definition of the instance that depends on the source instance

	instance
SourceDefinition	The definition of the source instance. This instance is not required to be aware of the dependency.

以下抽象类型对的结合对参考关系适用：

Dependent Type	Source Type
System	System
Resource	Resource

3．8．3 隐式基本关系（Implicit base relationships）

如图16所示，所有抽象关系隐式扩展基本关系中的一个。这些定义为每一个关系树形成一个根。通过做这些，我们可以从约束定义内部引用根的定义，并且可以从根类型来继承一般类型约束。

3．8．4 具体关系（Concrete Relationships）

具体关系是两个具体对象定义之间的关系。每个具体关系应当执行一个抽象关系。该抽象关系应当在抽象对象定义匹配对之间，该抽象对象匹配对直接或间接（通过继承）被具体对象定义执行。

```
<xs:complexType name="ConcreteRelationship">
    <xs:complexContent>
        <xs:extension base="RelationshipDefinition">
            <xs:attribute name="Implements" type="QualifiedName"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

3．8．4．1 寄宿关系（Hosting Relationship）

当我们向数据中心部署应用程序时，需要利用应用程序为系统解析所有未完成的寄宿关系。为了做这些，操作者将需要为每个需求的寄宿关系建立寄宿成员。为了简化操作者的任务并允许开发者引导部署过程，开发者可以替代地

建立一个具体寄宿关系。该具体寄宿关系用于对一组寄宿关系成员进行分组，通过这种方式，在部署应用程序时，操作者仅需要声明一个简单寄宿成员。

```
<xs:complexType name="HostingDefinition">
  <xs:complexContent>
    <xs:extension base="ConcreteRelationship">
      <xs:sequence>
        <xs:element name="Hosting" type="HostingMember" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="GuestDefinition" type="QualifiedName" use="required"/>
      <xs:attribute name="HostDefinition" type="QualifiedName" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
HostDefinition	The name of the guest concrete object definition to which this hosting relationship applies.
GuestDefinition	The name of the host concrete object definition to which this hosting relationship applies.
Hosting	A list of hosting relationship members that reference members rooted to the guest and host definitions of the concrete relationship

以下具体类型对的结合对寄宿关系适用：

Guest Type	Host Type
System	System

```
A guest can be bound to a host iff
For each guestMember in Guest
There exists one or more hostMember in Host where
guestMember.Type has a hosting relation with hostMember.type
and guestMember.hostConstraints validate against hostMember.settings
and hostMember.guestConstraints validate against guestMember.settings
and for each member of guestMember there exists a binding to a member of hostMember
```

例如下列具体关系将第三层系统（Bike）与第二层的主机（操作系统）绑定。在这种情况下，我们为具有“系统文件夹”默认值的寄宿关系定义一个设定。我们使该设定流向三寄宿成员中的一个，该寄宿成员定义第3层的应用程序的系统和第2层主机的系统之间的寄宿关系。

```
<HostingDefinition name="DefaultBikePlacement" guestDefinition="Bike" hostDefinition="OperatingSystem:OperatingSystem">
    <settingDeclaration name="fileLocationRelativeToRoot" definition="xs:sting" access="readwrite" dynamic="false"/>
    <settingValue name="dirPath">systemFolder</settingValue>
    <flow name="copyPath" definition="copy">
        <input name="source" path="dirPath"/>
        <output name="destination" path="bikeExecutableHost.hostRelativePath"/>
    </flow>
    <hosting name="bikeExecutableHost" relationship="fileDirectoryHost"
                guestMember="guest.bikeFile" hostMember="host.FileSystem"/>

        <hosting name="bikeEventKeyHost" relationship="registryKeyRegistryKeyHost"
                    guestMember="guest.bikeEventKey" hostMember="host.Registry.ApplicationEventKey"/>
        <hosting name="bikeSoftwareKeyHost" relationship="registryKeyRegistryKeyHost"
                    guestMember="guest.bikeSoftwareKey" hostMember="host.Registry.HKLM"/>
  </HostingDefinition >
```

3．8．4．2 参考关系（Reference Relationship）

我们可以使用两个具体类型之间的具体参考关系来捕获不涉及通信关系的系统之间的特殊相关性。例如，我们可以捕获一个应用程序被装入另一个已经存在的事实。

```
<xs:complexType name="ReferenceDefinition">
    <xs:complexContent>
        <xs:extension base="ConcreteRelationship">
            <xs:sequence>
                <xs:element name="Reference" type="HostingMember" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="DependentDefinition" type="QualifiedName" use="required"/>
            <xs:attribute name="SourceDefinition" type="QualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
DependentDefinition	The name of the dependent concrete object definition to which this reference relationship applies.
SourceDefinition	The name of the source concrete object definition to which this reference relationship applies.
Reference	A list of reference relationship members that reference members of the guest and host definitions.

如下具体类型对的结合对参考关系适用：

Dependent Type	Source Type
System	System
Resource	Resource
Resource	Endpoint
Endpoint	Resource

3.9对象和关系约束（OBJECT AND RELATIONSHIP CONSTRAINTS）

当在特定关系内使用时，我们使用对象和关系约束来限定具体空间的拓扑以及来约束在特定关系内使用时对象的设置。

例如在抽象对象定义（A）内我们或许想确定该抽象定义的执行程序应当包含另一个抽象对象实例（B）的一个实例。假定至少一个合适的包含关系已经存在，为了这样做，我们将在A内使用关系约束，如下所示：

```
<RelationshipConstraint name="AContainsB"
                relationship="ContainmentDefinition"
                myRole="parent"
                targetType="B"
                minOccurs="1"
                maxOccurs="1"/>
```

该约束确定应当存在包含关系，其中在该包含关系内A的执行程序充当母体的角色并且在该关系（成员）另一端的类型为类型B。如果我们想对B的结构进行更多的控制，我们可以在类型B的设置上增加约束，如下：

```
<RelationshipConstraint name="AContainsB"
            relationship="ContainmentDefinition"
            myRole="parent"
            targetType="B"
            minOccurs="1"
            maxOccurs="1">

 <Constraint definition="simpleValueConstraint"
       name="BValueConstraint">

   <input name="LHS" path="member.name" />
   <settingValue name="operator">=</settingValue>
   <settingValue name="RHS">myPort</settingValue>

   </Constraint>
</RelationshipConstraint>
```

在这种情况下，我们添加要求成员命名等于字符串“myPort”的约束。

我们也可以为关系添加约束；我们称这些为对象约束。从关系内我们约束了在关系内参与的对象。对于关系内每个角色，我们可以确定对象定义并且然后我们可以添加设置约束到那些对象定义上。从关系角度看基数总是为minOccurs＝1和maxOccurs＝1，因此这在约束声明内不显示出来。

```
<ObjectConstraint name="allowedPair"
    primaryRole="host"
    primaryType="IIS"
    secondaryRole="guest"
    secondaryType="webApp"/>
```

最后我们可以使用嵌套约束。这给了我们将约束链在一起的能力；外部约束为内部约束设定环境。下面是寄宿webapp系统的IIS系统实例，随后它约束webApp仅仅特定类型的包含终端。

在这种情况中，我们使用一组对象约束来确定一组可能性，这些可能性中至少有一个应当为真实的。

```
<AbstractSystemDefinition name="IIShost">
    <RelationshipConstraint name="WebAppHostConstraint" relationship="hostingRelationship" myRole="host"
targetType="webApp">
        <RelationshipConstraint name="WebAppContainsPort" relationship="containmentRelationship" myRole="parent"
targetType="portType">
            <ObjectConstraintGroup mode="oneTrue">
                <ObjectConstraint name="hasWebPort" primaryRole="member" primaryType="webPort"/>
                < ObjectConstraint name="hasSqlPort" primaryRole="member" primaryType="sqlPort"/>
            </ObjectConstraintGroup>
        </RelationshipConstraint >
    </RelationshipConstraint >
</ AbstractSystemDefinition >
```

嵌套约束形成我们可以从外部评估的路径。在该路径上的每个约束可以访问在该路径上的在前实例以及当前实例的设置。进行嵌套约束的评估就像该约束已经在确定系统内进行限定一样。

从foo的前景看，下面两种情形应当是等价的。在第一foo中在约束系统bar上放置嵌套约束，在第二个中，类型bar已经包含了约束。

情形1：

```
< AbstractSystemDefinition name="foo">
    < RelationshipConstraint name="containsBar" relationship="containment"
        myRole="parent" targetType="bar" minOccurs="1">
        < RelationshipConstraint name="containsX" relationship="containment"
            myRole="parent" targetType="X" minOccurs="1"/>
    </ RelationshipConstraint >
</ AbstractSystemDefinition >

< AbstractSystemDefinition name="bar"/>
```

情形2：

```
< AbstractSystemDefinition name="foo">
    < RelationshipConstraint name="containsBar" relationship="containment"
        myRole="parent" targetType="bar" minOccurs="1"/>
</ AbstractSystemDefinition >

< AbstractSystemDefinition name="bar">
    < RelationshipConstraint name="containsX" relationship="containment"
        myRole="parent" targetType="X" minOccurs="1"/>
</ AbstractSystemDefinition >
```

3.9.1约束模型(Constraint Model)

有两部分来约束模型：保护和判定。我们使用保护来限定我们执行判定的上下文关系。例如在关系内，我们使用保护来确定我们想执行判定的类型的特定组合。在对象内，我们使用保护来确定到其它对象的一组关系。

当它们的保护需求已经满足时，随后执行判定。我们具有两种形式的判定：验证设置值的设置约束和验证一组约束的群组约束。

我们可以在保护内嵌套保护，在这种情况中内部保护仅仅当外部保护满足

时进行检查。这允许我们构建支持关系结构验证的路径。

保护的组合和它的判定可以具有基数，该基数表明保护应当符合的次数以及判定评估为真。

更形式地，

```
Guard :== ObjectConstraint(ObjectDefintion, ObjectDefintion ,required)
            { (Guard | predicate) * } |

          RelationshipConstraint(RelationshipDefinition,
                                 TargetObject,lBound,uBound)
            { (Guard | predicate) * }
```

保护限定为ObjectConstraint或RelationConstraint。对象约束确定与关系每一端相联系的两个对象定义。关系约束确定关系定义和目标对象定义。当关系约束具有下限和上限时，对象约束可以为可选的或者为所需的。基数的该不同反映了关系可以仅仅曾经确定两种类型而类型可以参与多个关系的事实。

```
Predicate :== SettingsConstraint(rule)  | group{ (guard)* }
```

判定是包含规则的设定约束或包含一组保护的群组。判定在保护的上下文关系内进行评估。在设定约束的情况下，判定可以从根保护的所有者和由每个嵌套保护所确定的上下文关系来确定设定。群组用于确定一组保护，这组保护中至少一个应当匹配和评估为真。

示例：

```
1. RelationshipConstraint(containmentRelationship,webapp,0,1){}
```

只要存在有对webapp的包含关系，该示例就显示出评估为真的保护。该保护至多一次可以评估为真。进一步的匹配将导致向用户返回错误。

```
2. RelationshipConstraint(containmentRelationship,webapp,0,1)
   {
      SettingsConstraint(webapp.name=2)
   }
```

该示例添加判定到保护上。当关系和目标定义匹配以及设置约束评估为真时，该保护将仅仅评估为真。如果关系和目标定义匹配以及设置约束不真，那么错误将返回给用户。如果关系和目标类型匹配以及设定约束不止一次评估为真，那么错误将返回给用户。

```
3. RelationshipConstraint(containmentRelationship,webapp,0,1)
   {
      RelationshipConstraint(containmentRelationship,vdir,0,1)
   }
```

在本示例中，我们在保护内嵌套保护。当外部包含为真（包含约束的类型

也包含webapp）时，那么我们在外部保护环境内评估内部保护。这意味着内部关系约束将在webapp实例环境内进行评估。如果webApp包含零或一个vdirs，内部约束将返回真，如果它包含多于一个vdir，那么该约束将返回错误给用户。

```
4. ObjectConstraint(webapp,iis,0,1)
   {
      RelationshipConstraint(containmentRelationship,systemType,0,1)
      {
            TypeConstraint(webapp,vdir,0,1)
      }
   }
```

对象约束的环境是最初的对象定义（第一对象定义）。这意味着关系约束将在webapp的环境内进行评估。该关系约束限定两种可能的环境，第一为关系，它将为对象约束的环境，第二是目标对象定义，它是关系约束的环境。

```
5. RelationshipConstraint(containmentRelationship,webapp,0,1)
   {
      group
      {
         RelationshipConstraint(containmentRelationship,vdir,0,1)
         RelationshipConstraint(containmentRelationship,directory,0,1)
      }
   }
```

在本示例中，我们使用群组来包含都将在Webapp的环境内评估的两种关系约束。该群组将引发错误除非至少一个关系激发并返回真。在这种情况下，该Webapp应当包含Vdir或文件夹。

3.9.2基础约束(Base Constraint)

```
<xs:complexType name="Constraint">
    <xs:sequence>
        <xs:element name="Description" type="Description" minOccurs="0"/>
        <xs:element name="DesignData" type="DesignData" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="name" type="SimpleName"/>
</xs:complexType>
```

Attribute / element	Description
Name	Name of this constraint section
DesignData	Design surface specific information about this constraint

3.9.3对象约束(Object Constraint)

对象约束描述了关系角色中一个或两个的约束。该约束具有一个命名在它失败的情况下来辅助约束的确定，它包含一列设置约束，这些设置约束将与角色相联系的类型作为目标，并且它可以进一步将实例约束为与角色相联系的定

义衍生而来的对象。

```
<xs:complexType name="ObjectConstraint">
    <xs:complexContent>
        <xs:extension base="Constraint">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="SettingsConstraint" type="ConstraintMember"/>
                <xs:element name="RelationshipConstraint" type="RelationshipConstraint"/>
                <xs:element name="RelationshipConstraintGroup" type="RelationshipConstraintGroup"/>
            </xs:choice>
            <xs:attribute name="PrimaryRole" type="RolesList" use="required"/>
            <xs:attribute name="PrimaryObject" type="QualifiedName" use="required"/>
            <xs:attribute name="SecondaryRole" type="RolesList" use="optional"/>
            <xs:attribute name="SecondaryObject" type="QualifiedName" use="optional"/>
            <xs:attribute name="Required" type="xs:boolean" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
SettingConstraint	A list of setting constraints that apply relative to the Context of this constraint
RelationshipConstraint	A nested relationship constraint. The relationship is evaluated as if it had been declared on the type associated with the primary role.
RelationshipConstraintGroup	A nested relationship group – the relationships in the group are evaluated as if it had been declared on the type associated with the primary role. Previous constraints become well-know names for settings roots.
PrimaryRole	The name of a role in the relationship that this constraint targets.
PrimaryObject	The name of the object definition associated with the primary role.

SecondaryRole	The name of the other role on the relationship that this constraint targets.
SecondaryObject	The name of the object definition associated with the secondary role on the relationship. This is required if a secondary role is specified.
Required	If required is true then the constraint should match the definitions it has declared for the roles on the relationship. If required is false, then guard does not have to match the types in use. Required is used to force a relationship to use a particular combination of types.

