[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 17/30 (2006.01)

[12] 发明专利说明书

专利号 ZL 200510131641.1

[45] 授权公告日 2009 年 7 月 1 日

[11] 授权公告号 CN 100507913C

[22] 申请日 2005.12.12

[21] 申请号 200510131641.1

[73] 专利权人 北京书生国际信息技术有限公司

地址 100083 北京市海淀区学院路 35 号世宁大厦 1301 室

[72] 发明人 王东临 郭 旭 刘昌伟 邹开红 陆小青

[74] 专利代理机构 北京银龙知识产权代理有限公司

代理人 郝庆芬

[56] 参考文献

CN1455901A 2003.11.12

CN1363069A 2002.8.7

CN1647035A 2005.7.27

WO00/20985A1 2000.4.13

审查员 杨 薇

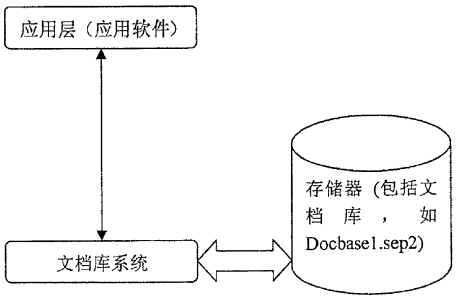
权利要求书 3 页 说明书 58 页 附图 12 页

[54] 发明名称

一种文档处理方法及系统

[57] 摘要

本发明提供一种文档系统中在页面上实现层的方法和系统，所述文档系统包括用于存储文档的存储器和应用软件系统，所述方法包括，将文档中的页的数据分成组；所述每一组数据都能够被文档系统实施不同的管理和控制。 所述被分成组的数据按先后顺序排列，所述每一组数据能够被以层的方式进行管理和控制，包括单独签名和层叠显示等。 应用上述技术方案，在文档库中实现了多层操作，使文档的操作更加灵活。



1. 一种文档处理方法，所述方法包括：

1）根据创建文档的指令，创建包含页的文档数据，至少一页所述文档数据分成多个组，所述被分成组的数据按先后顺序排列；

2）根据操作指令，对所述文档数据中的所述分成多个组的数据的至少一组数据执行操作；

所述对所述文档数据中的所述分成多个组的数据的至少一组数据执行操作的步骤由文档库系统和应用软件执行，其中，文档库系统和应用软件通过一种标准调用接口连接，

所述应用软件向所述文档库系统发出操作指令；

所述文档库系统按照所述操作指令，对文档数据中相应组的数据执行相应的操作。

2. 根据权利要求 1 所述的文档处理方法，其特征在于，所述至少一组数据由版面对象组成，所述版面对象包括状态、文字、图形、图像、语义信息、源文件、脚本、插件、嵌入式对象、书签、链接、流媒体和二进制数据流。

3. 根据权利要求 2 所述的文档处理方法，其特征在于，一个或多个版面对象组成一个对象组。

4. 根据权利要求 3 所述的文档处理方法，其特征在于，所述对象组还包括子对象组，所述子对象组由文字、图形、图像、流媒体、插件、脚本、二进制数据流和/或超链接组成。

5. 根据权利要求 3 所述的文档处理方法，其特征在于，同一组数据的不同对象组用不同的页面描述语言来定义。

6. 根据权利要求 1 所述的文档处理方法，其特征在于，所述操作包括对所述至少一组数据单独进行签名。

7. 根据权利要求 6 所述的文档处理方法，其特征在于，所述操作包括根据不同的组数据的内容由不同的角色进行签名。

8. 根据权利要求 6 所述的文档处理方法，其特征在于，所述操作包括新增加一组数据或修改不属于所述单独进行签名的数据的一组数据，并且所

述新增或修改操作不破坏所述签名。

9. 根据权利要求 1 所述的文档处理方法，其特征在于，不同的组数据的内容由不同的软件来设置。

10. 根据权利要求 1 所述的文档处理方法，其特征在于，当需要显示页面时，单独控制每一组数据的显示方式，所述显示方式包括显示、不显示、叠加和/或水印效果。

11. 根据权利要求 10 所述的文档处理方法，其特征在于，当某一组数据被显示时，则其前面的所有组数据都会同时显示，显示方式都为叠加显示。

12.根据权利要求 1 所述的文档处理方法，其特征在于，所述应用软件向文档库系统发出的指令是表示对预定义的文档模型所描述的信息进行的操作，该预定义文档模型与文档数据的存储形式无关。

13. 根据权利要求 12 所述的文档处理方法，其特征在于，所述预定义的文档模型所描述的信息为至少一个对象构成的树状结构。

14. 根据权利要求 12 所述的文档处理方法，其特征在于，所述预定义的文档模型能够描述页面上的任意可见内容。

15. 一种文档系统，包括：用于存储文档数据的存储器、文档库系统和应用软件；

所述存储器用于存储包含页且至少一页分成多个组的文档数据，所述被分成组的数据按先后顺序排列；

所述文档库系统和应用软件用于操作所述文档数据中的所述分成多个组的数据的至少一组数据；

其中，文档库系统和应用软件通过一种标准调用接口连接，

所述应用软件向所述文档库系统发出操作指令；

所述文档库系统按照所述操作指令，对文档数据中相应组的数据执行相应的操作。

16. 根据权利要求 15 所述的文档系统，其特征在于，所述至少一组数据由版面对象组成，所述版面对象包括状态、文字、图形、图像、语义信息、源文件、脚本、插件、嵌入式对象、书签、链接、流媒体和二进制数据流。

17. 根据权利要求 16 所述的文档系统，其特征在于，一个或多个版面对

象组成一个对象组。

18. 根据权利要求 17 所述的文档系统，其特征在于，所述对象组还可以包括子对象组，所述子对象组由文字、图形、图像、流媒体、插件、脚本、二进制数据流和/或超链接组成。