3.9.4对象约束群组(Object constraint group)

对象约束群组允许对象约束集合被分组在一起使得它们可以使用at-least-one语义进行评估。该群组将返回错误除非至少一个对象约束在关系上与对象匹配，随后它的所包含的判定评估为真。如果约束为群组的直接成员，我们忽略用于类型约束的所需属性。

```
<xs:complexType name="ObjectConstraintGroup">
    <xs:complexContent>
        <xs:extension base="Constraint">
            <xs:sequence>
                <xs:element name="ObjectConstraint" type="ObjectConstraint" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
ObjectConstraint	A list of type constraints defined within the group

3.9.5关系约束(Relationship Constraint)

关系约束用于约束对象可以参与的关系。关系约束确定关系定义，可选地确定处在关系另一端的实例的对象定义和该关系的基数。该约束被给定一个命名使得它可以在错误信息中被确定。关系约束体包含对关系和处在关系另一端的实例两者的判定。

关系约束可以用于多个目的：不用额外判定简单地使用基数，它们可以用于确定应当为实例正确操作而提供的关系，具有这些判定它们可以用来限定用于实例的结构组，其中该对象期望与这些实例交互。

```
<xs:complexType name="RelationshipConstraint">
    <xs:complexContent>
    <xs:extension base="Constraint">
        <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="SettingsConstraint" type="ConstraintMember"/>
            <xs:element name="RelationshipConstraint" type="RelationshipConstraint"/>
            <xs:element name="RelationshipConstraintGroup" type="RelationshipConstraintGroup"/>
            <xs:element name="ObjectConstraint" type="ObjectConstraint"/>
            <xs:element name="ObjectConstraintGroup" type="ObjectConstraintGroup"/>
        </xs:choice>
        <xs:attribute name="Relationship" type="QualifiedName" use="required"/>
        <xs:attribute name="MyRole" type="RolesList" use="required"/>
        <xs:attribute name="TargetObject" type="QualifiedName" use="optional"/>
        <xs:attribute name="MinOccurs" type="MinOccurs" use="optional"/>
        <xs:attribute name="MaxOccurs" type="MaxOccurs" use="optional"/>
    </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
SettingConstraint	Constraints on the values of settings within the relationship or on objects at the other end of the relationship.
RelationshipConstraint	A relationship constraint that is evaluated in the context of the target object (target object definition should be specified). This is equivalent to the adding the constraint to the target object.
RelationshipConstraintGroup	A relationship group that is evaluated in the context of the target object (target object should be specified). This is equivalent to the adding the group to the target object.
ObjectConstraint	A nested object constraint that is evaluated as though it were part of the relationship defintion identified by the outer constraint.
ObjectConstraintGroup	A nested object constraint group that is evaluated as though it were part of the relationship definition identified by the outer constraint.
Name	Unique name for the constraint within the scope of the containing definition
Relationship	The name of the relationship definition that is being constrained
MyRole	The name of the role this object instance will plays in this relationship – this is the name corresponding to the attribute name in the relationship definition eg client/server, guest/host etc If this is not provided we infer the role from the types involved in the relationship.

TargetObject	Optional name of the definition of the object that can appear on the other side of the relationship
MaxOccurs	The maximum number of times instances of this object can be identified as a participant in the defined role in the named relationship. If this is zero, then the type explicitly forbids participation in the named relationship.
MinOccurs	The minimum number of times instances of this object can be identified as a participant in the defined role in the named relationship

3.9.6关系约束群组(Relationship Constraint group)

关系约束群组允许关系约束组被分组在一起使得它们可以作为带有at-least-one语义的判定进行评估。该群组将返回错误除非至少一个包含的关系约束与关

系定义和目标对象匹配以及它的所包含的判定返回真。如果在包含的约束内的任何判定返回错误，则这些错误被传播到用户。所包含关系约束的minOccurs基数被忽略，但是如果maxOccurs基数被忽略，那么错误将被返回给用户。

```
<xs:complexType name="RelationshipConstraintGroup">
    <xs:complexContent>
        <xs:extension base="Constraint">
            <xs:sequence>
                <xs:element name="RelationshipConstraint" type="RelationshipConstraint" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
relationshipConstraint	A relationship constraint defined within the group

3.10对象管理器(OBJECT MEMBER)

对象管理器为类型和关系将用户行为插入运行时环境所依靠的机构。对于它管理的每个类型，有几种管理器可以支持的角色：它可以参与类型的安装，它可以提供类型的CLR表示，它可以包含在与类型之间的绑定如何解决相关的策略决定内，以及它可以为复杂约束和流提供执行程序。

所有对象管理器角色通过CLR示出作为强烈命名的分类的进入点。对象管理器以与sdm中其它类型一样的方式进行打包和制成版本；它们分布在系统分布单元内并且它们的版本和强命名由它们进行声明的sdm文件衍生而来。

```
<xs:complexType name="Manager">
    <xs:sequence>
        <xs:element name="Description" type="Description" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="Name" type="SimpleName" use="required"/>
    <xs:attribute name="AssemblyName" type="xs:string" use="required"/>
    <xs:attribute name="Version" type="FourPartVersionType" use="optional"/>
    <xs:attribute name="PublicKeyToken" type="PublicKeyTokenType" use="optional"/>
    <xs:attribute name="Culture" type="xs:string" use="optional"/>
    <xs:attribute name="Platform" type="xs:string" use="optional"/>
    <xs:attribute name="SourcePath" type="xs:string" use="optional"/>
</xs:complexType>
```

Attribute / element	Description
Name	A unique name for this manager in the scope of the containing sdm file.
Description	A text description of the manager
AssemblyName	The assembly name
Version	The assembly version
PublicKeyToken	The public key token for the assembly
Culture	The culture of the assembly
Platform	The platform of the assembly
SourcePath	The path to the assembly within the SDU

3.10.1角色(Roles)

对象管理器对于它所支持的每个类型可以支持一个或多个角色。这些角色包括:

a） 为类型或关系评估约束

b） 为类型或关系评估流

c） 为类型构建/解构/更新支持

d） 为类型或关系上的设定陈述对象表示

e） 为类型或关系执行发现

f） 围绕类型或关系支持设计表面特定UI

3.11 SDM文档结构(SDM DOCUMENT STRUCTURE)

sdm文档为一组关系、对象和管理器提供强的身份、版本和位置信息。

```
<xs:element name="Sdm">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Information" type="Information" minOccurs="0"/>
            <xs:element name="Import" type="Import" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="DesignData" type="DesignData" minOccurs="0"/>
            <xs:element name="SettingDefinitions" type="SettingDefinitions" minOccurs="0"/>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="AbstractEndpointDefinition" type="AbstractEndpointDefinition"/>
                <xs:element name="AbstractSystemDefinition" type="AbstractSystemDefinition"/>
                <xs:element name="AbstractResourceDefinition" type="AbstractResourceDefinition"/>
                <xs:element name="AbstractCommunicationDefinition" type="AbstractCommunicationDefinition"/>
                <xs:element name="AbstractHostingDefinition" type="AbstractHostingDefinition"/>
                <xs:element name="AbstractContainmentDefinition" type="AbstractContainmentDefinition"/>
                <xs:element name="AbstractDelegationDefinition" type="AbstractDelegationDefinition"/>
                <xs:element name="AbstractReferenceDefinition" type="AbstractReferenceDefinition"/>
                <xs:element name="ReferenceDefinition" type="ReferenceDefinition"/>
                <xs:element name="HostingDefinition" type="HostingDefinition"/>
                <xs:element name="EndpointDefinition" type="EndpointDefinition"/>
                <xs:element name="ResourceDefinition" type="ResourceDefinition"/>
                <xs:element name="ServiceDefinition" type="ServiceDefinition"/>
                <xs:element name="ConstraintDefinition" type="ConstraintDefinition"/>
                <xs:element name="FlowDefinition" type="FlowDefinition"/>
                <xs:element name="Manager" type="Manager"/>
            </xs:choice>
        </xs:sequence>
        <xs:attributeGroup ref="NamespaceIdentity"/>
        <xs:attribute name="documentLanguage" type="Culture"/>
    </xs:complexType>
</xs:element>
```

3.11.1信息（Information）

SDM文档的该信息部分包含人工可读的信息来支持sdm文档的确定和管理。

```
<xs:complexType name="Information">
    <xs:annotation>
        <xs:documentation>Human readable information about the SDM Definition library.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="FriendlyName" type="xs:string" minOccurs="0"/>
        <xs:element name="CompanyName" type="xs:string" minOccurs="0"/>
        <xs:element name="Copyright" type="xs:string" minOccurs="0"/>
        <xs:element name="Trademark" type="xs:string" minOccurs="0"/>
        <xs:element name="Description" type="Description" minOccurs="0"/>
        <xs:element name="Comments" type="xs:string" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
```

Attribute / element	Description
FriendlyName	
CompanyName	
Copyright	
Trademark	
Description	
Comments	

3.12变更请求（CHANGE REQUEST）

图17显示了变更请求的实例。变更请求确定一组对SDM运行时的变更。使用变更请求，通过允许构建请求的API's或者以xml格式来启动对运行时的所有变更。

初始请求包含单个的群组动作。由于请求由运行时处理，更多结构通过嵌套群组被添加并且更多动作作为扩展和流过程的结果被添加。已经通过该评估过程并且现在准备好相对目标机器执行的变更请求被称为完全合格的变更请求。参见3.13部分可以得到更多信息。

3.12.1一致性规则(Consistency rules)

当动作在SDM实例空间上执行时，我们验证在动作完成之后SDM实例空间中的所有实例依旧处在一致状态。通过一致状态我们意味着应用到实例的所有约束依旧有效。例如，如果我们生成一个需要连接到服务器的客户端的实例，当用于生成和连接客户端的动作顺序是完整的，在客户端和服务器之间应当存在该连接。

用于评估模型一致性的约束可以在每次动作基础上或者在一组动作结束的基础上进行评估。我们称这两种类型的一致性为操作一致性和处理一致性。

如果对象在处理完成后将处在不一致，我们允许用户显示标记那个实例为脱机。当实例脱机时，我们不评估应用到实例的约束并且该实例从其它实例的角度将不显示出存在。这可以意味着依次所有那些实例也应当被标记为脱机。脱机从母体到下一代和从宿主到客户端进行扩展，因此标记系统为脱机将标记所有它的拥有实例为脱机以及标记寄宿在它上的所有实例为脱机。

3.13模型评估（MODEL EVALUATION）

在该部分我们在SDM运行时范围内描述SDM模型的行为。

3.13.1定义空间（Definition space）

定义空间包含sdm运行时已知的所有定义。图18的步骤限定了将新的定义载入运行时的实例过程。该过程也由编辑过程共享，当设计表面验证sdm文档时产生该编辑过程。

3.13.1.1载入（Load）

sdm文档作为sdu一部分或者作为单独文档提供给运行时。我们试图从盘上载入该文件。

Validation error	Description
Unknown file	We could not find the file in the specified location.
Access denied	We were denied access to the file.

3.13.1.2模式验证（Schema validation）

第一步是验证sdm文档匹配sdm模式。在这一点我们将为所有未知元素、未得到所需元素的类型或属性或包含无效数据的类型返回错误。

Validation error	Description
Invalid document	The document has invalid xml syntax – unclosed nodes, more than one top level node etc.
Unknown element	An unexpected element was found in the sdm document
Unknown Attribute	And unknown attributed was found in the sdm document
Invalid value	A value failed the schema validation (this does not include setting value validation)
Missing attribute	A required attributed was missing
Missing element	A required element was missing
Invalid attribute combination	A combination of attributes or elements was used that was invalid such as: - minOccurs != maxOccurs | minOccurs = 0 with byValue

We could return warnings for unknown elements and attributes and just ignore them.

3.13.1.3设置值和类型解析（Setting value and type resolution）

在类型解析阶段，我们在sdm文件内解析对类型的所有参考（在模式中任何地方使用合格的命名）。首先我们验证在文档范围内的所有类型参考是有效的。这些为不包含别名的所有类型参考。然后我们试着解析所有输入语句。如果我们不能解析输入语句，我们产生一个命名空间载入错误，如果我们可以解析和输入语句，我们试着在命名空间内定位类型。如果我们被迫从sdm文件载入命名空间，那么该命名空间解析过程可能产生其它错误。

Type resolution error	Description
Unknown type	A type could not be found in local or aliased namespace. This will occur for settings, system, endpoint, resource, relationships, constraints and flow.
Unknown namespace	A namespace could not be found. The namespace has not been previously loaded
Version conflict	We could not locate a namespace with matching version information.
Culture conflict	We could not locate a namespace with matching culture information.
Invalid use of type	Use of invalid type in this situation eg. Settings type in a relationship member etc.
Invalid setting value	A setting value failed to pass its type's validation.
illegal setting value	A setting value was provided that violated an access modifiers on a setting declaration or a fixed value declared previously in a type or base type.

3.13.1.4路径解析(Path resolution)

在路径解析过程中，我们试着解析所有成员路径和在文档内限定的设置。涉及成员的路径或具有未解决类型的设置将不引发错误。

Path resolution errors	Description
Unknown setting	A setting declaration was not found to match the path
Unknown member	A member declaration was not found to match a name in the path
Cardinality mismatch	A path to a setting value in flow and constraint statement included a member or relationship that had cardinality greater than one.
Type mismatch	The declared type of the variable and the type of the resolved setting or member did not match.
Use mismatch	The declared intent of the path – input or output – violated an access modifier on the setting declaration or a fixed modifier on a value.
No value	A required flow input path referenced a setting that has no default value and is not exposed so as to allow a user to provide a value.

Path resolution warnings	Description
Runtime path warning	A path that references an isReference member that references an abstract type may not be verifyable until runtime. (We cannot check that a member exists until the user creates the concrete type).
Flow warning	Flow warnings – we will raise a warning if a value is provided for a setting that is also a target of a flow defined at the same time or earlier. We will not raise a warning if the flow is defined after the setting value has been provided as long as the setting is not fixed.

3.13.1.5关系参与(Relationship participation)

在类型空间内，我们检查类型说明相对于关系内它的成员参与不违反任何约束。为了如此做我们评估没有相关设置和约束的所有类型和关系约束。

Type Space Constraint errors	Description
Relationship type violation	A relationship member identifies a particular relationship and two members that are incompatible based on the types identified by the relationship. Eg a vdirToVsite hosting relationship is declared between a vdir and a file.
Relationship use violation	A relationship is declared between two members but the use of the relationship in this context is not allowed because the members are not accessible. For example declaring a delegation relationship between twoendpoints that are not on systems in a direct containment relationship
Relationship constraint violation	A relationship has been declared between a combination of types that it does not support based on constraints within the relationship. For example a relationship may be declared between vsites but it only supports certain types derived from vsite.

Type space constraint warnings	Description
Possible Cardinality mismatch	When counted up the maxOccurs for a member results in a set of relationships that would violate the cardinality of a relationship constraint.

3.13.1.6实例模拟(Instance simulation)

在实例模拟中我们试图以这样一种方式流动数值和评估约束，这种方式为我们可以确定我们知道应当失败的约束但未标记在用户输入的基础上会或不会失败的约束。为了如此做，我们构建实例空间模型并且评估在该实例空间基础上的流动和约束。如果知道流动或约束导致了错误，那么我们产生一个错误，

如果它可能导致错误，那么我们产生一个警告。

我们使用minOccurs约束在所有byReference系统上构建实例空间变更请求。当minOccurs为0时，我们产生单个实例并且将它标记为可选的。然后我们通过我们用于标准变更请求的相同的扩展和流动过程来传送变更请求。

Simluation warning	Description
Optional system	Because the system is optional, the runtime could not fully determine all errors that may result from this configuration.

然后我们评估所有具有完全限定输入值的流动。如果该输入值不固定并且可以由用户变更，那么我们标记流动输出为临时的。临时输入将使得使用它的任何流操作成链。如果流不具有完整的输入值并且用户可以提供数值，那么我们标记所有流输出为未定义。来自可选系统的流动也导致临时数值。

Flow error	Description
Flow input undefined	A value was not provided for a required flow input value.

一旦我们具有流数值，我们就依据这些数值来评估约束。提供数值失败的约束将被产生为警告；当由于未定义数值使得约束不可以进行评估时，也将产生警告。

Setting constraint error	Description
Settings input undefined	A value was not provided for a required constraint input value.
Settings violate constraint	One or more input settings to a constraint resulted in a constraint violation

Setting constraint warnings	Description
Settings could violate constraint	A combination of input settings based on defaults can violate the constraint.
Settings constraint not evaluated	A constraint could not be evaluated because it depends on settings provided at deployment or use time.

3.13.2实例空间(Instance space)

模型评估过程由声明变更请求的提交而启动。该请求将包含在运行时内以实例为目标的一组产生、更新或删除操作。然后如图19所示，在将所需变更施

加到目标系统上之前我们传送该请求通过一系列流水线阶段。

下面的部分指出了每个扩展步骤的职责。

3.13.2.1请求提交(Request submission)

为了对系统启动变更，操作者或过程应当提交变更请求。该变更请求包含一组操作者想在运行时内对实例执行的动作；这些动作分为三组：产生动作、更新动作和删除动作。

然后该请求作为原子组动作处理，该原子组动作应当作为群组完成或失败。当评估该组动作是否导致对模型的有效变更时，这允许约束验证过程考虑在请求内的所有动作。

Change request validation errors	Description
Invalid document	

Unknown element/attribute	An unknown element or attributed was found in the xml schema

3.13.2.1.1 类型解析(Type resolution)

在类型解析阶段，我们对在变更请求内被参考的所有类型和成员进行解析。该变更请求将假定这些已经由运行时载入；如果它们不存在时，该运行时将需要启动载入/编辑动作。

3.13.2.1.2路径解析(path resolution)

在路径解析阶段，我们对在变更请求内对现存实例和由产生动作限定实例的参考进行解析。

3.13.2.2扩展(Expansion)

扩展是我们得到变更请求并且填充需要来执行该请求的所有剩余动作的过程：总体上这些动作为用于类型和关系实例的构建和解构动作。理论上操作者可以为构建或解构实例所需的所有动作提供详细内容，但是我们不需要这样因为它将使得变更请求编辑过程非常复杂。相反，我们试图尽可能多的自动化该过程：操作者通过确认在byReference成员上的动作提供关于他们所想变更的关键信息；然后我们在嵌套的byReference和byValue成员和关系上填入剩余动作。

3.13.2.2.1数值成员

在扩展阶段我们确定所有非参考类型成员。我们知道这些成员的基数并且我们知道所有所需的参数，因此对于每个成员我们为那些母体正在产生的成员添加产生请求到变更请求上。如果该变更请求包含解构操作，我们为所有它们的包含实例添加解构操作。

3.13.2.2.2参考成员扩展（发现）

总体上，参考成员比数值成员需要更多信息来进行构建。它们的基数经常未限定并且它们可以具有需要数值的部署时间设定以便使得实例被构建。因此扩展byReference成员的过程可以比运行时处在提供位置需要更多信息。我们获得该信息的该过程称为发现。

该发现过程将填充参考类型成员作为构建或更新动作的一部分。仅仅具有支持发现的对象管理器的参考成员将参与该过程。

当新的实例被发现，我们使用实例特定密钥数值首先检查该实例已经不在SDM数据库内存在。一旦我们知道它是一个新的实例，那么我们根据我们发现的成员类型分类实例。如果该实例不与成员匹配或存在模糊匹配，那么我们将成员参考留为空并且标记该实例为脱机或未完成。

3.13.2.2.3关系扩展

一旦我们知道将被构建的所有关系实例，我们就产生将类型实例绑定在一起的关系实例。如果类型实例被销毁，那么我们就删除参考类型实例的所有关系实例。

为了产生关系我们返回成员空间来确认应当在实例之间存在的关系的结构。在类型成员具有基数比1大的地方，我们不得不推断关系拓扑。我们将在XX部分详细讨论我们如何这样做。

3.13.2.3流(Flow)

在流阶段中我们穿过所有关系实例评估流。该阶段可以为受变更参数流动影响的实例添加更新请求到变更请求上去。

作为变更请求的结果，流通过确定具有更新设定的实例组而被评估。对于这些中每一个，依赖于修正设置的任何流出的设流经评估并且目标节点添加到变更实例组上。该过程持续到该组为空或该组包含循环为止。

Error/warning	Description
Unterminated flow	

3.13.2.4重复检测

重复检测过程相对已经在sdm数据存储器内存在的实例匹配扩展实例。例如，我们将检测是否另一个应用程序已经安装了共享文件。当我们检测实例已经存在时我们可以依赖于现存实例版本进行几个动作中的一个：

a） 我们可以让安装失败

b） 我们可以参考计算实例

c） 我们可以升级实例

d） 我们可以同步安装

3.13.2.5约束评估

在约束评估阶段，我们检查在变更请求已经处理后在模型内的所有约束依旧有效。

For v1 we may have to visit every node in the graph that has a constraint as determining the scope of constraints may be difficult (we may be able to tag the member space in such a way that we can prune the instance space)

3.13.2.6请求排序

我们现在具有动作的完整列表，因此我们可以使用在系统之间的关系来确定有效的变更排序。

3.13.2.7执行

我们分配命令分组为机器特定的动作组。我们应当支持这些机器特定组的交叉机器同步。

3.13.2.8请求返回

基于受影响的寄宿关系，通过拆散变更请求成为分布的部分来执行变更。一旦所有部分完成（或失败）结果在运行时内进行核对并且总结返回到用户。

3.13.3深度扩展(Expansion in depth)

在该部分我们对类型和关系的扩展过程进行详细讲述。

3.13.3.1参考成员扩展（发现）

与寄宿关系用于构建类型的新的实例相同的方式，我们也使用寄宿关系来

发现存在的类型实例。该寄宿关系唯一地进行安置来做这一点，因为它自身知道类型实例在宿主上表示的方式。

当参考成员为发现进行标记时我们检查来看是否寄宿关系支持发现。如果是支持，我们传递宿主实例到该关系去，并且要求它为它在宿主上发现的客户端实例返回构建动作。

我们使用验证来发现实例不再存在。这再次使用寄宿关系来验证在宿主上客户端的存在。如果客户端不再存在，那么寄宿关系就将解构动作添加到变更请求。

3.13.3.2非参考成员扩展

运行时通过为类型的每个非参考成员简单添加构建或解构动作来处理所有非参考成员扩展，其中上述类型已经在变更请求内为构建或解构进行确认。

3.13.3.3通信关系扩展

如果操作者在存在通信关系成员的地方没有说明在两个类型成员之间的通信关系实例，那么我们通过假定在成员之间的完全连接网来扩展通信关系。

> The connectivity constraint may be loosened if there are important topologies that we cannot capture, but loosening it complicates the deployment process. For example, if we allowed looser topologies we could provide a way to create and manage these wires, do path checking for any changes, and expose the topology to the operator creating the deployment.

这意味着什么？如果两个成员在成员空间内连接，那么每个成员的所有实例应当能够互相看到。如图20所示，给出下面两个成员，由成员基数限制的实例空间拓扑。两个实例成员在1800示出。在1802，示出了在两个实例最大值之间的简单的点对点关系。在1804，示出了连接的输出端。实例可以为穿过一组服务器载入平衡请求的客户端。在1806，示出了连接的输入端。实例可以为享有各个服务器的一组客户端。在1808，示出了上面一组客户端享有一组服务器的情况的组合。

当我们构建通信链接时，代理终端变成透明的使得我们以与所有通信关系匹配的连接而告终，其中如果代理终端被移除上述这些通信关系将会存在。图21显示了两个结构1902和1904，只要涉及到在A、B和C的实例之间的连接这两个结构就是等价的。

3.13.3.4寄宿关系扩展

在寄宿关系是模糊的地方，我们需要寄宿关系的操作者或管理者来确定正

确的拓扑。

如果寄宿关系支持扩展，那么我们传递该组宿主和客户端到关系管理器并且要求管理器返回正确的构建动作。如果管理器不支持扩展那么我们返回变更请求到操作者使得他们可以提供更多信息。

3.13.3.5参考关系扩展

3.13.3.6包含关系扩展

包含关系决不会是模糊的，因此运行时可以总是添加合适的构建动作到变更请求去。

3.13.3.7代理关系扩展

为了扩展，代理关系遵循与通信关系一样的规则。

3.13.4流动(Flow)

3.13.5执行(Execution)

3.14 SDM实例空间(SDM INSTANCE SPACE)

下面部分为sdm运行时的实例空间限定对象模型。该实例空间用于跟踪由sdm建模的系统构造的变更。

图22举例说明了提供实例空间概况的示例性UML图表。方框2002、2004、2006和2008显示了在该文档其它部分限定的类型。

该实例空间围绕由变更请求启动的成版本的变更进行构建。每个实例都可以具有线性系列版本，该线性系列版本代表对运行实例进行的原子变更。在未来版本扩展到运行系统之前，它们也可以存在于运行时内。

对于该版本的SDM模型，我们仅仅允许对给定实例的线性变更。将来我们可以允许版本分支和引入版本解析模型。这将相对特定实例允许多于一个的变更为未完成的。

由于我们确实允许线性制成版本，我们可以载入构建在前面变更上的一系列变更请求。这支持在例如滚动升级这样的过程中可以采取的动作顺序的在前验证。

3.14.1 SDM实例(SDM Instance)

所有实例从sdm实例衍生而来。它们共享为设定模式限定数值的元素和与在实例定义上的成员相匹配的成员列表。它们也共享为该实例限定唯一标识符的

一组属性、实例的版本号、实例的命名以及指示该版本是否代表系统运行状态的标记。