19. 根据权利要求 18 所述的文档系统，其特征在于，同一组数据的不同对象组用不同的页面描述语言来定义。

20. 根据权利要求 15 所述的文档系统，其特征在于，所述操作包括对所述至少一组数据单独进行签名。

21. 根据权利要求 20 所述的文档系统，其特征在于，所述文档库系统和应用软件能够根据不同组数据的内容由不同的角色进行签名。

22. 根据权利要求 20 所述的文档系统，其特征在于，所述操作包括新增加一组数据或修改不属于所述单独进行签名的数据的一组数据，并且所述新增或修改操作不破坏所述签名。

23. 根据权利要求 15 所述的文档系统，其特征在于，不同组数据的内容由不同的文档库系统和应用软件来设置。

24. 根据权利要求 15 所述的文档系统，其特征在于，所述文档库系统和应用软件能够在显示页面时，单独控制每一组数据的显示方式，所述显示方式包括显示、不显示、叠加和/或水印效果。

25. 根据权利要求 24 所述的文档系统，其特征在于，当某一组数据被显示时，则其前面的所有组数据都会同时显示，显示方式都为叠加显示。

26.根据权利要求 15 所述的文档系统，其特征在于，所述应用软件向文档库系统发出的指令是表示对预定义的文档模型所描述的信息进行的操作，该预定义文档模型与文档数据的存储形式无关。

27. 根据权利要求 26 所述的文档系统，其特征在于，所述预定义的文档模型所描述的信息为至少一个对象构成的树状结构。

28. 根据权利要求 26 所述的文档系统，其特征在于，所述预定义的文档模型能够描述页面上的任意可见内容。

一种文档处理方法及系统

技术领域

本发明涉及一种对文档进行处理的系统和方法，特别涉及一种文档处理系统和文档处理方法，该系统和方法能处理多个文档构成的文档库，还包括对页中的各层进行处理，并能使不同应用软件对同一文档进行互操作。

背景技术

目前关于各种非结构化文档的软件已经比较普及，形成了多种文档格式林立的状况。例如，一个内容管理软件往往要处理二三百种文档格式，而且这些格式还在不断更新，给软件开发商带来了巨大的困难。如何解决文档通用性、进行数字内容提取、格式兼容越来越成为人们的关注点，人们迫切希望解决以下问题：

1）文档不通用

基本上只能用同一种软件在不同的人之间交换文档，但不能在不同的软件之间互相交换文档，形成信息封闭。

2）文档信息提取困难

文档描述信息丰富，数据结构复杂，实现难度较大。每一家公司都把自己的书面文档描述作为独家特有技术、基本上不提供开放接口。

3）访问接口不统一、数据兼容困难或代价太高

不同的文档处理软件之间，文件格式互不兼容，在处理过程中要么利用对方组件解析（前提是对方提供相应接口），要么自己投入研发力量从头到尾的解析对方的格式。

4）信息安全较差

目前针对书面文档的权限控制手段单一，主要是数据加密、口令认证。因为信息泄露，每年造成巨大损失的公司案例层出不穷。

5）都是针对单个文档的处理，缺乏多文档管理手段

每个人电脑中都有大量文档，但多个文档之间缺乏有效的组织管理，而

且资源共享很难。如，字库/字体文件、全文数据检索等。

6）行业竞争层次还停留在各自格式描述之争上

由于书面文档数据结构复杂、数据描述丰富、文档数据长度不确定，每一个文档都千差万别。长期以来，大家都在关注文档格式标准，各大公司都努力将自己特有的文档格式发展为市场标准，各标准组织也致力于制订通用的文档格式标准。但不管是专有的文档格式（如.doc）还是开放的文档格式（如 PDF），只要是以文档格式为标准，就不可避免产生以下问题：

a）重复开发，效果不统一

使用同一标准的不同软件都需要自己去解释、生成该格式的文档，造成大量重复开发，而且会因为各家解释程序不同，有的完善有的相对简单，有的支持新版本有的只支持旧版本数据，同一文档在不同软件下显现出不同的版式，甚至出现解释错误无法打开。

b）阻碍创新

软件是不断创新的行业，但由于每增加一个新功能就需要增加描述该功能的信息，但只有等到标准修订的时候才能增加新的格式，因此把存储格式固定死之后，将会妨碍技术创新的竞争。

c）影响性能

对海量信息，需要增加大量的检索信息以提高检索性能，但固定死的存储格式难以增加检索信息。

d）影响可移植性和可伸缩性

在不同的系统环境下，不同的应用需求，可能会有不同的存储要求。例如，存储在硬盘上就需要考虑如何减少磁头寻道的次数以提高性能，而在嵌入式应用中数据都相当于存储在内存中的，就不存在这个问题。事实上，数据库软件也往往都是这样设计的，同一个厂商的数据库软件在不同平台上就可能会使用不同的存储格式。因此，设置文档存储标准将会影响系统的可移植性和可伸缩性。

7）页面分层的技术不完善

目前一些软件，如 Adobe 的 photoshop，Microsoft 的 word，多多少少已经有层的概念，但层的功能还比较单一，管理手段比较简单，不能满足应用

需求。

8）检索手段还不够丰富

随着信息的海量化，用任何一个关键词来搜索都会得到数量庞大的检索结果，全文检索技术基本解决了查全率的问题，但查准率迅速上升为首要问题。现有技术还没有很充分地利用全部信息来解决查准率问题，例如每个文字的字体、字号完全可以用来判断该文字的重要性，但都在检索时被忽略了。

事实上，一种文档格式不管是否开放，最后结果往往都是被特定软件所垄断。商业实践的结果证明，不管是.doc 这种已经被无数的同行研究得比较透、大家都花了巨大的精力和人力物力去兼容的文档格式，还是 PDF 这种完全公开的文档格式，在实际应用中用户还是会选择用原厂商的软件（即 MS Word 和 Adobe Acrobat）来处理，而不太愿意用第三方的软件。一种文档格式被特定软件所垄断会造成信息流不畅通，非常不利于进行信息化建设，而且还会造成用户过分集中到大软件公司的软件上，形成对用户不利的垄断。例如，MS Office 的表格功能不够好，但即使有人开发了非常好用的表格编辑软件，也很难在市场上生存，因为基本上没有哪个文档通篇只有表格，这样用户还只能使用那些功能比较全的软件，尽管其中的表格功能并不好用，因此市场就被 MS Office 这种全能软件所垄断，大量中小软件公司开发的“专而不全”的软件缺乏市场空间。

现有技术中最开放、可交换性最好的是 Adobe Acrobat 采用的 PDF。PDF 已经成为全球分档分发、交换的事实标准，但也只能在不同的人之间交换文档，不能在不同的软件之间交换文档，即不能实现文档的互操作性。而且不管是 Acrobat,还是 Office，都只能对单文档进行处理，缺乏对多文档的管理功能，不具备对文档库进行操作的功能。

在文档信息安全方面，现有技术也存在较多缺陷。Word 和 PDF 这些应用最广泛的文档，都是采用对数据加密或者口令认证等进行数据安全控制，没有提供系统的身份认证机制，对权限的控制都是整个文档范围的，不能细化到文档内的任意区域，对逻辑数据指定加密和签名是受限的，无法对任意逻辑数据设定加密和签名。内容管理系统虽然能够提供很好的身份认证机制，但由于与文档处理系统是分离的，不能在核心层集成，不仅管理粒度只能做

到文档级，而且在文档使用过程中就脱离了内容管理系统的安全控制，难以进行必要的安全管理。通常情况下，安全机制与文档处理是分离的模块，容易出现安全缝隙。

下面介绍本发明中会涉及到的一些现有技术和概念：

非对称密钥加密算法也叫公开密钥体系（Public Key Infrastructure，PKI）算法，由美国斯坦福大学赫尔曼教授于 1977 年提出。它主要指加密密钥和解密密钥不相同，而且相互之间不存在推导关系，用户公开其中一个密钥不会泄漏另一把密钥。这样其他人可以用公钥对发送的信息进行加密，安全地传送到该用户，然后由该用户用自己的私钥进行解密。PKI 技术解决了密钥的发布和管理问题，是目前常用的密码技术。使用 PKI 技术，进行数据通信的双方可以安全地确认对方身份和公开密钥，提供通信的可鉴别性。目前，常用的 PKI 算法有椭圆曲线密码加密算法（Elliptic Curves Cryptography，ECC），RSA 加密算法（Ron Rivest，Adi Shamir，Len Adleman 公私钥算法）等。

RSA 算法描述如下：

公钥：n=pq，（p，q 为两个不同的很大的质数，p 和 q 必须保密）；

将(p-1)和(q-1)相乘得到 $\Phi(n)$；

选择一个整数 e（$1<e<\Phi(n)$）与 $\Phi(n)$互质；

私钥：$d=e^{-1} \bmod \Phi(n)$，即计算一个数字 d，使得它满足公式 $de=1 \bmod \Phi(n)$；

加密：$c=m^c \pmod n$ ；

解密：$m=c^d \pmod n$，m 为明文，c 为密文。

椭圆曲线密码加密算法（ECC）是另一种非对称密钥加密算法，椭圆曲线用于密码算法，于 1985 年由 *Koblitz* 和 *Victor Miller* 分别独立地提出。它问世以来一直是密码分析学的研究对象。现在，在商业和政府的用途中，椭圆曲线密码系统（ECC）都被认为是安全的。根据已知的密码分析学知识，椭圆曲线密码系统相比于传统的密码系统来说提供了更高的安全性。

ECC 加密算法描述如下：

大素数域上的椭圆曲线可通过同构映射将一般的曲线方程变换为特别简单的形式：$y^2=x^3+ax+b$，其中曲线参数 $a,b \in Fp$ 并且满足 $4a^3+27b^2 \neq 0 \pmod$

p)。

因此，满足下列方程的所有点(x, y)，再加上无穷远点 O∞ ，构成一条定义在大素数域 Fp 上的椭圆曲线。

Y2=x3+ax+b (mod p)

其中 x，y 属于 0 到 p-1 间的大素数，并将这条椭圆曲线记为 Ep(a, b)。

考虑如下等式：

K=kG [其中 K，G 为 Ep(a,b)上的点，k 为小于 n（n 是点 G 的阶）的整数，不难发现，给定 k 和 G，根据加法法则，计算 K 很容易；但给定 K 和 G，求 k 就相当困难了。

这就是椭圆曲线密码系统基于的数学难题。把点 G 称为基点（base point），k（k<n，n 为基点 G 的阶）称为私有密钥（private key），K 称为公开密钥（public key）。

加密算法还可以是公知的对称算法，对称算法就是指加密和解密过程均采用同一把密钥。如 AES 算法。

AES 算法是 1997 年 1 月由 NIST 提出的，其目的是开发一种新的能保证政府信息安全的编码算法。最后经过多方评估从 15 种算法中选出 Rijndael 算法作为 AES 编码标准算法。AES 算法是对称加密的迭代分组密码。它把数据块分成比特阵列，每一项密码操作都是面向比特的。Rijndael 算法分为四层，第一层是 8×8 比特置换（即输入 8 比特，输出 8 比特）；第二、三层是线性混合层（阵列的行移位、列混合）；第四层是子密钥与阵列的每比特异或。

AES 的分组长度为 128 比特，密钥长度为 128/192/256 比特，相对应的轮数 r 为 10/12/14，相应的密钥方案为：在加密的过程中，需要 r＋1 个子密钥，需要构造 4（r＋1）个 32 比特字。当种子密钥为 128 和 192 比特时，构造 4（r＋1）个 32 比特字的过程是一样的。但当种子密钥为 256 比特时，构造 4（r＋1）个 32 比特字的过程是不同的。

HASH 也称为散列或消息摘要或数字摘要，就是通过把单向 HASH 函数应用于信息，将任意长度的一块数据转换为一段定长的、不可逆转的数据，称为该数据的 HASH 值。从理论上讲，任何 HASH 算法，产生碰撞（即两块

不同的数据具有相同的 HASH 值）是必然的。HASH 算法的安全性有两层含义：一是由 HASH 值不能反推出原数据；二是要构造两块具有相同 HASH 值的不同的数据在计算上是不可行的，尽管理论上是存在的。目前 MD5、SHA1 和 SHA256 被认为是比较安全的 HASH 算法。另一方面，HASH 函数的计算一般都比较快，相对简单。