```
<xs:complexType name="sdmInstance">
    <xs:sequence>
        <xs:element name="settingValues" type="settingValues" minOccurs="0"/>
        <xs:element name="member" type="member" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="id" type="instanceID" use="required"/>
    <xs:attribute name="version" type="xs:int" use="required"/>
    <xs:attribute name="isCurrent" type="xs:boolean" use="required"/>
    <xs:attribute name="name" type="xs:string" use="optional"/>
    <xs:attribute name="incomplete" type="xs:boolean" use="required"/>
</xs:complexType>
```

Attribute / element	Description
settingsValues	A list of setting values that defines the desired state of the modeled system object. This list includes all the values defined by, the developer on the definition or the member, the operator when deploying the instance or via flow from related instances.
member	This is a list of members that represent the members on the definition. Each member identifies the instances assigned to the member.
id	An identifier for the instance that is unique at global scope (to support distributed runtimes)
version	The version number increments linearly with changes to the instance.
isCurrent	This is a flag that indicates whether this version represents the running state of the system. There can exist latter versions that represent updates that have not been propagated to the running system.
name	A unique name for the instance within the scope of its containing member (may not be unique from the perspective of delegated members)

3.14.2 成员(Member)

成员用于联系为一组参考实例的实例成员。实例成员由实例定义进行限定。参考实例是已经为所述成员产生的实例或所述成员进行代理的实例。成员可以代表一数组，在这种情况下，可以有多于1个的参考实例。

```
<xs:complexType name="member">
    <xs:sequence>
        <xs:element name="instance" type="instanceRef" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="memberDeclaration" type="qualifiedName" use="optional"/>
    <xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>
```

Attribute / element	Description
instance	An instance referenced by this member
memberDeclaration	The declaration of this member on the associated definition
name	The name of the associated member on the definition

3.14.3变更(Change)

变更代表对实例状态的变更。它将变更请求与受影响的实例组相关联。如果变更已经被执行时，它也确认变更的状态（参见XXX部分）和变更响应。

```
<xs:complexType name="change">
    <xs:sequence>
        <xs:element name="instance" type="instanceVersionRef" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="id" type="changeID" use="required"/>
    <xs:attribute name="status" type="changeStatus" use="required"/>
    <xs:attribute name="changeRequest" type="qualifiedName" use="optional"/>
    <xs:attribute name="changeResponse" type="qualifiedName" use="required"/>
</xs:complexType>
```

Attribute / element	Description
instance	A list of instance versions that we created as a result of the associated change request
id	A unique identifier for this change (at least unique to the runtime)

status	An enumeration identifying the current status of this change
changeRequest	A link to the change request that was used to create this change
changeResponse	A link to the results return from the execution of the change

3.14.3.1变更状态

变更请求可以处在下面状态之一：

- 未启动—指示没有相对变更请求试图执行
- 处理中—指示它目前正在执行
- 完成—指示变更请求成功完成
- 失败—指示变更请求失败并且该变更处在未完成状态

- 退回一指示失败的变更请求已经成功退回

```
<xs:simpleType name="changeStatus">
    <xs:restriction base="xs:string">
        <xs:enumeration value="notStarted"/>
        <xs:enumeration value="inProgress"/>
        <xs:enumeration value="completed"/>
        <xs:enumeration value="failed"/>
        <xs:enumeration value="rolledBack"/>
    </xs:restriction>
</xs:simpleType>
```

3.14.4具体对象实例(Concrete object instance)

具体对象实例代表实例为由类型属性确定的具体类型。由于实例存在实字表达，因此我们需要跟踪实例与它现实中的对应部分是否相一致。作为该变更的结果，我们也想知道该实例是否联机。联机实例应当相对所有它的约束有效。脱机实例对它参与的通信关系的其它参与者显示出不可见。如果实例不完整，那么在实例联机之前需要进一步的变更请求。

```
<xs:complexType name="ObjectInstance">
    <xs:complexContent>
        <xs:extension base="sdmInstance">
            <xs:attribute name="inSync" type="xs:boolean" use="required"/>
            <xs:attribute name="online" type="xs:boolean" use="required"/>
            <xs:attribute name="type" type="qualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
inSync	This indicates whether the settings on the instance match those on its real world counterpart.
online	This indicates whether the real world counterpart should be considered online and active. An instance cannot be put in the online state if any constraints are not satisfies. An offline instances will not be visible to other participants in the communication relationships that reference it. (flow will not be evaluated?)
incomplete	This flag indicates that required information is missing from this version of an instance. This situation may arise as a result of a discovery process that could not identify all information required by instance or as a result of a change request that did not supply all required information.
type	A reference to the type of the instance.

3.14.5关系实例(Relationship Instances)

关系实例代表确认关系类型的实例。由于关系没有直接的现实表示，我们不需要保留关于关系是否一致或联机的信息。由于关系相对简单我们也不期望

它们不完整，尽管它们可以让它们的约束失败。

```
<xs:complexType name="relationshipInstance">
    <xs:complexContent>
        <xs:extension base="sdmInstance">
            <xs:attribute name="relationship" type="qualifiedName" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
relationship	The relationship type associated with this instance.

3.14.5.1包含实例

这代表了包含关系的实例。

```
<xs:complexType name="containmentInstance">
        <xs:complexContent>

        <xs:extension base="relationshipInstance">
            <xs:attribute name="parentInstance" type="instanceID" use="required"/>
            <xs:attribute name="childInstance" type="instanceID" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
parentInstance	Identifies the parent instance that participates in the relationship.
childInstance	Identifies the child instance that participates in the relationship.

3.14.5.2通信实例

这代表了通信关系的实例。

```
<xs:complexType name="communicationInstance">
    <xs:complexContent>
        <xs:extension base="relationshipInstance">
            <xs:attribute name="clientInstance" type="instanceID" use="required"/>
            <xs:attribute name="serverInstance" type="instanceID" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
clientInstance	Identifies the client instance that participates in the relationship.
serverInstance	Identifies the server instance that participates in the relationship.

3.14.5.3代理实例

这代表了代理关系的实例。

```
<xs:complexType name="delegationInstance">
    <xs:complexContent>
        <xs:extension base="relationshipInstance">
            <xs:attribute name="proxyInstance" type="instanceID" use="required"/>
            <xs:attribute name="delegateInstance" type="instanceID" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
proxyInstance	Identifies the proxy instance that participates in the relationship.
delegateInstance	Identifies the delegate instance that participates in the relationship.

3.14.5.4寄宿实例

这代表了寄宿关系的实例。

```
<xs:complexType name="hostingInstance">
    <xs:complexContent>
        <xs:extension base="relationshipInstance">
            <xs:attribute name="guestInstance" type="instanceID" use="required"/>
            <xs:attribute name="hostInstance" type="instanceID" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
guestInstance	Identifies the guest instance that participates in the relationship.
hostInstance	Identifies the host instance that participates in the relationship.

3.14.5.5参考实例

这代表了参考关系的实例。

```
<xs:complexType name="referenceInstance">
    <xs:complexContent>
        <xs:extension base="relationshipInstance">
            <xs:attribute name="sourceInstance" type="instanceID" use="required"/>
            <xs:attribute name="dependentInstance" type="instanceID" use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Attribute / element	Description
sourceInstances	Identifies the source instance that participates in the relationship.
dependentInstance	Identifies the dependent instance that participates in the relationship.