并集是指多个集合的所有元素组成的集合。

设 A、B 为两集合，若 A 中任意的元素 x 都属于 B，则称 B 为 A 的超集，称 A 为 B 的子集。

发明内容

本发明的目的在于提供创新的对文档进行处理的文档处理系统和文档处理方法，实现文档的更加丰富、灵活互操作、对多文档的管理、更好的文档安全性以及更好的查询检索。

依照本发明的对文档库进行处理的文档处理系统，其包括文档库系统、存储器、应用软件，其中，文档库的数据存储在存储器中，文档库系统和应用软件通过一种标准调用接口连接起来，该标准调用接口根据预先定义的动作和对象而定义。应用软件对文档的操作都统一成对一种预定义的通用文档模型进行的操作，并通过该标准调用接口向文档库系统发出指令，文档库系统按照应用软件的指令，对存储在存储器中的文档库执行相应的操作。

依照本发明的文档处理方法，该方法包括以下步骤：应用软件通过标准调用接口向文档库系统发出指令，其对文档的操作都统一成对一种预定义的通用文档模型进行的操作；文档库系统按照应用软件的指令，对存储在存储器中的文档库执行相应的操作。其中，文档库系统和应用软件通过上述标准调用接口连接起来，该标准调用接口根据预先定义的动作和对象而定义。

本发明改变了从用户界面到文档存储都由一个软件来完成的现状，将其划分为应用软件和文档库系统两层，并定义了一个接口标准。文档库系统是具备各种文档操作功能的通用技术平台，并具有符合该标准的接口部，应用软件要对文档进行操作时就通过该接口部来向文档库系统发出相应指令，文档库系统根据该指令执行相应操作。这样，只要各应用软件和各文档库系统都遵循同样的标准，不同应用软件就可以通过同一个文档库系统对同一文档

操作，即可实现对文档的互操作。同样，同一个应用软件也可以通过不同文档库系统对不同文档进行操作，而不用分别对每种文档格式都进行单独开发。

本发明还包括一个通用文档模型，该模型能与各应用软件所需要处理的文档相符合。接口标准就是基于该文档模型来设计的，这样才能实现不同的应用软件都可以通过同一个接口部来对文档进行操作。该通用文档模型也适用于各种文档格式，这样同一个应用软件才可以通过同一个接口部来对不同文档格式进行操作。该通用文档模型的具体内容请参见后面的实施例。

本发明还包括一个通用文档安全模型，该通用文档安全模型符合各应用软件对文档安全的需求，使不同的应用软件都可以通过同一个接口部实现对文档的安全控制。该通用文档安全模型的具体内容请参见后面的实施例说明。

接口标准定义了基于该通用文档模型和通用文档安全模型对文档进行操作的各种指令，以及应用软件向文档库系统发送指令的方式。文档库系统具备实现这些指令的功能，以供应用软件调用。接口标准可以用命令串

该通用文档模型还包括由多个文档组成的文档集、文档库和文档仓库等层次，接口标准中也包含对多文档的组织管理、查询检索、安全控制等指令。

该通用模型还包括将页由具有上下顺序的层组成，接口标准中也包含对层的各种操作指令，以及对一个文档某一层所对应源文件的存储和提取。

文档库系统还具备对文档的信息安全管理控制功能，如基于角色的细粒度权限管理，并在接口标准中定义了相关的操作指令。

依照本发明的系统由存储器、文档库系统和应用软件组成。其中，文档数据存储在存储器中，文档库系统有一个下接口部，应用软件有一个上接口部。当应用软件需要对文档库进行操作时，通过其上接口部向文档库系统的下接口部发出指令，文档库系统按照应用软件发出的指令，对存储在存储器中的文档数据执行相应的操作。

本发明还提供一种文档处理方法，所述方法包括：

1）根据创建文档的指令，创建包含页的文档数据，至少一页所述文档数据分成多个组，所述被分成组的数据按先后顺序排列；

2）操作根据操作指令，对所述文档数据中的所述分成多个组的数据的至少一组数据执行操作。

所述至少一组数据由版面对象组成，所述版面对象包括状态、文字、图形、图像、语义信息、源文件、脚本、插件、嵌入式对象、书签、链接、流媒体和二进制数据流。

一个或多个版面对象组成一个对象组。

所述对象组还包括子对象组，所述子对象组由文字、图形、图像、流媒体、插件、脚本、二进制数据流和/或超链接组成。

同一组数据的不同对象组用不同的页面描述语言来定义。

所述操作包括对每一组数据单独进行签名。

所述操作包括根据不同的组数据的内容由不同的角色进行签名。

不同的组数据的内容由不同的软件来设置。

当需要显示页面时，单独控制每一组数据的显示方式，所述显示方式包括显示、不显示、叠加和/或水印效果。

当某一组数据被显示时，则其前面的所有组数据都会同时显示，显示方式都为叠加显示。

所述对文档数据的操作由文档库系统和应用软件执行，其中，文档库系统和应用软件通过一种标准调用接口连接，

1）所述应用软件向所述文档库系统发出操作指令；

2）所述文档库系统按照所述操作指令，对文档数据中相应组的数据执行相应的操作。

本发明还提供一种文档系统，包括：用于存储文档数据的存储器、文档库系统和应用软件；

所述存储器用于存储包含页且至少一页分成多个组的文档数据中，所述被分成组的数据按先后顺序排列；

所述文档库系统和应用软件用于操作所述文档数据中的所述分成多个组的数据的至少一组数据。根据本发明中的上述实现层的技术方案，可以对表示层的组数据进行独立操作，可以在层上添加各种数据对象，也可以在层上添加二进制数据流，同时，可以对层进行灵活的签名以及用不同的角色管理不同的层，也可以一个角色管理多个层。同时，还可以控制层的显示，层既可以单独显示，也可以采用叠加的方式显示某层之前所有的层。使得文档

的操作更加灵活，功能更加丰富。

依照本发明，使得应用层和数据处理层分离。这样应用软件不再直接跟具体的文档格式打交道，文档也不再与特定应用软件绑定，从而使得同一文档能在不同的应用软件之间通用，同一应用软件也能对不同文档进行操作，实现了文档的互操作；整个文档处理系统还具备多文档处理功能，而不局限在单文档处理；将页分成多层后，可以实现对不同层实施不同操作，更便于不同应用软件对同一页的操作（可以设计成不同应用软件管理和维护不同层），为以源文件方式进行编辑提供了便利，也是一种很好的保留历史痕迹的方式；通过将信息安全集成在文档处理的核心层，可以消灭安全缝隙，还能使安全机制与文档操作紧密地结合为一体，而不是可以分离的两个模块，同时有更多的空间部署安全管理技术，相关代码也能隐藏得更深，能更有效地防御非法攻击，提高安全可靠度，另外还能提供细粒度的安全管理手段，如更多的权限类别，更小的管理单元。

附图说明

图 1 为依照本发明的文档处理系统的结构框图。

图 2 为依照本发明的通用文档模型。

图 3-9 为依照本发明的文档模型的详细逻辑结构。

图 10-17 给出了本发明中所定义的动作。

图 18 为以 UOML 接口为例子的文档处理系统。

具体实施方式

下面，参照附图，描述依照本发明的文档操作系统。

如图 1 所示，依照本发明的文档处理系统主要由三个部分组成：应用软件、文档库系统和存储器。其中应用软件有一个上接口部，文档库系统有一个下接口部。

存储器常用的是硬盘或者内存，也可以是光盘、闪存、软盘、磁带，甚至还可以是远程的存储设备，总之只要具备数据的存储能力即可。在存储器中存储有多个文档，但对应用软件而言并不需要关心文档的具体存储方式，只需要按照预定的文档模型进行操作。图 2 所示为依照本发明的一种通用文

档模型。

各个软件的功能千差万别，对文档的操作和记录的数据也各自不同，例如 Word 和 Excel 处理的文档就大相迥异。为了能够定义出通用的文档模型，我们可以参考纸张的特性，这是因为以纸张作为文档信息的记录手段是通行至今的标准方法，只要能具备纸张的所有功能，就能满足工作、生活等实际应用的需求。

根据这个思路，我们把文档中的一页当成一张纸，凡是能画到纸上的就记录下来，即该文档模型能够描述页面上的所有可见内容。现有技术中的页面描述语言（如 PostScript）可以描述所有能印在纸上的信息，因此这一部分就不再详细阐述。一般说来，页面上的可见内容最终都可以归为文字、图形、图像三类。

如果文档中涉及到特定字体或特殊字符的话，为了保证在各台电脑上都能有相同的效果，就需要在文档中嵌入相应字库。为了提高存储效率，字库资源应当共享，这样即使在多处使用了同一字符，也只需要嵌入一个字库。图像有时也是可能在多处出现的，例如每一页共同的底图，或经常出现的公司标识，这种情况下最好也能共享这些图像。

当然，作为更加先进的信息处理工具，不能仅仅模拟纸张的特性，还可以增加一些增强的数字特性，例如元数据、导航、导读、微缩版面。元数据是描述数据的数据，例如作者、出版社、出版时间、ISBN 号等就是图书的元数据。元数据是业内通用名词，也不在此赘述。导航是类似图书目录的信息，也是业内通用名词。导读信息描述了一篇文章所在的区域和阅读顺序，这样当阅读者读完一屏后就可以根据该信息自动判断下一屏应该显示什么，这样还能做到自动换栏、自动转版，而不用阅读者再手工指定位置。微缩版面是事先生成的各页面的微缩图，阅读者可以通过查看微缩版面来指定阅读哪一页。

文档模型包含文档仓库、文档库、文档集、文档、页、层、对象组、版面对象等多个层次。

其中，文档仓库由一个或多个文档库组成，文档库之间的关系相对于文档库之下的层次之间的关系相对要松散一些，文档库之间可以非常简单地组

合和拆离，而不用对文档库本身的数据做改动，该多个文档库之间往往没有建立统一索引（特别是全文索引），很多对文档仓库的检索操作一般都需要遍历各文档库的索引，而没有统一的索引可用。每个文档库由一个或多个文档集组成，每个文档集由一个或多个文档组成，还可以包含任意数量的子文档集。这里所说的文档相当于目前普通的一个文档文件（例如 DOC 文档），文档模型可以规定一个文档只能属于一个文档集，但允许一个文档属于多个文档集也是一种不错的选择。文档库不是多个文档的简单组合，它把多个文档紧密地组织起来，特别是为文档内容统一建立了各种检索索引后就能带来更大的便利性。

每个文档由一页或存在一定顺序（如前后顺序）的多页组成，每页的版心可以不同，而且版心也不一定是矩形的，可以是任意形状，可以用一条或多条封闭曲线表示版心。

每页又由一层或按一定顺序（如上下顺序）的多层组成，各层之间如同玻璃板的叠加关系。层由任意数量的版面对象和对象组组成，版面对象是指状态（如字体、字号、颜色、ROP 等）、文字（包括符号）、图形（如直线、曲线、填充了指定颜色的闭合区域、渐变色等）、图象（如 TIF、JPEG、BMP、JBIG 等）、语义信息（如标题开始、标题结束、换行等）、源文件、脚本、插件、嵌入式对象、书签、链接、流媒体、二进制数据流等。一个或多个版面对象可以组成一个对象组。对象组也可以包含任意数量的子对象组。

文档库、文档集、文档、页、层都可以还包括元数据（如名称、最后修改时间等，其类型可以根据应用需求来设置）和／或历史痕迹；文档中还可以包括导航信息、导读信息、微缩版面；也可以把微缩版面放在页或者层这个层次；文档库、文档集、文档、页、层、对象组都可以还包括数字签名；语义信息最好跟着版面信息走，这样可以避免数据冗余，也比较容易与版面建立对应关系；文档库、文档还可以包括字库、图像等共享对象。

该文档模型还可以定义一个或多个角色，为每个角色分配一定权限。权限以文档库、文档集、文档、页、层、对象组、元数据为单元进行分配，定义每个角色对该单元是否可读、是否可写、是否可复制、是否可打印；