3.14.6实例(Instances)

这些实例组代表在sdmIinstance文件中存在的实例元素组。

```
<xs:group name="instances">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="SystemInstance" type="concreteTypeInstance"/>

      <xs:element name="portInstance" type="concreteTypeInstance"/>
      <xs:element name="resourceInstance" type="concreteTypeInstance"/>
      <xs:element name="member" type="member"/>
      <xs:element name="containmentInstance" type="containmentInstance"/>
      <xs:element name="communicationInstance" type="communicationInstance"/>
      <xs:element name="hostingInstance" type="hostingInstance"/>
      <xs:element name="delegationInstance" type="delegationInstance"/>
      <xs:element name="referenceInstance" type="referenceInstance"/>
      <xs:element name="placementInstance" type="placementInstance"/>
  </xs:choice>
</xs:group>
```

3.14.7实例参考(Instance References)

3.14.7.1实例参考

实例参考是对实例的简单参考。将默认isCurrent实例除非在变更请求上下文进行参考以及实例受该变更请求影响。

```
<xs:complexType name="instanceRef">
    <xs:attribute name="instanceID" type="instanceID" use="required"/>
</xs:complexType>
```

3.14.7.2实例版本参考

实例版本参考确认实例的特定版本。

```
<xs:complexType name="instanceVersionRef">
    <xs:attribute name="instanceID" type="instanceID" use="required"/>
    <xs:attribute name="version" type="xs:int" use="required"/>
</xs:complexType>
```

3.15部署单元结构(DEPLOYMENT UNIT STRUCTURE)

要求

- 包含需要来安装一组SDM类型的所有位
- 可以被设计和版本化
- 容易构建/打包/传送
- 可以通过参考或包含物来参考其它SDU

- SDM类型定义的部署部分直接参考SDU内的文件

3.16局域化(LOCALIZATION)

我们需要确定SDM模型的什么部分支持局域化以及我们如何通过系统设计和部署来支持局域化。

第一种手段：

我们完全将局域化留给给单个类型去管理。局域化隐式穿过约束。局域化不是第一类型公民。这意味着：

a） SDU可以包含类型特定版本的执行程序：存在一个特定版本的执行程序。这意味着不可能有单独基于局域化而不同的执行程序。因此每个执行程序应当支持场所范围或者该执行程序应当为不同的类型（为此目的使用版本化将为错误行为！）

b） 然后局域化通过使用资源作为mixins来支持特定版本或者通过使用确认支持不同版本的执行程序的一组类型而来获得。

c） 客户端不能区分/要求服务器的局域化版本。

第二种手段：

局域化是与命名和版本一起的身份的第一类型公民。这意味着在对类型进行参考的任何地方都应当考虑局域化。

a） 客户端现在可以在包含、寄宿或者通信关系的任一个基础上区分服务器的局域化版本。

b） 部署引擎应当知道局域化并且允许操作者在类型的局域化版本之间选择。

c） SDU可以由命名确认，版本和场所或SDU也可以包含多个仅仅基于它们的场所而不同的执行程序（由于非局域代码应当放置在分开的sdu第一意味着SDU的更好粒度的打包，第二意味着我们可以具有多个带有相同命名的sdu……）

如果场所非常广泛地用作约束，从设计/ui角度看第二步骤潜在地会变得很复杂。例如，如果终端被局域化或者宿主将它们的客户端局域化，那么发现连接/设置会变得非常复杂。如果第二种手段通过第一种手段中的b）被用作提出的机制，那么复杂性可以比较容易地管理，但是某人将不得不确认、打包和传送局域化资源。

3.17版本化和变更管理

3.17.1总体说明

● 我们想能够在合适位置将系统版本化－ie应用qfe到sql而不变更实例身份。这意味着变更实例类型。

● 我们想允许版本化策略来控制允许的版本变更－例如系统类型设计者可以选择用于系统成员的版本化策略如何严格，或者操作者可以出于安全原因选择单方面地升级成员的版本。

● 我们想限制版本变更的扩展－例如，如果我们改变成员类型，我们不想不得不变更系统类型从而扩展类型变更到根。

● 中断变更将由在版本号前两部分内的变更来进行指示，非中断变更将由在版本号其次两部分内的变更来进行指示。

示例计算机环境

图23示出了总的计算机环境2300，它可以用来执行此处所论述的技术。该计算机环境2300仅仅是计算环境的一种实例，并不用来对使用范围或者计算机和网络体系结构的功能性进行任何限定。该计算机环境2300也不应当解释为对说明性的计算机环境2300中所示的组件中任何一个或其组合具有依赖性或具有与之相关的需求。

计算机环境2300包括计算机2302形式的通用计算设备。计算机2302可以为例如图1的计算设备102或者执行开发系统202或者为图2的控制器206、或者为图2的目标装置212、或者为图6的控制器620或目标622。计算机2302组件可以包括但不限于一个或多个处理器或处理单元2304、系统存储器2306以及将包含处理器2304的不同系统组件与系统存储器2306相结合的系统总线2308。

系统总线2308代表任何几种类型总线结构中的一个或多个，包括存储总线或存储控制器、外围总线、图形加速端口、以及处理器或使用不同总线体系结构中任一个的局域总线。作为举例，这些体系结构可以包括工业标准结构（ISA）总线、微信道结构（MCA）总线、增强型ISA（EISA）总线、视频电子标准协会（VESA）局域总线、以及也称为中层总线的周边元件扩展接口（PCI）总线。

计算机2302一般包括不同的计算机可读媒介。这种媒介可以由计算机2302访问的任何可用媒介，并且包括易失性的或非易失性的媒介、可拆卸或不可拆卸媒介。

系统存储器2306包括计算机可读媒介，该计算机可读媒介为易失性存储器，例如随机存取存储器（RAM）2310，和/或非易失性存储器，例如只读存储器（ROM）2312。基本输入/输出系统（BIOS）2314中包含了例如启动过程中帮助计算机2302内部件之间传送信息的基本程序并且存储在ROM 2312内。RAM 2310一般包含由处理单元2304直接可得到和/或马上在其上进行操作的数据和/或程序模块。

计算机2302也可以包括其它可拆卸/不可拆卸、易失性/非易失性计算机存储媒介。作为举例，图23示出了用于从不可拆卸、非易失性磁媒介（未示出）上读写的硬盘驱动器2316，用于从可拆卸、非易失性磁盘2320（如“软盘”）上读写的磁盘驱动器2318，和用于从可拆卸、非易失性光盘2324如CD-ROM、DVD-ROM或其他光媒介上读写的光盘驱动器2322。硬盘驱动器2316、磁盘驱动器2318和光盘驱动器2322分别通过一个或多个数据媒介接口2326而与系统总线2308相连接。作为选择，硬盘驱动器2316、磁盘驱动器2318和光盘驱动器2322可以通过一个或多个接口（未示出）和系统总线2308相连接。

盘驱动器和与它们相关联的计算机可读媒介为计算机2302提供了存储有计算机可读指令、数据结构、程序模块和其他数据的非易失性存储器。虽然本实例示出了硬盘2316、可拆卸磁盘2320和可拆卸光盘2324，但是应当理解，可存储数据并可由计算机存取的其他类型的计算机可读媒介也可以用于实现当前所示范的计算机系统和环境，这些其他类型的计算机可读媒介如盒式磁带和其他磁存储器、闪速记忆卡、CD-ROM、数字化视频光盘或其他光盘存储器、随机存取存储器（RAM）、只读存储器（ROM）、电可擦除可编程只读存储器（EEPROM）等等。

任意数目的程序模块均可存储在硬盘2316、磁盘2320、光盘2324、ROM 2312、和/或RAM 2310上，作为举例，包括操作系统2326、一个或多个应用程序2328、其他程序模块2330和程序数据2332。每一个这样的操作系统2326、一个或多个应用程序2328、其他程序模块2330和程序数据2332（或它们的其他组合）均可以实现全部或部分支持分布式文件系统的驻留成分。

使用者可以通过输入装置例如键盘2334和定点装置2336（如“鼠标”）来向计算机2302输入命令和信息。其他输入装置2338（未明确示出）可以包括麦克风、操纵杆、游戏板、圆盘式卫星电视天线、扫描仪和/或同类的装置。这些

和其他的输入装置通过输入/输出接口2340和与系统总线2308相耦合的处理单元2304相连接，但也可能通过其他的接口和总线结构相连接，例如并行端口、游戏端口或通用串行总线（USB）。

监视器2342或其他类型的显示装置也可以通过诸如像视频适配器2344之类的接口而与系统总线2308相连接。除监视器2342之外，其他的输出外围设备还可以包括组件，比如扬声器（未示出）和打印机2346，其可以通过输入/输出接口2340与计算机2302相连接。

计算机2302可以通过与一个或多个远程计算机的逻辑连接在网络环境下操作，该远程计算机例如远程计算装置2348。作为举例，远程计算装置2348可以是个人计算机、便携式计算机、服务器、路由器、网络计算机、对等装置或其他普通网络节点等等。远程计算装置2348作为便携式计算机示出，其可以包括相对于计算机2302所描述的多数或所有的元素和特征。

计算机2302和远程计算机2348之间的逻辑连接描述为本地网（LAN）2350和常规广域网（WAN）2352。这些网络环境常见于办公室、企业广域计算机网络、企业内部网和因特网。

当在LAN网络环境中实现时，计算机2302通过网络接口或适配器2354与本地网2350相连接。当在WAN网络环境中实现时，计算机2302典型地包括调制解调器2356或其他用于在广域网2352上建立通信的装置。在计算机2302内部或外部的调制解调器2356可以通过输入/输出接口2340或其他专用机构与系统总线2308相连接。应当理解的是，所示出的网络连接是示范性的，其他用于在计算机2302和2348之间建立通信链接的装置也可使用。

在网络环境中，例如在计算环境2300中所示出的，相对于计算机2302描述的程序模型或其中的部分存储于远程内存存储装置（memory storage device）中。作为举例，远程应用程序2358驻留于远程计算机2348的存储装置中。如所示出的目的，应用程序和其他可执行的程序组件，例如操作系统在此处作为不连续块示出，不过所公认的是，这些驻留的程序和组件在不同的时间存储在计算装置2302的不同存储组件中，并且这些程序是通过计算机的数据处理器执行的。

各种模块和技术将通过计算机可执行指令的普通语境在此进行描述，例如程序模块，通过一个或多个计算机或其他装置来执行。通常，程序模块包括执

行特殊任务或实现特殊抽象数据类型的例程、程序、对象、组件、数据结构等等。典型地，根据需要在各实施例中可以组合或分布所述程序模块的功能。

这些模块和技术的执行程序可以存储在一些形式的计算机可读媒介上或者跨越一些形式的计算机可读媒介进行传送。计算机可读媒介可以为计算机允许存取的任何可用媒介。以实例的形式并不是作为限制，计算机可读媒介可以包括“计算机存储媒介”和“通信媒介”。

“计算机存储媒介”包括以任何方法或技术执行的易失性和非易失性、可拆卸和不可拆卸媒介，其用于存储信息例如计算机可读指令、数据结构、程序模块或其它数据。计算机存储媒介包括但不限于RAM、ROM、EEPROM、闪存或其它存储技术、CD-ROM、数字通用盘（DVD）或其它光存储器、磁盒、磁带、磁盘存储器或其它磁存储设备、或者能够用于存储所需信息和能被计算机存取的任何其它媒介。

“通信媒介”一般包含计算机可读指令、数据结构、程序模块或者在调制数据信号内的其它数据，例如载波或其它传送机构。通信媒介也包括任何信息传送媒介。“调制数据信号”一词意思为具有一个或多个设定或改变特征的信号，这种设定或改变以在信号内编码信息的方式进行。作为举例而不作为限定，通信媒介包括有线媒介例如有线网络或单线连接和无线媒介例如声、RF、红外和其它无线媒介。上述的任何结合也包括在计算机可读媒介的范围内。

或者，部分框架可以在硬件或者硬件、软件和/或固件的组合物内执行。例如，可以设计或编程一个或多个专用集成电路（ASIC）或可编程逻辑设备（PLD）来执行框架的一个或多个部分。

结论

尽管本发明已经用语言对结构特征和/或方法动作做了详细说明，但是应当理解在示范性的附加权利要求中所限定的本发明并不是对所描述的特定特征或动作的限定。相反，这些特定特征和动作以实现所要求保护发明的示范性形式公开。而且，这些权利要求在范围和主题方面为示范性的。此处所描述特征的许多其它组合和部分组合可以在要求该申请为优先权的在后专利申请中进行保护。

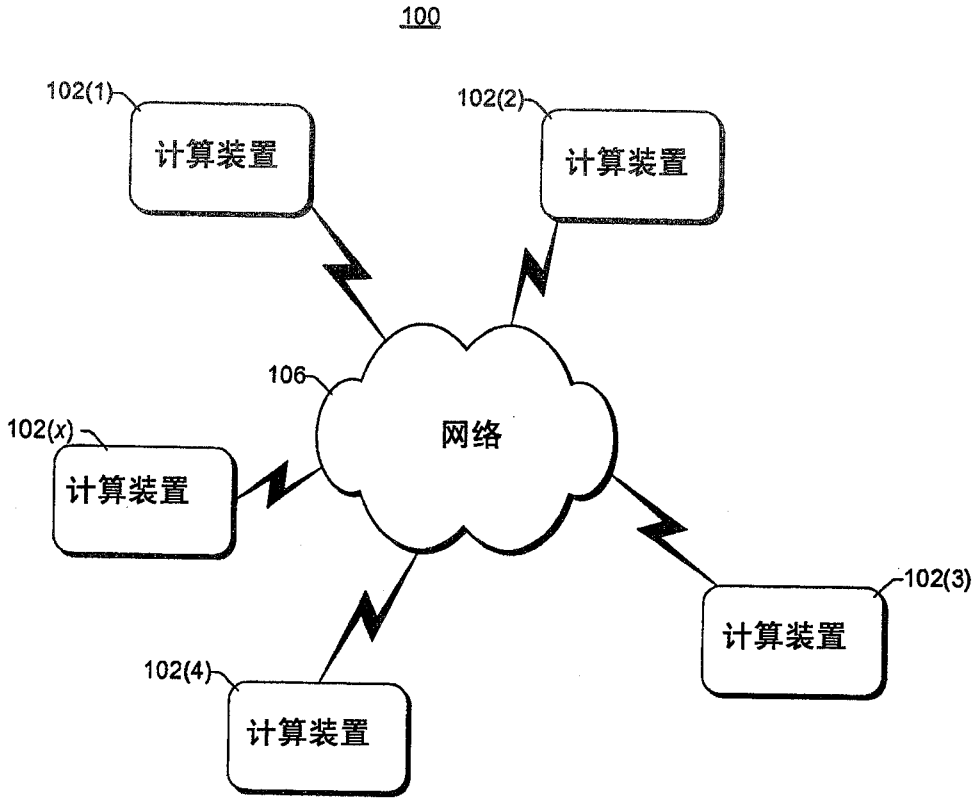


图 1

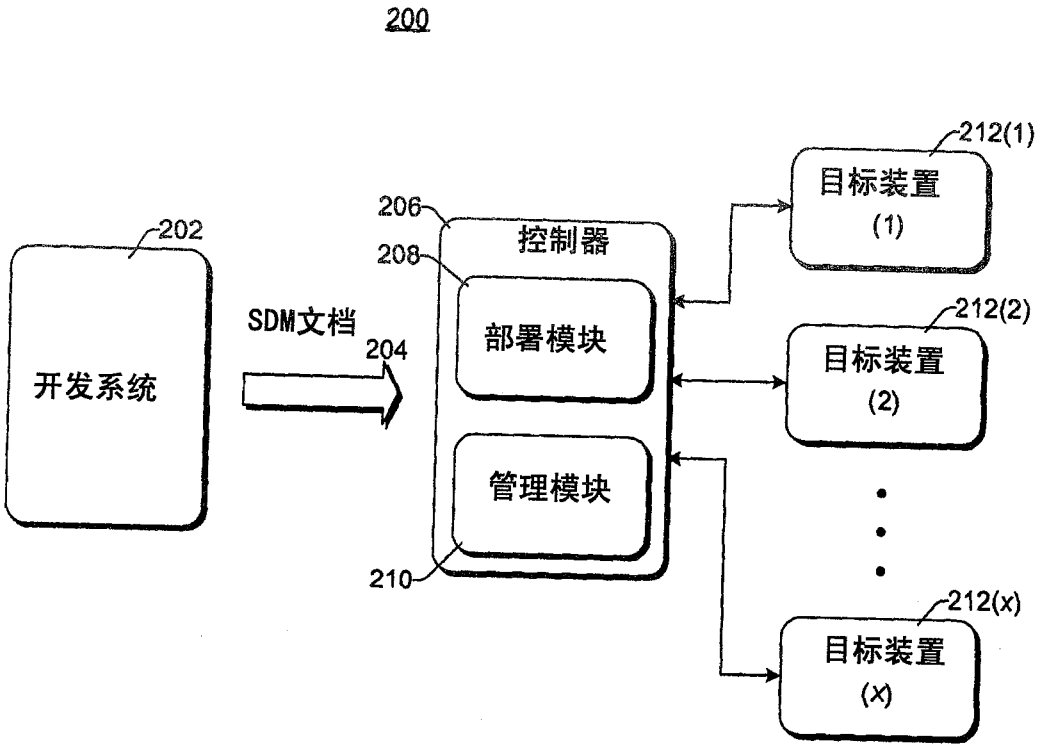

200
202
开发系统
SDM文档
204
206
控制器
208
部署模块
管理模块
210
212(1)
目标装置
(1)
212(2)
目标装置
(2)
212(x)
目标装置
(x)

图 2

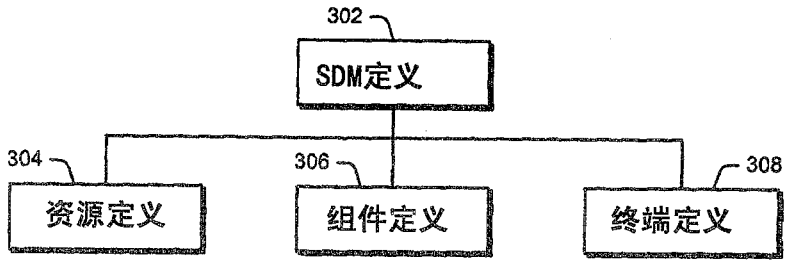

302
SDM定义
304
资源定义
306
组件定义
308
终端定义

图 3

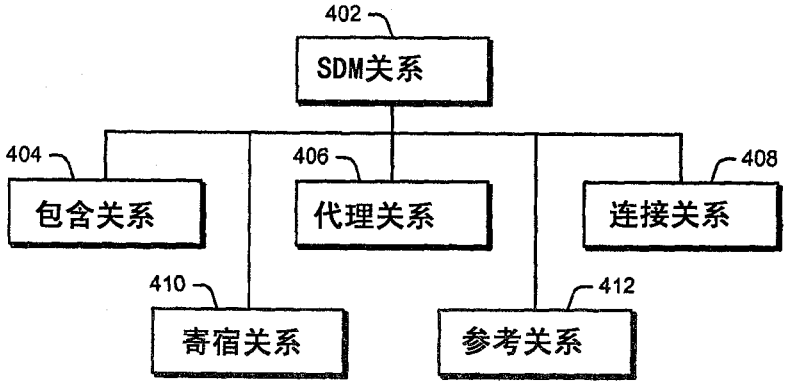

402
SDM关系
404
包含关系
406
代理关系
408
连接关系
410
寄宿关系
412
参考关系

图 4

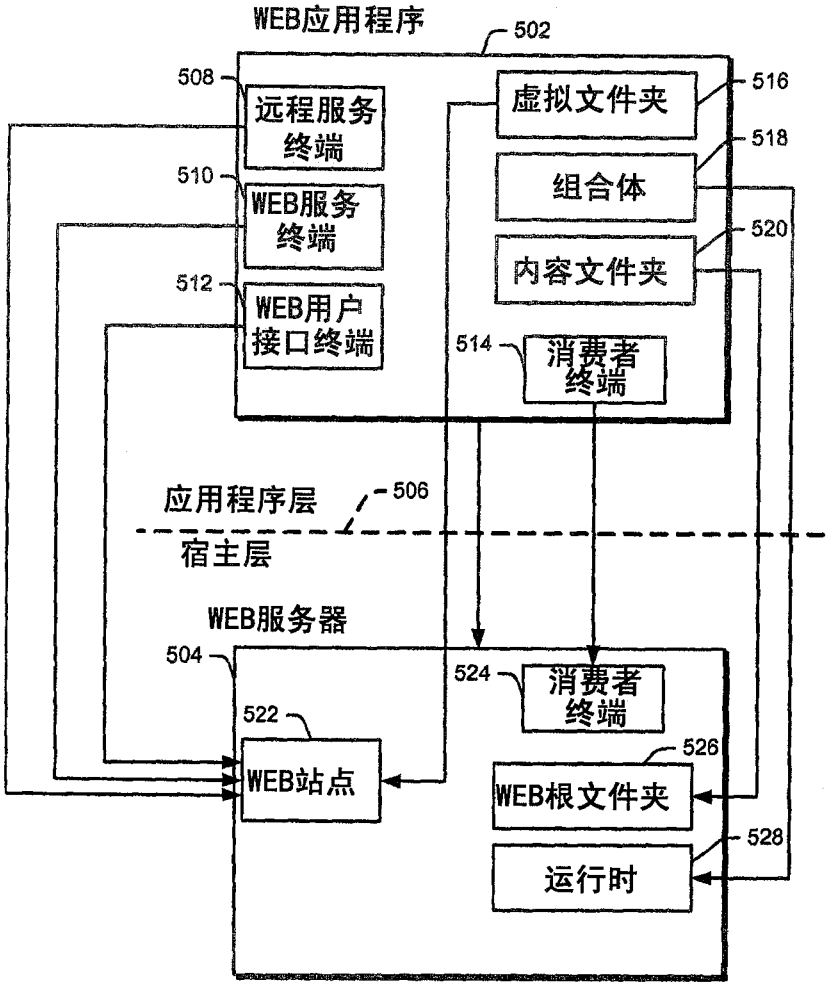

WEB应用程序
502
508
远程服务终端
510
WEB服务终端
512
WEB用户接口终端
虚拟文件夹
516
518
组合体
520
内容文件夹
514
消费者终端
应用程序层
506
宿主层
WEB服务器
504
524
消费者终端
522
WEB站点
526
WEB根文件夹
528
运行时

图 5

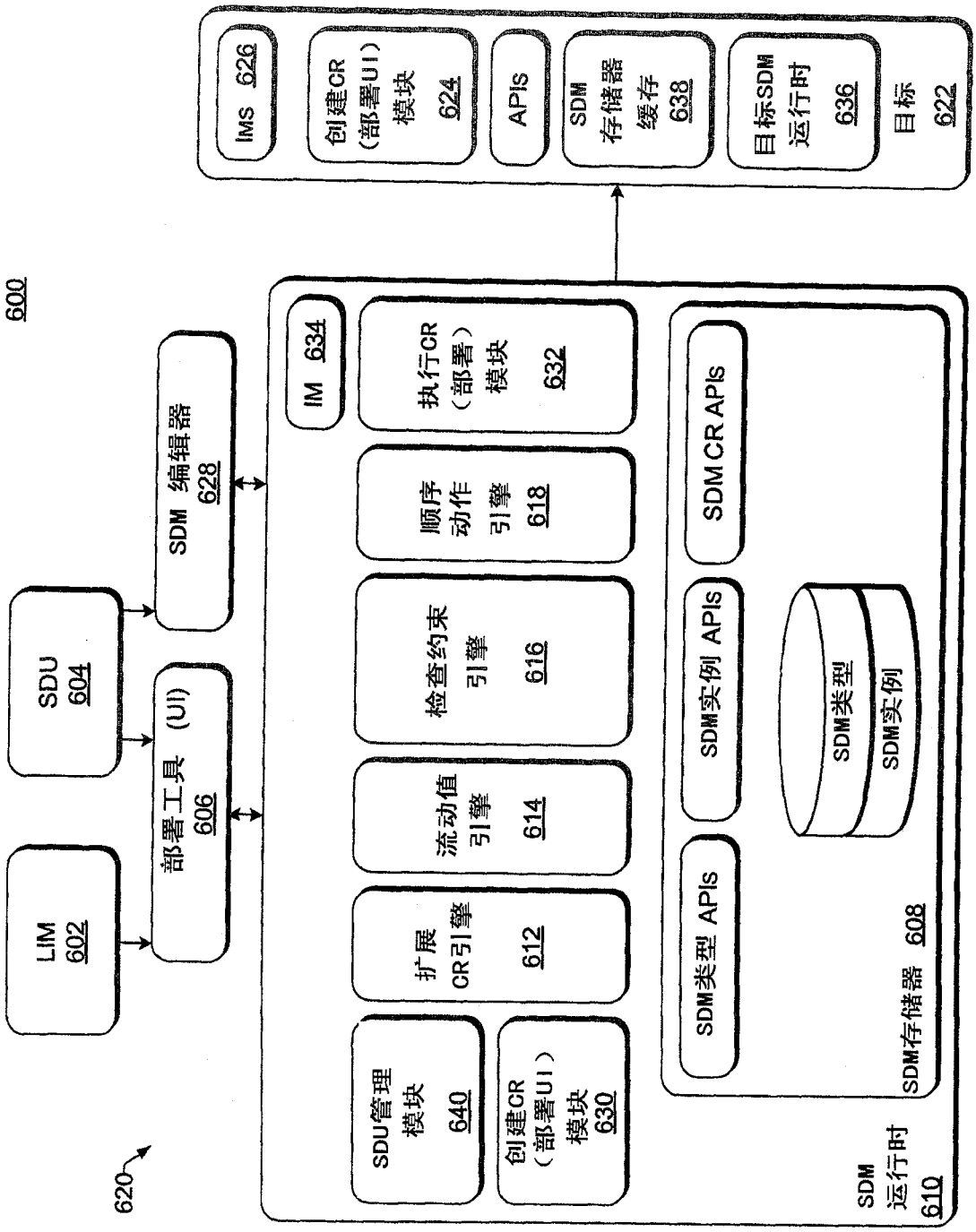

600
620
LIM 602
SDU 604
部署工具 (UI) 606
SDM 编辑器 628
SDM 运行时 610
SDU管理模块 640
创建CR（部署UI）模块 630
扩展CR引擎 612
流动值引擎 614
检查约束引擎 616
顺序动作引擎 618
IM 634
执行CR（部署）模块 632
SDM类型 APIs
SDM实例 APIs
SDM CR APIs
SDM类型
SDM实例
SDM存储器 608
IMS 626
创建CR（部署UI）模块 624
APIs
SDM存储器缓存 638
目标SDM运行时 636
目标 622

图 6

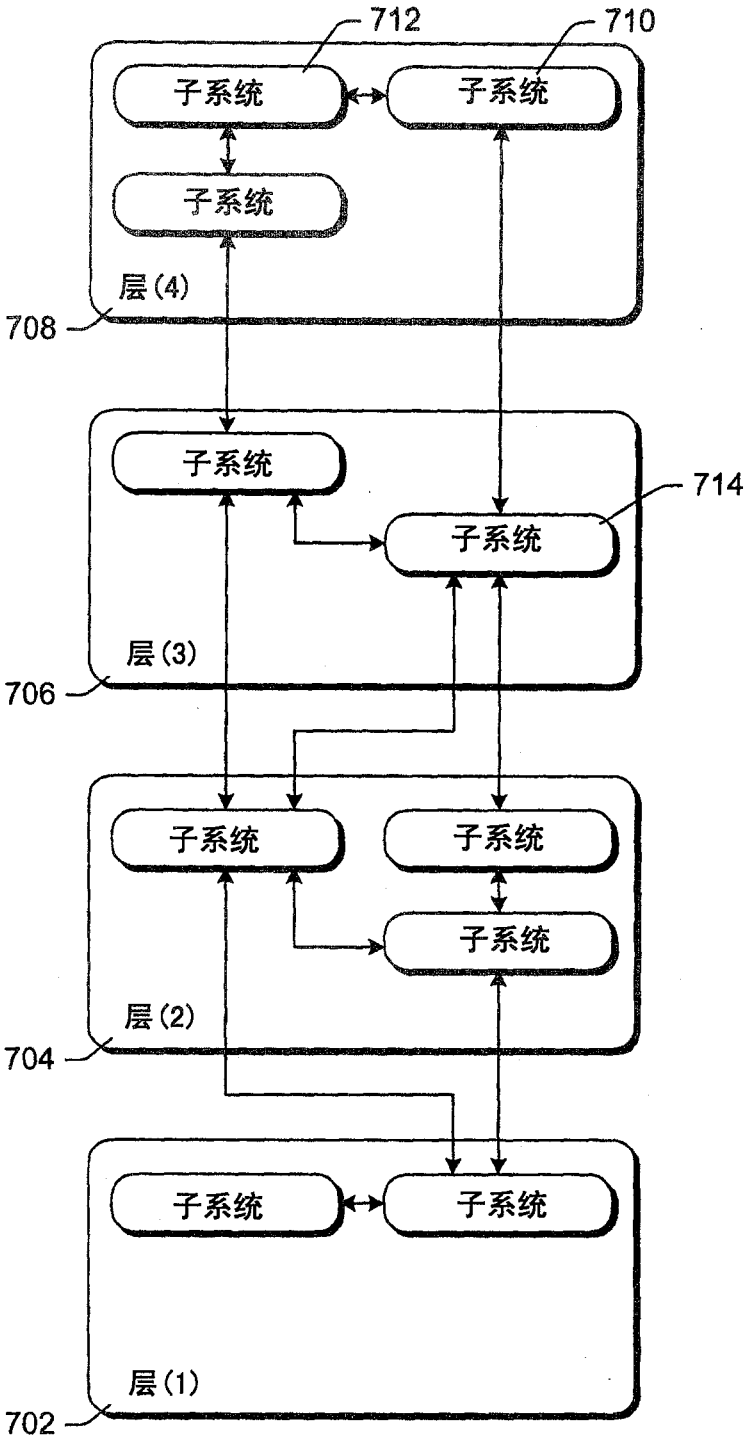

712
710
子系统
子系统
子系统
层(4)
708
子系统
714
子系统
层(3)
706
子系统
子系统
子系统
层(2)
704
子系统
子系统
层(1)
702

图 7

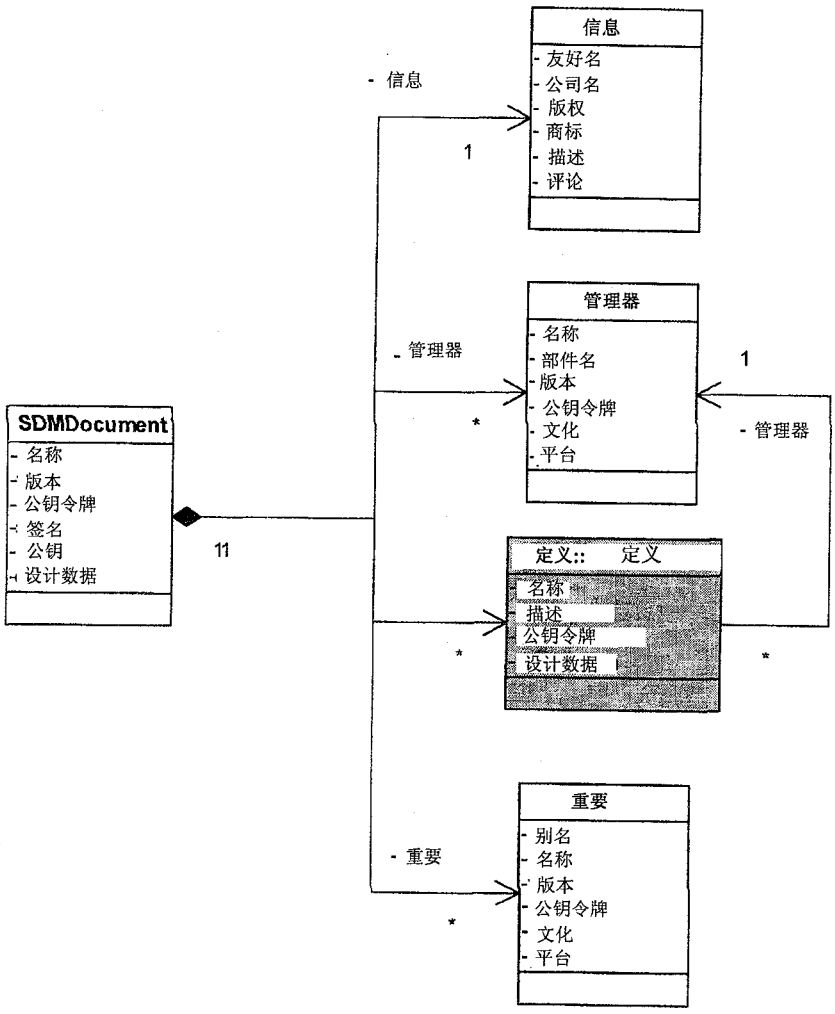

信息
- 友好名
- 公司名
- 版权
- 商标
- 描述
- 评论
- 信息
1
管理器
- 名称
- 部件名
- 版本
- 公钥令牌
- 文化
- 平台
- 管理器
1
- 管理器
*
SDMDocument
- 名称
- 版本
- 公钥令牌
- 签名
- 公钥
- 设计数据
11
定义:: 定义
名称
描述
公钥令牌
设计数据
*
*
重要
- 别名
- 名称
- 版本
- 公钥令牌
- 文化
- 平台
- 重要
*