该文档模型是一个超越以往单个文档对应单个文件的方式，文档库中包

含多个文档集、文档集中包含多个文档，而对于文档库中文档内容，采用了细粒度的访问和安全控制，我们可以具体访问文档库中某个文字或者矩形，而不像现在的文档管理系统只能访问到文件名。

图 3-9 给出了一种符合本发明的文档模型，文档模型中所涉及的各对象以树状结构组织，逐层展开、细化。

文档仓库对象是由一个或多个文档库对象组成。

如图 3 所示，文档库对象是由一个或多个文档集对象、任意数量文档库辅助对象和任意数量的文档库共享对象组成。

其中，如图 4 所示，文档库辅助对象是指元数据对象、角色对象、权限对象、插件对象、索引信息对象、脚本对象、数字签名对象、历史痕迹对象等，文档库共享对象是指文档库中的不同文档可能共同使用的对象，如字库对象、图像对象等。

其中，如图 5 所示，每个文档集对象由一个或多个文档对象、任意数量的文档集对象和任意数量的文档集辅助对象组成。文档集辅助对象是指元数据对象、数字签名对象、历史痕迹对象。当文档集对象包括多个文档集对象时，其类似于文件夹包括多个文件夹的形式。

并且，如图 6 所示，每个文档对象由一个或多个页面对象、任意数量的文档辅助对象和任意数量的文档共享对象组成。文档辅助对象是指元数据对象、字库对象、导航信息对象、导读信息对象、微缩版面对象、数字签名对象、历史痕迹对象等，文档共享对象是指文档中的不同页面可能共同使用的对象，如图像对象、印章对象等。

在图 7 所示的页面对象中，每个页面对象由一个或多个层对象和任意数量的页面辅助对象组成。页面辅助对象是指元数据对象、数字签名对象、历史痕迹对象。

每个层对象由一个或多个版面对象、任意数量的对象组和任意数量的层辅助对象组成（如图 8 所示）。层辅助对象是指元数据对象、数字签名对象、历史痕迹对象。对象组由任意数量的版面对象、任意数量的对象组和可选的数字签名对象组成。当对象组包括多个对象组时，其类似于文件夹包括多个文件夹的形式。

进一步，如图9所示，版面对象是指状态对象、文字对象、直线对象、曲线对象、圆弧对象、路径对象、渐变色对象、图像对象、流媒体对象、元数据对象、批注对象、语义信息对象、源文件对象、脚本对象、插件对象、二进制数据流对象、书签对象以及超链接对象。

其中，状态对象又是由任意数量的字符集对象、字体对象、字号对象、文字颜色对象，光栅操作对象、背景色对象、线颜色对象、填充色对象、线型对象、线宽对象、线接头对象、画刷对象、阴影对象、阴影颜色对象、旋转对象、空心字对象、勾边字对象、透明对象、渲染模式对象组成。

在具体实施过程中，可以在上述文档模型基础上进一步增强或简化。如果在简化模型中省略了文档集对象，则文档库对象直接由文档对象组成；如果在简化模型中省略了层对象，则页面对象直接由版面对象组成。最简化的文档模型是只有文档对象、页面对象、版面对象，其中版面对象只有文字对象、直线对象、图像对象、字体对象、字号对象。完整模型和最简化模型之间的各种中间模型都属于本实施例的变形。

为了满足各种应用对文档安全性的需求，我们还需要定义一种通用的文档安全模型。由于现有软件的文档安全功能不够强，或者是安全管理机制与文档处理模块脱节，因此不难定义一个涵盖并超越现有应用软件的通用文档安全模型：

1. 在文档库中定义了若干角色，角色对象是文档库的子对象。如果对应的文档模型中没有文档库对象，则角色是在文档中定义的，即角色对象是文档对象的子对象，此时本文档安全模型中所说的文档库均用文档替代。

2. 可以指定任意角色对任意对象（文档库、文档集、文档、页、层、对象组、版面对象等）的访问权限。如果指定了对某个对象的访问权限，则该权限将适用于其所有子对象。

3. 文档库系统实现的访问权限包括是否可读、是否可写、是否可再授权（使其他角色拥有自己的部分或全部权限）、是否可收回授权（去掉其他角色的部分或全部授权）及上述权限的排列组合。可以定义更多权限（如不可打印），但需要由应用软件来配合实现。

4. 可以用某个角色的身份对各对象进行签名。签名范围将包括该对象的

子对象，以及引用到的对象。

5. 文档库的初始创建者具有对该文档库的所有权限。

6. 任何应用软件都可以创建新角色。新角色的初始权限是对任何对象都没有任何权限。可以用具有再授权权限的角色对新角色授予一定的权限。

7. 创建角色对象的指令返回一个密钥，作为今后登录该角色的依据，需要应用软件妥善保管。该密钥通常是 PKI 的私钥。

8. 当应用软件以某一角色身份登录时，通常采用“挑战－应答”机制，即文档库系统用保存的角色公钥加密一块数据发给应用软件，应用软件解密后返回给文档库系统，如果正确表明应用软件确实拥有该角色对应的私钥（为保险起见该认证过程可能会重复几次）。采用“挑战－应答”机制可以更好地保护私钥的安全性

9. 可以创建一个特殊的缺省角色。当存在缺省角色时，任何应用软件一打开文档库就视为自动以缺省角色身份登录。

10. 可以同时以多个角色身份登录，此时拥有的权限是各角色权限的并集。

在具体实施过程中，可以在上述安全模型基础上进一步增强、简化或合并步骤，都属于本实施例的变形。

根据上述文档模型、文档安全模型和常用的文档操作，可以定义相应接口标准，用于发送对文档模型中各对象进行操作的指令。特别地，如果在接口标准中定义了获取版面位图的指令，将对保障版面一致性和文档互操作性起到非常关键的作用。

通过获取版面位图的指令，应用软件可以直接获取指定页面的指定位图格式的版面位图（用位图方式表示的该页面的显示效果），而不用自行解释处理每一个版面对象。也就是说，应用软件可以直接获得准确的版面位图用于显示／打印文档，而不再需要自己挨个读取页面上每一层的每一个版面对象、自行解释该对象的含义并在版面上体现出来。如果采用后一种方式的话，就又难免出现有的软件解释的比较全、比较准确，有的软件解释的不全或不准确，导致同一个文档在不同软件出现不同的显示／打印效果，影响了文档互操作的用户体验。通过由文档库系统统一生成版面位图的方式，将保持版面