图 8

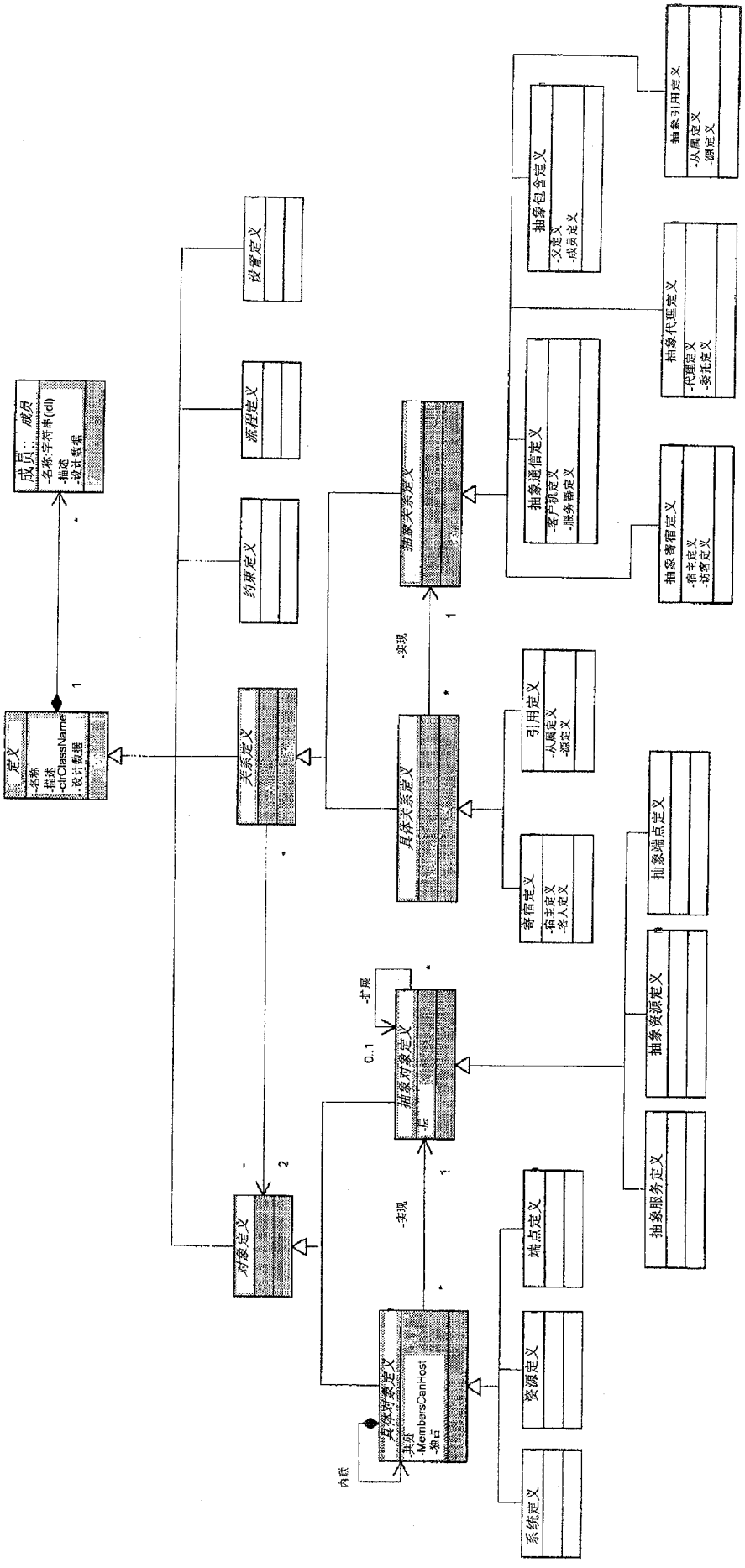

定义
-名称
-描述
-clrClassName
-设计数据
成员::成员
-名称:字符串[dll]
-描述
-设计数据
关系定义
对象定义
抽象对象定义
具体对象定义
-其它
-MembersCanHost
-独占
属性定义
流程定义
约束定义
抽象关系定义
具体关系定义
抽象通信定义
-客户初定义
-服务器定义
抽象客宿定义
-宿主定义
-访客定义
抽象代理定义
-代理定义
-委托定义
抽象包含定义
-父定义
-成员定义
抽象引用定义
-从属定义
-源定义
引用定义
-从属定义
-源定义
寄宿定义
-宿主定义
-客人定义
抽象端点定义
抽象资源定义
抽象服务定义
端点定义
资源定义
系统定义
-实现
-扩展
0..1
内嵌

图 9

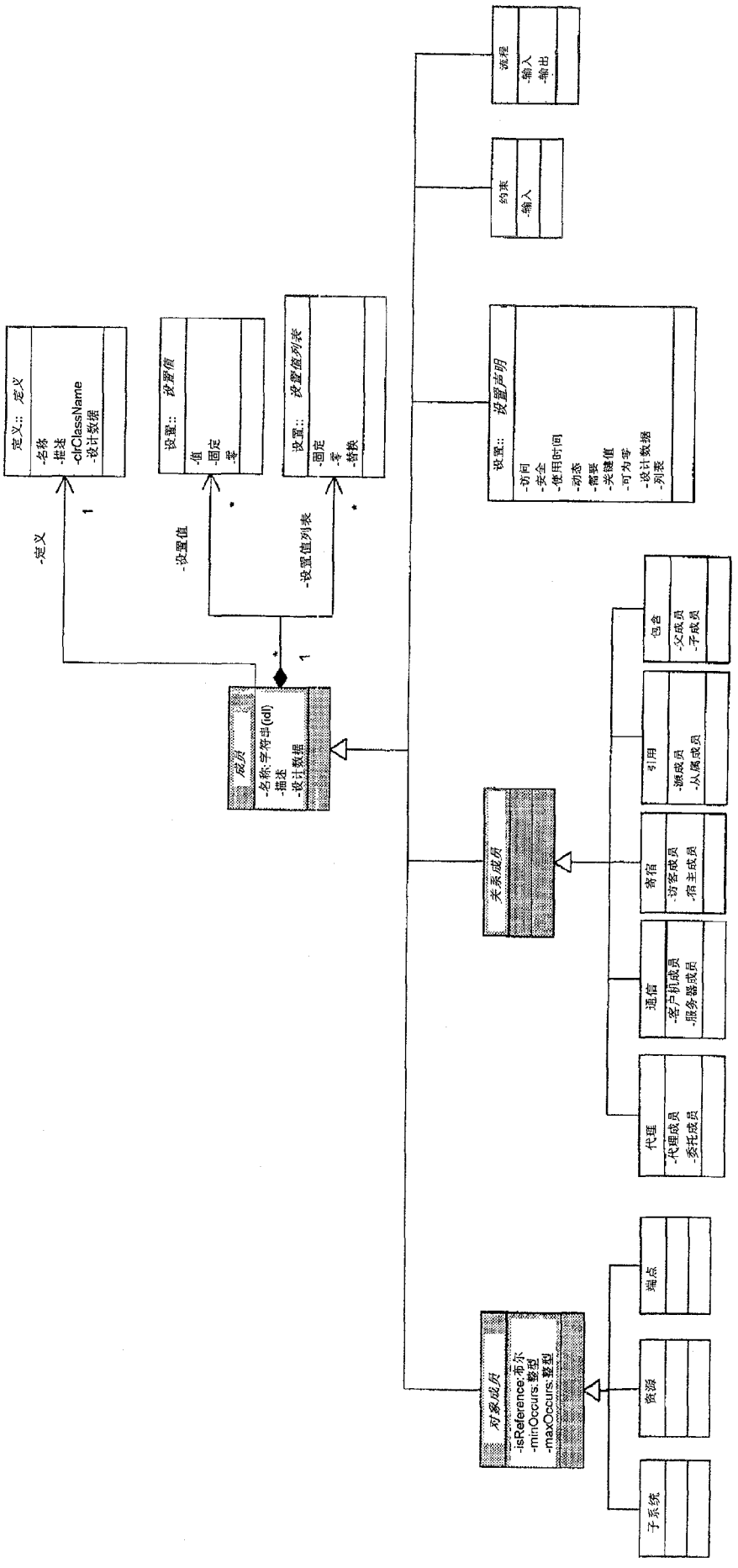

定义::定义
-名称
-描述
-clrClassName
-设计数据
设置::设置值
-值
-固定
-零
设置::设置值列表
-固定
-零
-替换
流程
-输入
-输出
约束
-输入
设置::设置声明
-访问
-安全
-使用时间
-动态
-需要
-关键值
-可为零
-设计数据
-列表
-定义
-设置值
-设置值列表
成员
-名称:字符串(idl)
-描述
-设计数据
关系成员
包含
-父成员
-子成员
引用
-源成员
-从属成员
寄宿
-访客成员
-宿主成员
通信
-客户机成员
-服务器成员
代理
-代理成员
-委托成员
对象成员
-isReference:布尔
-minOccurs:整型
-maxOccurs:整型
端点
资源
子系统

图 10

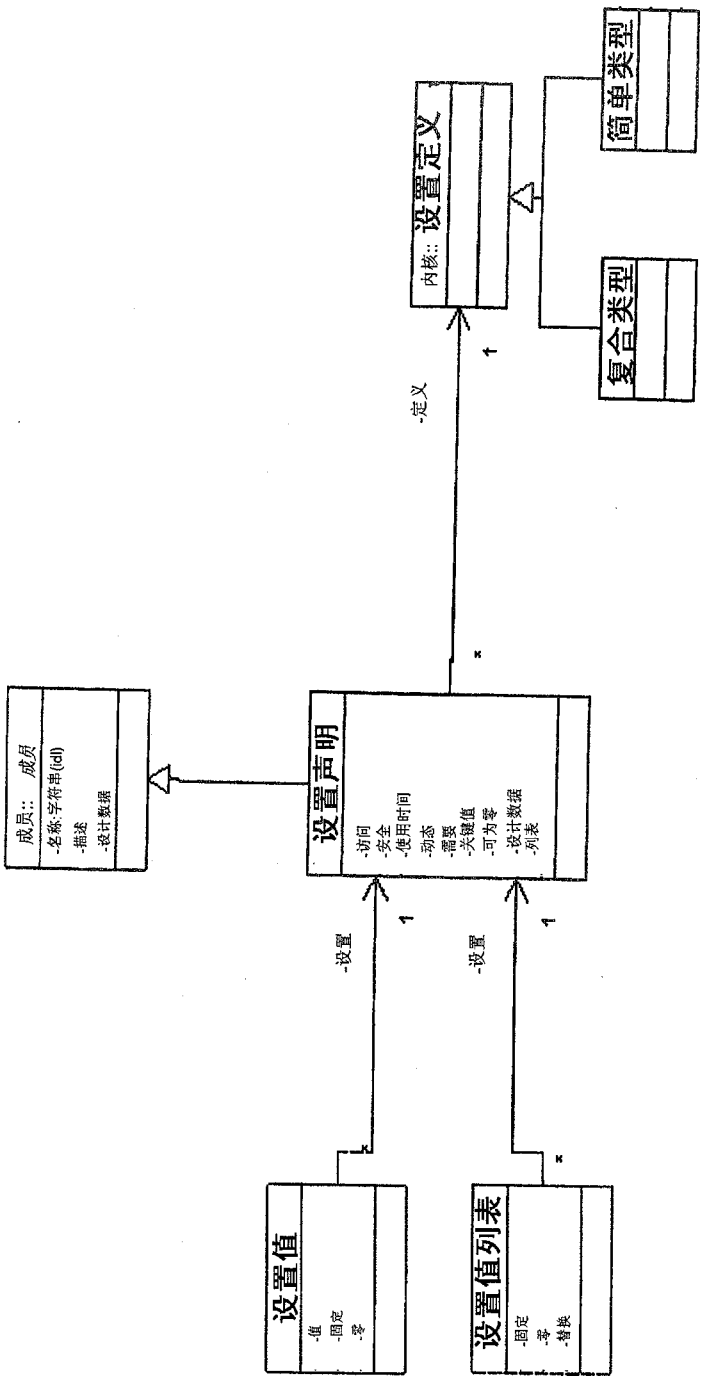

简单类型
内核::设置定义
复合类型
-定义
1
*
成员::成员
-名称:字符串(idl)
-描述
-设计数据
设置声明
-访问
-安全
-使用时间
-动态
-需要
-关键值
-可为零
-设计数据
-列表
-设置
1
-设置
1
设置值
-值
-固定
-零
设置值列表
-固定
-零
-替换
*

图 11

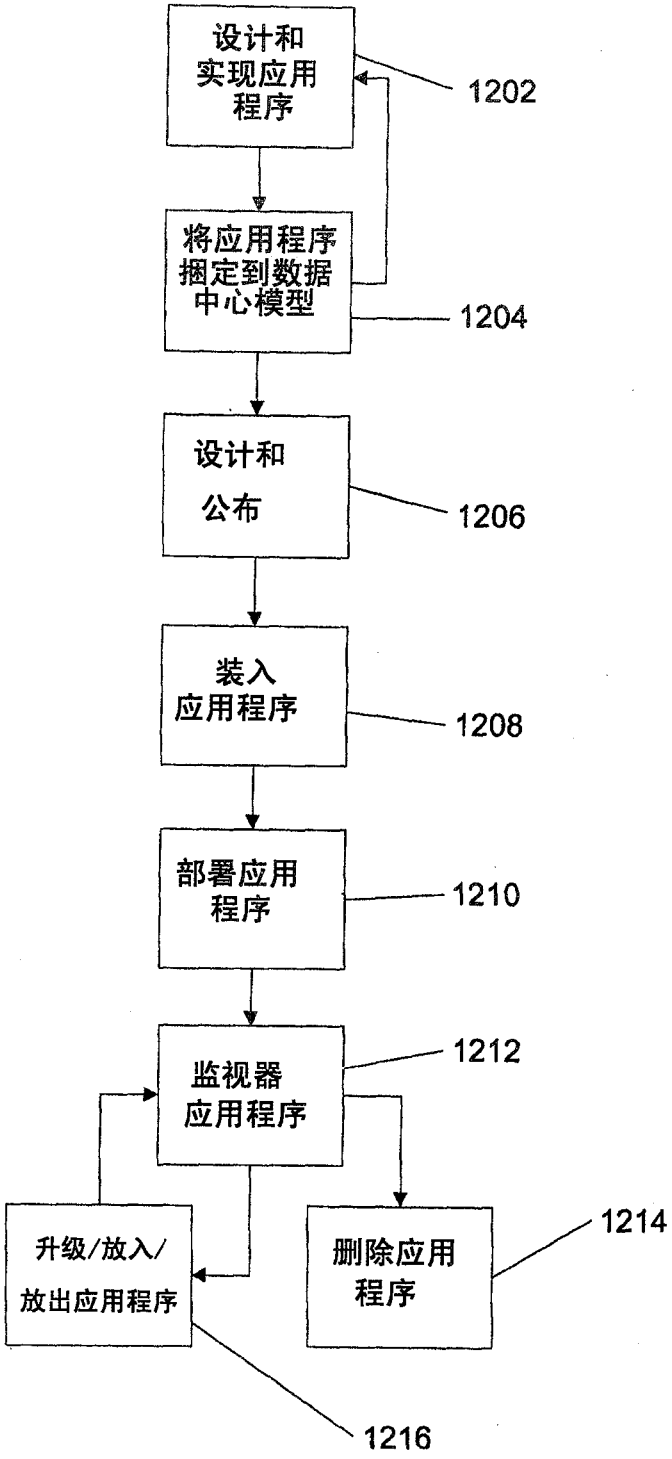

设计和
实现应用
程序
1202
将应用程序
捆定到数据
中心模型
1204
设计和
公布
1206
装入
应用程序
1208
部署应用
程序
1210
监视器
应用程序
1212
升级/放入/
放出应用程序
删除应用
程序
1214
1216