一致性的关键点从应用软件移到了文档库系统，从而为不同的应用软件打开同一文档都能出现同样的版面效果提供了可行之路。这一方面是因为文档库系统是统一的基础技术平台，由少数几家专业的技术厂商开发，肯定比各应用软件厂商实现得完整、准确，要求各文档库系统都能完整准确地解释处理各版面对象是可行的，而同样的要求对应用软件来说就不太可行了；另一方面是因为不同应用软件都可以与同一个文档库系统配套使用，这样就更能确保显示／打印效果的一致性了。简单来说，就是要求应用软件之间保持一致不太可行，而要求文档库系统之间保持一致则是可行的，要求同一个文档库系统保持一致就更没问题了。因此，为了保持同一文档在不同应用软件之间的版面一致性，就需要把相关责任从应用软件转移到文档库系统，而由文档库系统来统一生成版面位图是其中一个简单易行的办法。

更进一步，获取版面位图的指令还可以指定页面上的一个区域，可用于只显示页面的一个区域（例如当页面比屏幕大时就不需要显示整页，滚动页面时也只需要重画滚动的区域）；当该指令还允许指定获取特定层组成的版面位图，特别是可以指定由特定层以及该层下的所有层组成的版面位图时，就可以很好地用于展现历史痕迹，即可以看看在添加最近这一层以前是什么样，再往前又是什么样。如果需要的话，还可以具体指定哪一层参与位图的生成，哪一层不参与。

在检索查询指令中，除了常规的关键词检索外，还可以提供更加丰富的检索手段。在常规的搜索技术中，搜索是和文档处理分离的，搜索程序只能从文档中提取纯文本信息，而无法获取更多信息，只能基于文本信息检索。但在本发明中，检索查询功能是集成在文档处理的核心层（即文档库系统）的，这样就可以更充分地利用文档中蕴含的信息来提供更为强大的检索手段，如：

1. 基于字体信息的检索，如检索黑体字的“书生”，Times New Roman字体的“Sursen”。

2. 基于字号信息的检索，如检索三号字的“书生”，20磅以上的“Sursen”，长字（即字高超过字宽）的“文档库”。

3. 基于颜色的检索，如检索红色的“书生”，蓝色的“Sursen”。

4. 基于版面位置的检索，如检索位于页面上半部分的“书生”，位于页脚的“Sursen”。

5. 基于特殊修饰效果的检索，如检索斜体字的“书生”，顺时针旋转 30 度至 90 度之间的“Sursen”，空心字的“SEP”，勾边字的“文档库”。

6. 根据类似的思路，还可以进一步提供其它类型的检索，如检索反白（黑底白字）的“书生”，压图的“Sursen”等。

7. 可以检索多个版面对象的组合，如“书生”距离“Sursen”不超过 5 厘米。

8. 上述检索条件的任意组合。

现在介绍接口标准的实现方式。接口标准可以是上接口部按照预先定义的标准格式生成命令串（如“<UOML_INSERT (OBJ=PAGE, PARENT=123.456.789, POS=3)/>”），将该命令串发送给下接口部，并从下接口部接收执行结果或其它反馈信息；或者是下接口部提供一些具有标准名称和参数的接口函数（如“BOOL UOI_InsertPage(UOI_Doc *pDoc, int nPage)”），上接口部直接调用这些标准函数；或者是上述方法的组合。

接口标准还可以用“动作+对象”的方式来定义，这样便于学习和理解，也便于保持接口标准的稳定性。例如，对 20 种不同对象进行 10 种操作，可以定义 20x10=200 种指令，也可以定义 20 种对象和 10 种动作，但显然后一种方式大大减轻了记忆的负担，而且今后在对接口标准进行扩充时，增加一个对象或动作也很简单。

例如，我们定义以下 7 种动作：

打开：用于创建或打开文档库；

关闭：用于关闭会话句柄、关闭文档库；

获取：用于获取对象列表、对象相关属性和数据；

设置：用于设置/修改对象数据；

插入：插入指定对象或数据；

删除：用于删除对象的某个子对象；

检索查询：用于根据定义条件在文档中找到符合条件的内容,这些条件既可以是准确的信息，也可以是不准确的信息（模糊查找）。

我们再定义如下对象：文档库、文档集、文档、页、层、对象组、文字、图像、图形、路径（由一组顺序图形连接组成，可以是闭合也可以不闭合的）、源文件、脚本、插件、音频、视频、角色等。

对象还包括下列状态对象：背景色、线的颜色、填充色、线型、线宽、ROP、画刷、阴影、阴影颜色、字符高、字符宽、旋转、透明、渲染模式等。

在采用“动作+对象”方式时，不能自动理解为每一个对象和每一个动作的所有组合都一定能构成有实际意义的操作指令，在很多实施例中会存在一些组合是没有意义的，就象不是所有动词和所有名词都能组成以意义的词组一样。

以下是用“动作+对象”的格式定义命令的一种实施例，该实施例被称为 UOML,是用 XML 描述的一系列的命令。上接口部生成符合 UOML 格式的字符串，并将该字符串发送给下接口部，就将相应的操作指令发送给了文档库系统。文档库系统执行这些命令后，下接口部将执行结果也生成一个符合 UOML 格式的字符串，返回给上接口部，使应用软件能够知晓操作执行结果。

所有执行结果都由 UOML_RET 表示，其定义如下（参阅图 10）：

属性：

SUCCESS:为 true 时表明操作成功，为 false 表明操作失败。

子元素：

ERR_INFO：可选，仅当操作失败时出现，描述了相应的错误信息。

其它子元素：根据具体动作确定，可参考以下各动作说明。

UOML 动作包括：

1. UOML_OPEN 创建或打开文档库（参阅图 11）

1.1 属性

1.1.1 create：为 true 时是创建，否则是打开已有文档库

1.2 子元素：

1.2.1 path：文档库路径。可以是磁盘文件名，也可以是 URL，或者是内存指针，或者是网络路径，或者是文档库的逻辑名称，或者其它能够指定文档库的表示方法。可以用不同特征的字符串区分上述各种情况，即不用改变