图 12

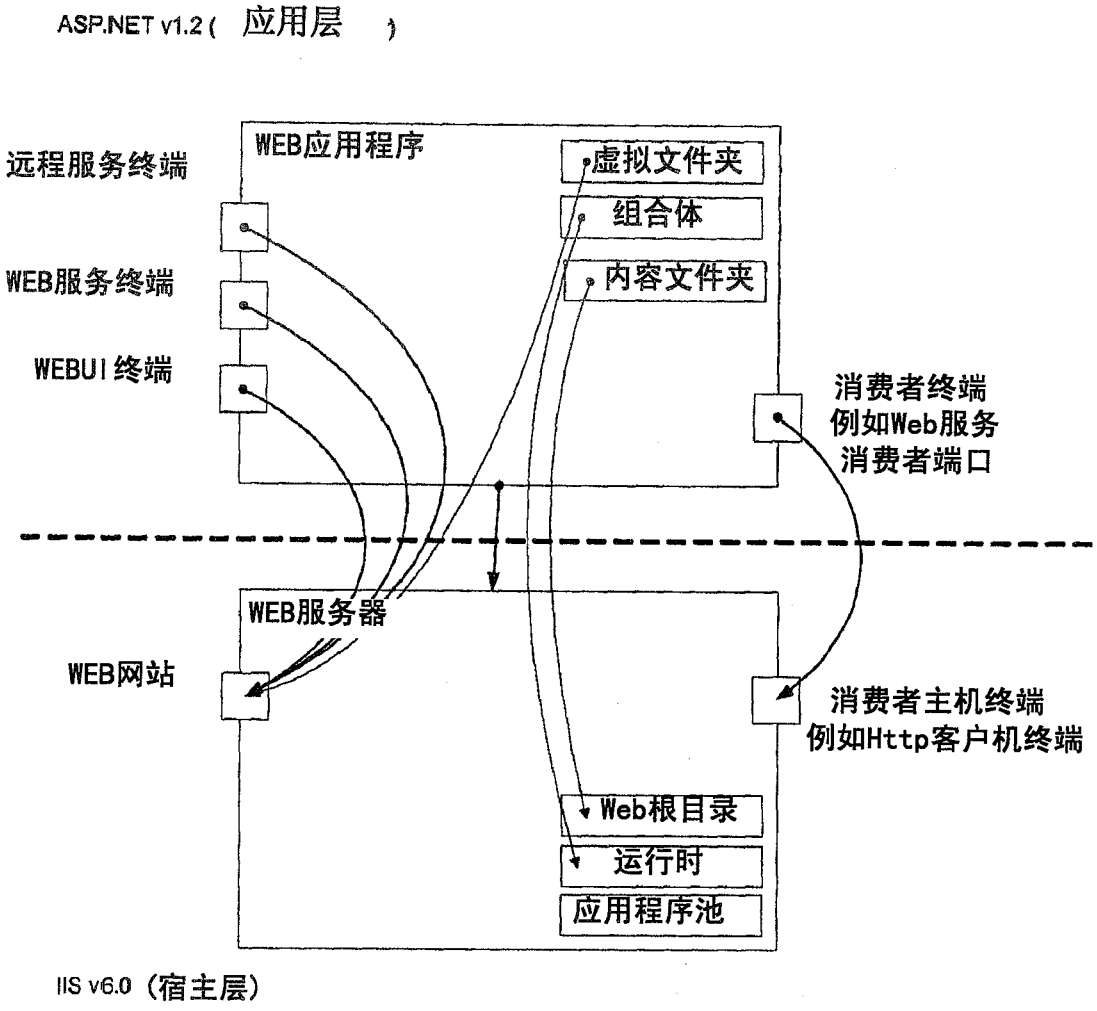

ASP.NET v1.2 (应用层)
远程服务终端
WEB服务终端
WEBUI终端
WEB应用程序
虚拟文件夹
组合体
内容文件夹
消费者终端
例如Web服务
消费者端口
WEB服务器
WEB网站
消费者主机终端
例如Http客户机终端
Web根目录
运行时
应用程序池
IIS v6.0 (宿主层)

图 13

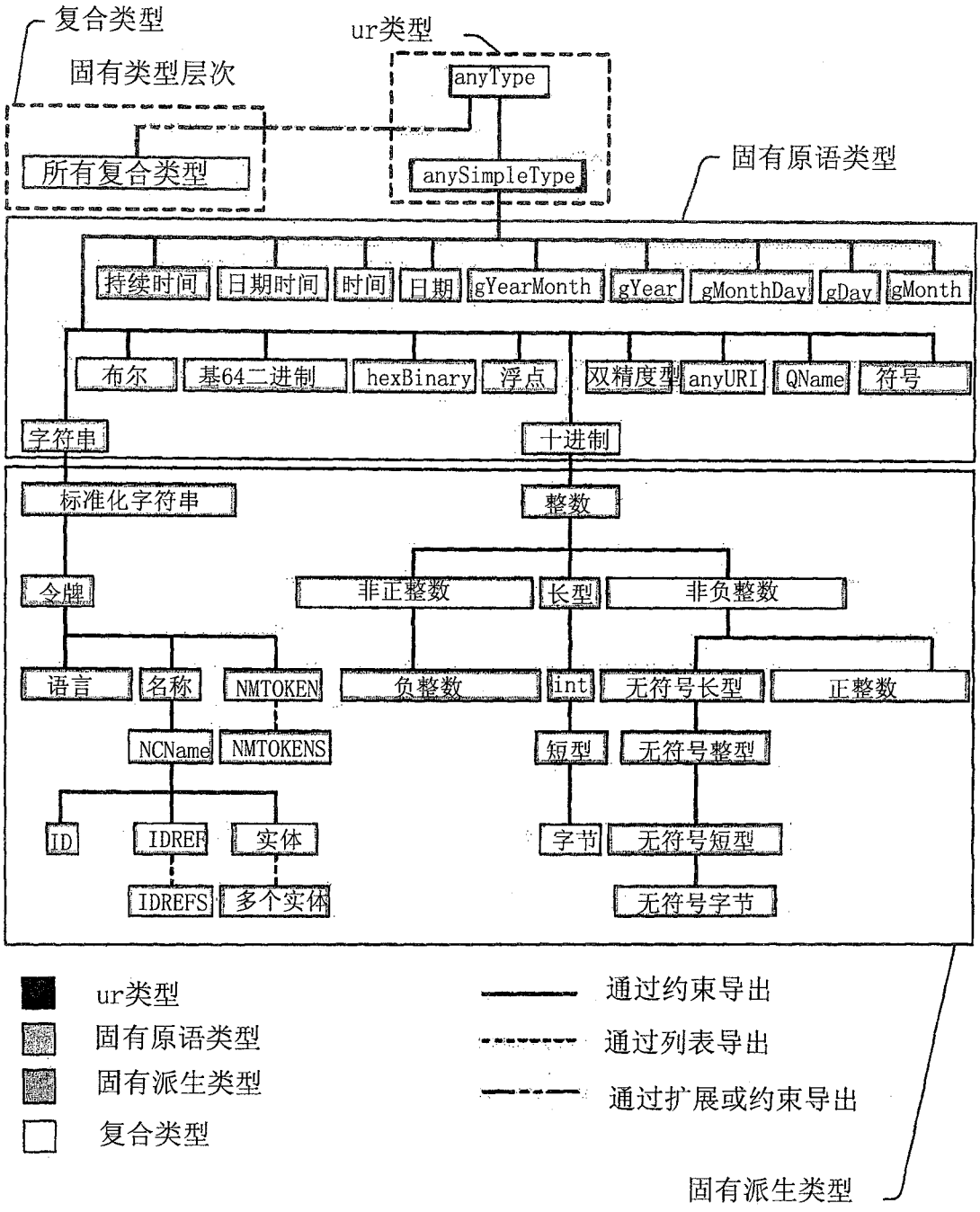

复合类型
固有类型层次
所有复合类型
ur类型
anyType
anySimpleType
固有原语类型
持续时间
日期时间
时间
日期
gYearMonth
gYear
gMonthDay
gDay
gMonth
布尔
基64二进制
hexBinary
浮点
双精度型
anyURI
QName
符号
字符串
十进制
标准化字符串
整数
令牌
非正整数
长型
非负整数
语言
名称
NMTOKEN
负整数
int
无符号长型
正整数
NCName
NMTOKENS
短型
无符号整型
ID
IDREF
实体
字节
无符号短型
IDREFS
多个实体
无符号字节
ur类型
固有原语类型
固有派生类型
复合类型
通过约束导出
通过列表导出
通过扩展或约束导出
固有派生类型

图 14

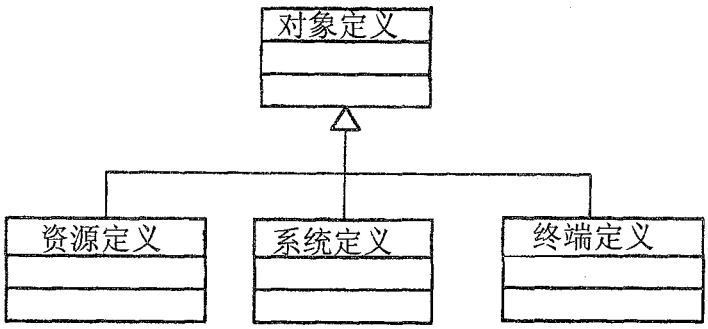


图 15

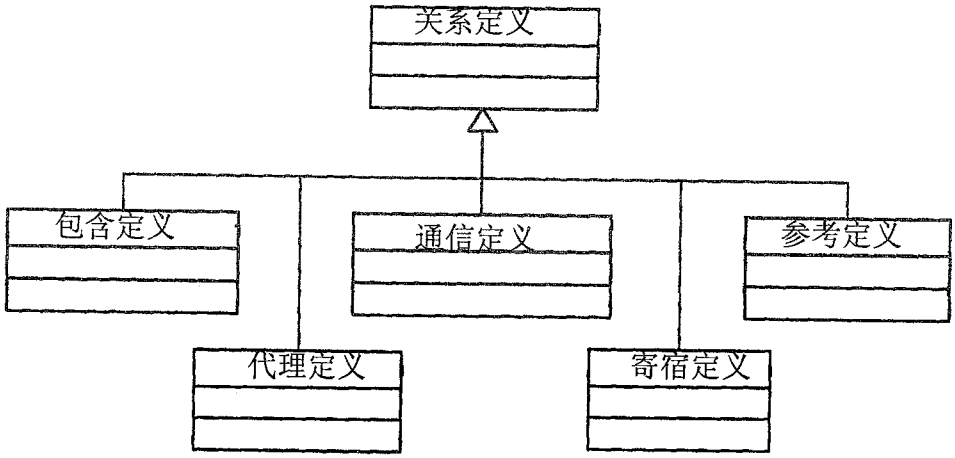


图 16

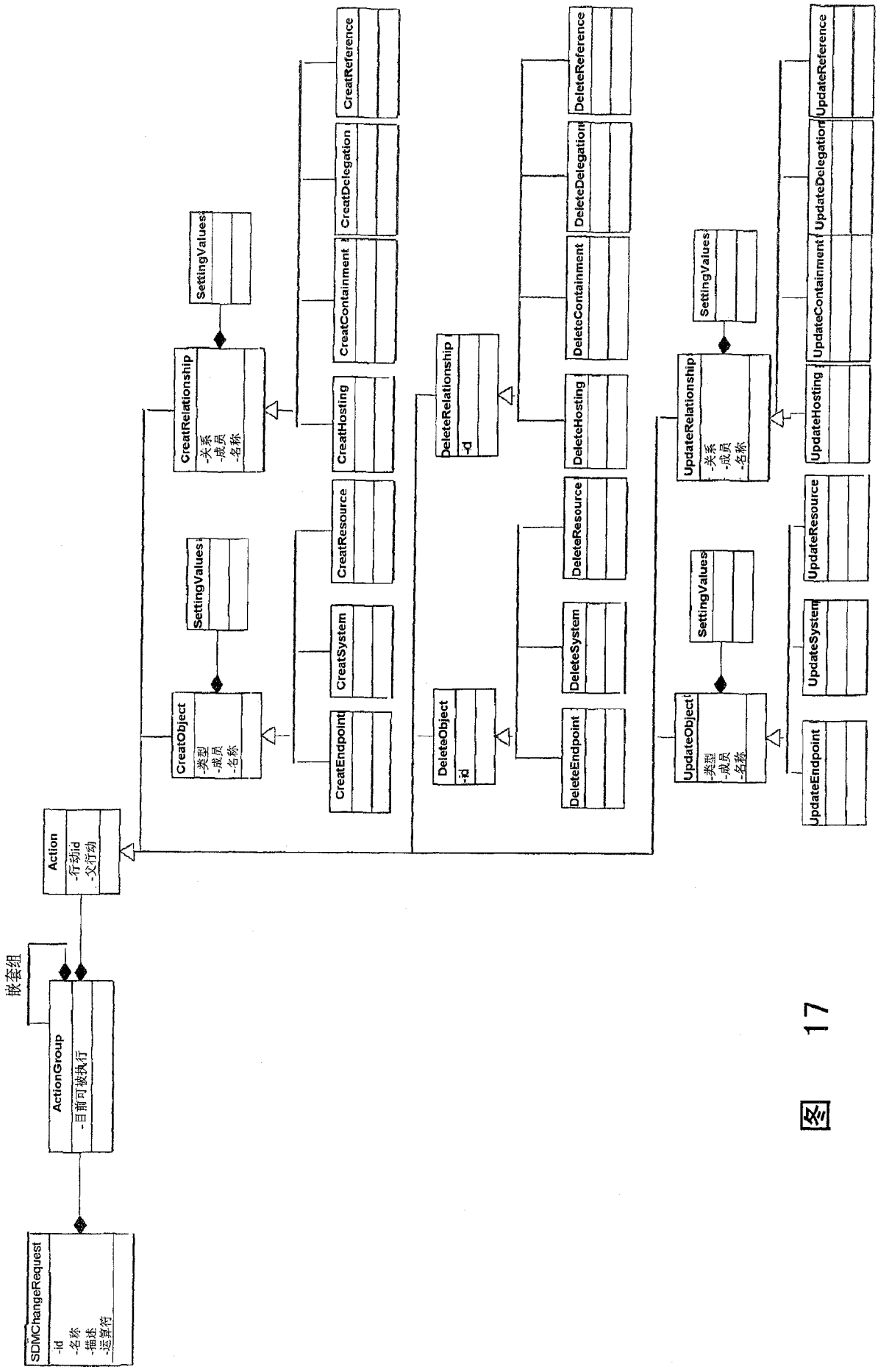

SDMChangeRequest
-id
-名称
-描述
-运算符
ActionGroup
-目前可被执行
嵌套组
Action
-行动id
-父行动
CreatObject
-类型
-成员
-名称
SettingValues
CreatRelationship
-关系
-成员
-名称
SettingValues
CreatEndpoint
CreatSystem
CreatResource
CreatHosting
CreatContainment
CreatDelegation
CreatReference
DeleteObject
-id
DeleteRelationship
-id
DeleteEndpoint
DeleteSystem
DeleteResource
DeleteHosting
DeleteContainment
DeleteDelegation
DeleteReference
UpdateObject
-类型
-成员
-名称
SettingValues
UpdateRelationship
-关系
-成员
-名称
SettingValues
UpdateEndpoint
UpdateSystem
UpdateResource
UpdateHosting
UpdateContainment
UpdateDelegation
UpdateReference

图 17

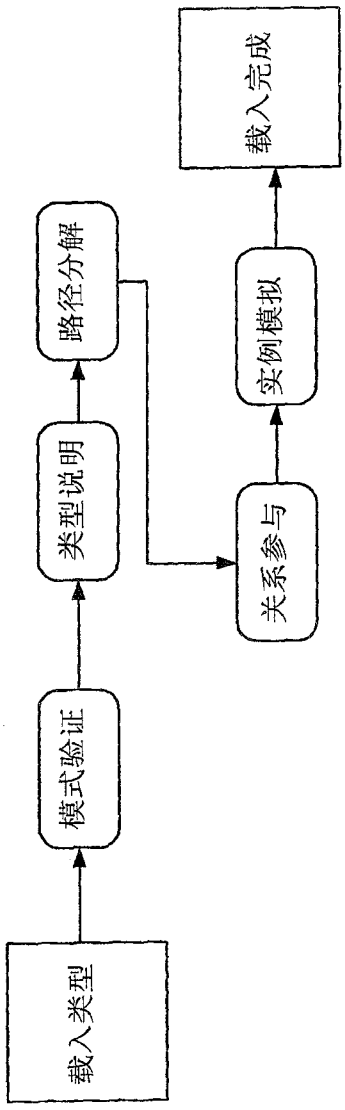


图 18

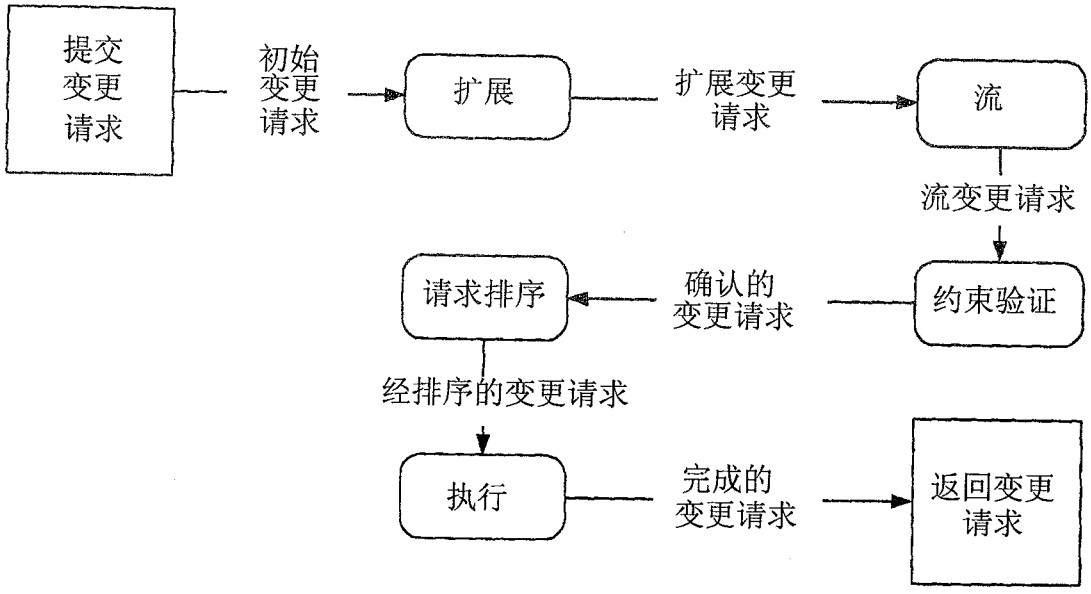

提交
变更
请求
初始
变更
请求
扩展
扩展变更
请求
流
流变更请求
约束验证
确认的
变更请求
请求排序
经排序的变更请求
执行
完成的
变更请求
返回变更
请求

图 19

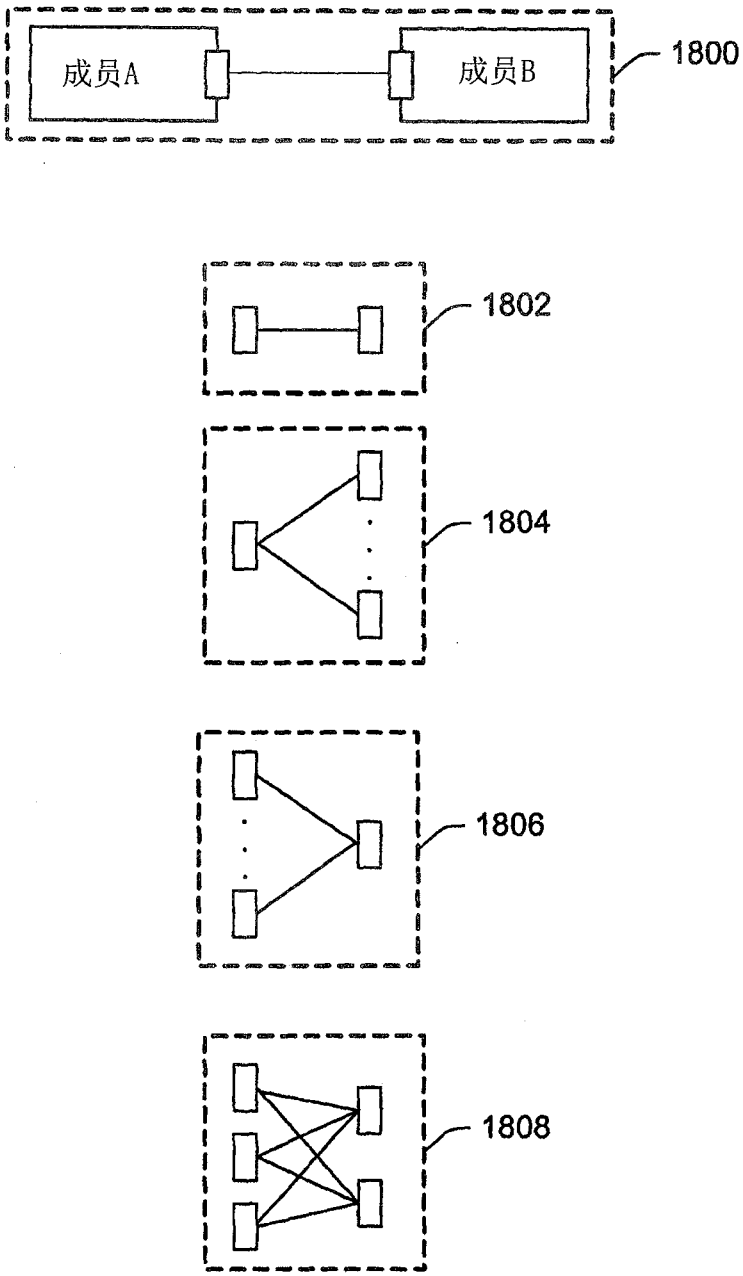


图 20

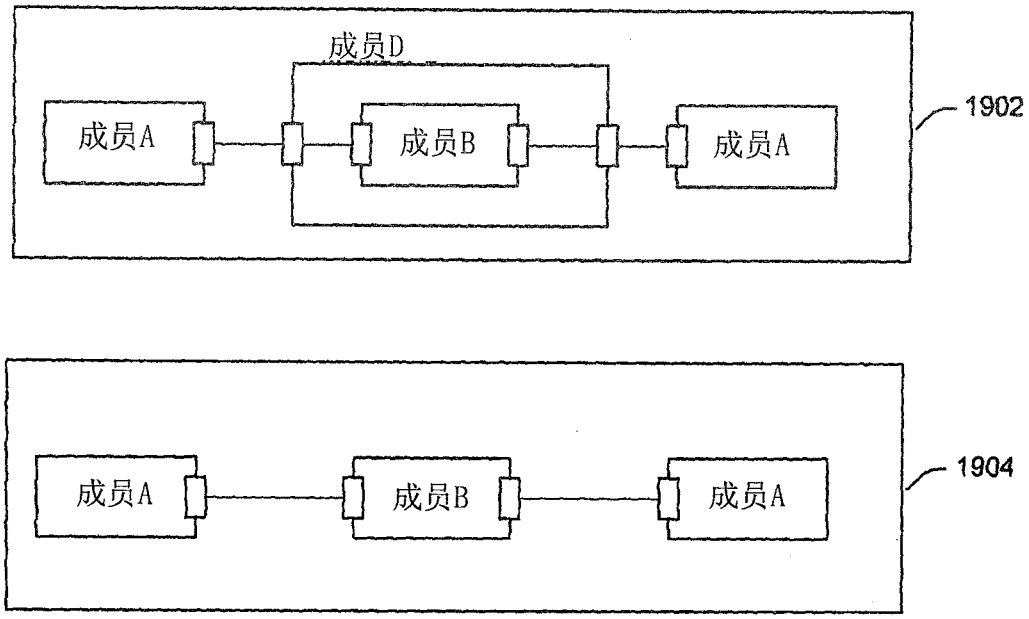


图 21

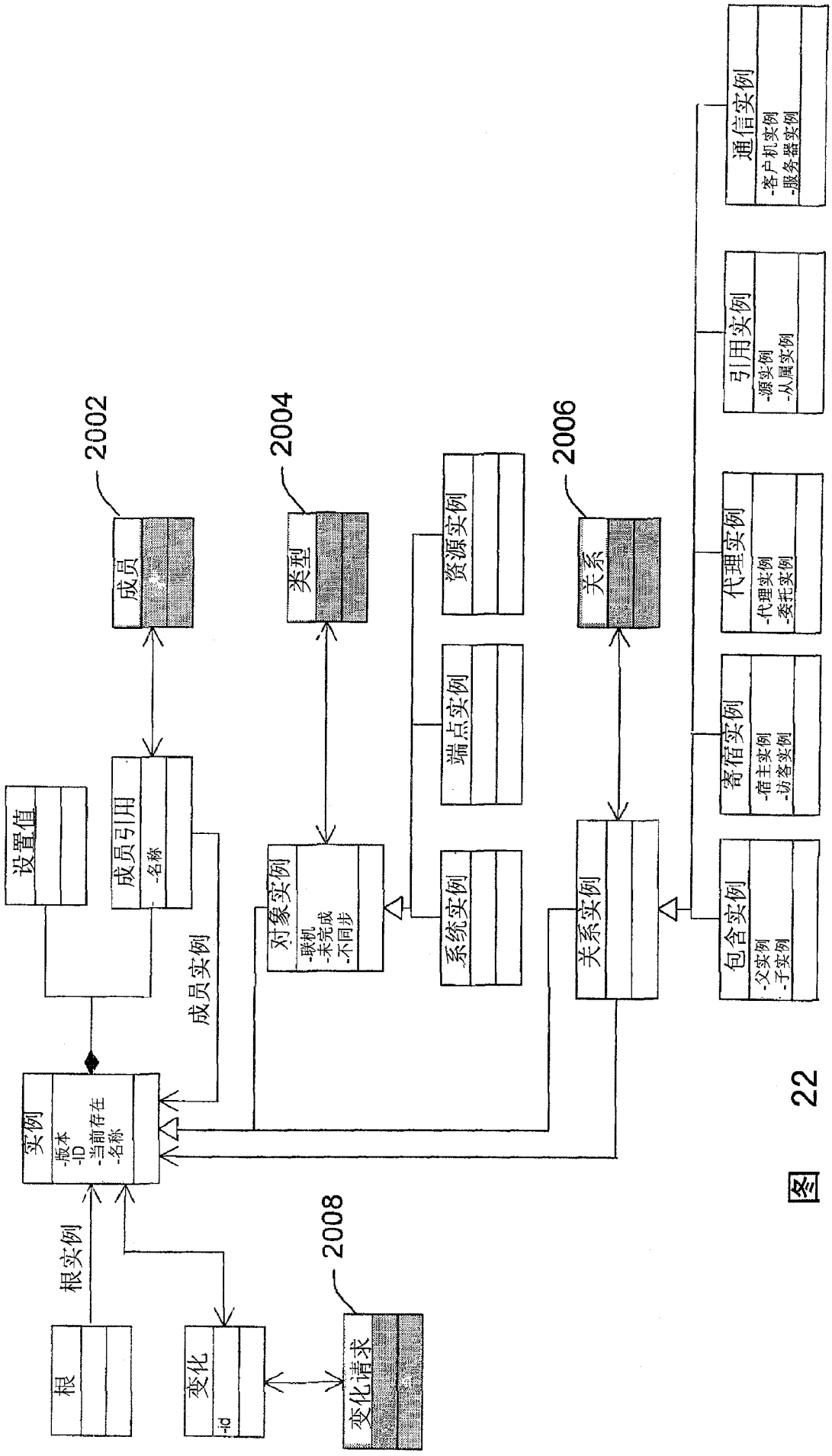

2002
2004
2006
2008
成员
类型
关系
变化请求
成员引用
-名称
设置值
成员实例
对象实例
-联机
-未完成
-不同步
资源实例
端点实例
系统实例
关系实例
通信实例
-客户机实例
-服务器实例
引用实例
-源实例
-从属实例
代理实例
-代理实例
-委托实例
寄宿实例
-宿主实例
-访客实例
包含实例
-父实例
-子实例
实例
-版本
-ID
-当前存在
-名称
根实例
根
变化
-id

图 22

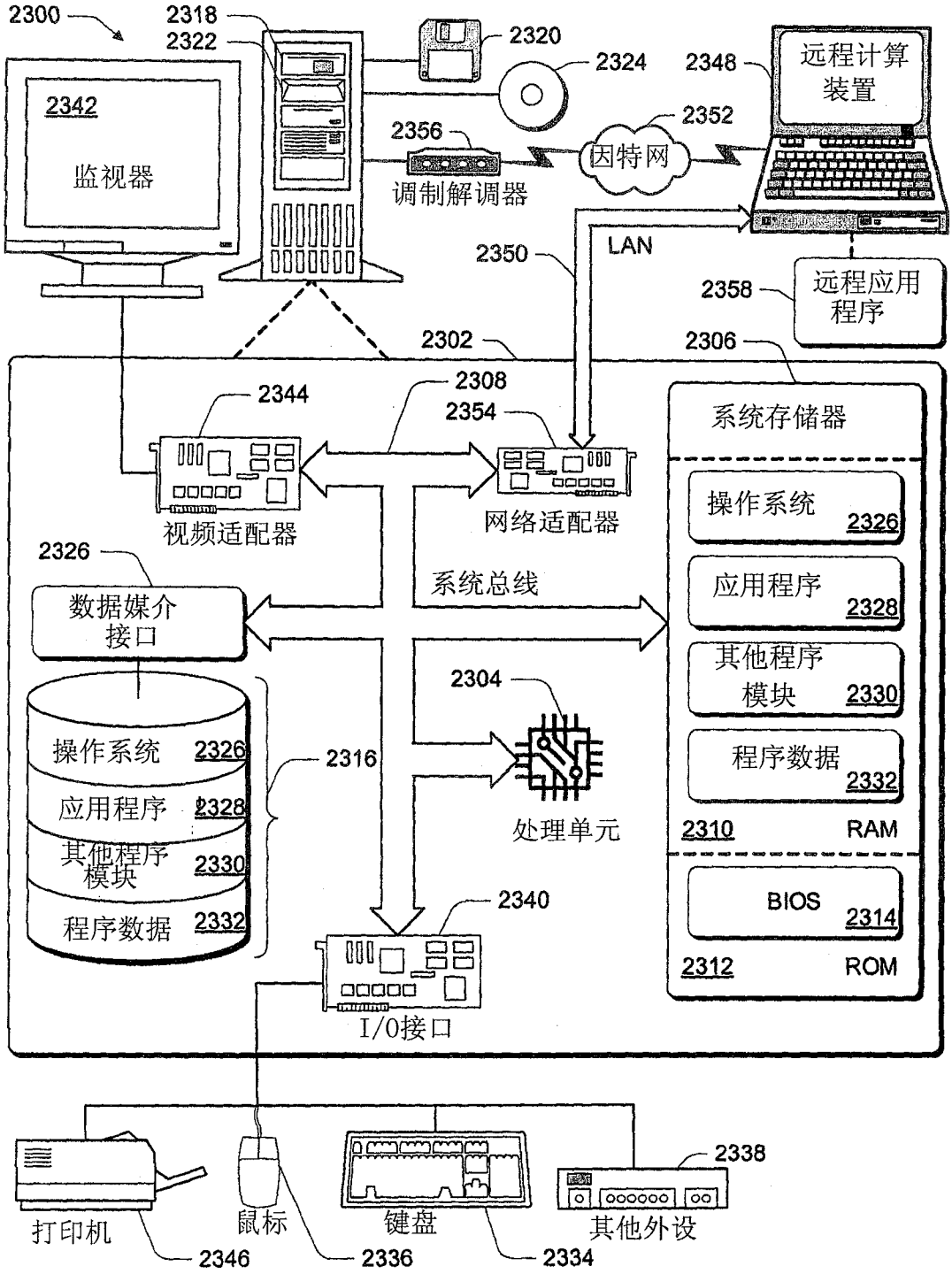

2300
2318
2322
2320
2324
2348
远程计算装置
2342
监视器
2356
2352
因特网
调制解调器
2350
LAN
2358
远程应用程序
2302
2306
2344
2308
2354
系统存储器
视频适配器
网络适配器
操作系统 2326
2326
应用程序 2328
系统总线
数据媒介接口
其他程序模块 2330
2304
程序数据 2332
2316
操作系统 2326
应用程序 2328
其他程序模块 2330
程序数据 2332
处理单元
2310 RAM
BIOS 2314
2340
2312 ROM
I/O接口
2338
打印机
鼠标
键盘
其他外设
2346
2336
2334

图 23