命令格式，只要给字符串设置不同特征，就可以用不同的方法指定文档库。例如，磁盘文件名采用设备名称（如盘符）和“:”开头（如“C:”、 “D:”），而且紧跟着“:”不会是“//”，也不会是又一个“:”； URL 采用协议名称和“://”开头（如“http://”）；内存指针用“MEM::”开头，后面是指针的字符串表示方式，例如“MEM::1234:5678”；网络路径是“\\”开头，后面是服务器名，以及服务器上的路径，如“\\server\abc\def.sep”；文档库的逻辑名称可以用“*”开头，如“*MyDocBase1”。在下接口解析时，如果第一个字母是“*”就表明该字符串代表文档库的逻辑名称；否则如果头两个字母是“\\”就表明该字符串代表网络路径；否则如果头五个字母是“MEM::”就表明该字符串代表内存指针；否则寻找字符串的第一个“:”，如果该“:”后面是“//”该就表明字符串代表 URL，否则就代表本地设备上的文件。对于打开服务器上的文档库的情形，可以设立一个专门的 URL 协议来区分，例如用“Docbase://myserver/mydoc2”指明打开服务器 myserver 上运行的文档库系统服务器系统所管理的 mydoc2 文档库。

总之，只要能给字符串设置不同特征，就可以用不同的方式来指定文档库。根据上述说明，我们还可以定义各种不同的字符串特征；该方式不仅能应用于指定文档库路径，还能应用于其它场合，特别是用来指定特定资源位置的应用场合。在很多情况下，我们希望能够用一种新方式来指定相关资源，但又不能或不希望改变现有的协议或函数，这时就可以通过在字符串中设置不同特征的方式来指定，因为这种方法具有最好的通用性（任何协议或函数，只要支持磁盘文件名或 URL，就支持字符串）。

1.3 返回值：

如果成功，则在 UOML_RET 中包含一个“handle”子元素，记录句柄。

2. 关闭（UOML_CLOSE）（图 12）

2.1 属性：无

2.2 子元素：

2.2.1 handle：对象句柄，是一个字符串表示的对象的引用指针。

2.2.2 db_handle：文档库句柄，字符串表示的文档库的引用指针。

2.3 返回值：无返回值

3. UOML_GET 获取（参阅图 13）

3.1 属性

3.1.1 usage: 用途，为"GetHandle"（获取指定对象句柄）、"GetObj"（获取指定对象数据）、"GetPageBmp"（获取版面位图）中的一个

3.2 子元素

3.2.1 parent：父对象句柄，usage 属性为"GetHandle"时使用。

3.2.2 pos：位置顺序号，usage 属性为"GetHandle"时使用。

3.2.3 handle：指定对象的句柄，当 usage 属性为"GetObj"时使用。

3.2.4 page：需要显示的页面的句柄，当 usage 属性为"GetPageBmp"时使用。

3.2.5 input：描述了对输入页面的约束，其中可以指定显示一层或者多层的内容（可以显示的层一定是当前角色有权限访问的层）；也可以通过指定 Clip 区域来指定显示区域的大小。当 usage 属性为"GetPageBmp"时使用。

3.2.6 output：描述了版面位图的输出方式，当 usage 属性为"GetPageBmp"时使用。

3.3 返回值：

3.3.1 当 usage 属性为"GetHandle"时，执行成功时在 UOML_RET 中包含一个“handle”子元素，记录 parent 下第 pos 个子对象的句柄。

3.3.2 当 usage 属性为"GetObj"时，执行成功时在 UOML_RET 中包含一个“xobj”子元素，含有 handle 对象的数据的 xml 表示。

3.3.3 当 usage 属性为"GetPageBmp"时，执行成功时在 output 指定位置输出版面位图。

4. UOML_SET 设置（参阅图 14）

4.1 属性：无

4.2 子元素：

4.2.1 Handle：设置对象的句柄

4.2.2 xobj：对象的描述

4.3 返回值：无返回值

5. UOML_INSERT 插入（参阅图 15）

5.1 属性：无

5.2 子元素：

5.2.1 parent：父对象句柄

5.2.2 xobj：对象的描述

5.2.3 pos: 插入位置

5.3 返回值：如果执行成功，则将 xobj 参数表示的对象，插入到 parent 中成为其第 pos 个子对象，并在 UOML_RET 中包含一个"handle"子元素，表示新插入对象的句柄。

6. UOML_DELETE 删除（参阅图 16）

6.1 属性：无

6.2 子元素：

6.2.1 handle：需要删除的对象的句柄。

6.3 返回值：无返回值

7. UOML_QUERY 检索查询（参阅图 17）

7.1 属性：无

7.2 子元素：

7.2.1 handle: 需要查询的文档库句柄

7.2.2 condition：查询条件

7.3 返回值：如果成功，在 UOML_RET 中包含一个“handle”子元素代表查询结果的句柄，一个“number”子元素代表查询结果的数量，可以用 UOML_GET 来获取每一个查询结果。

UOML 对象包括：

文档库（UOML_DOCBASE）、文档集（UOML_DOCSET）、文档（UOML_DOC）、页（UOML_PAGE）、层（UOML_LAYER）、对象组（UOML_OBJGROUP）、文字（UOML_TEXT）、图像（UOML_IMAGE）、直线（UOML_LINE）、曲线（UOML_BEIZER）、圆弧（UOML_ARC）、路径（UOML_PATH）、源文件（UOML_SRCFILE）、背景色（UOML_BACKCOLOR）、前景颜色（UOML_COLOR）、ROP（UOML_ROP）、字符尺寸（UOML_CHARSIZE）、字体（UOML_TYPEFACE）、角色（UOML_ROLE）、权限（UOML_PRIV）等。

以下我们以部分对象为例说明其定义方式：

1. UOML_DOC

1.1 属性：无

1.2 子元素：

1.2.1 metadata: 元数据

1.2.2 pageset: 各页面

1.2.3 fontinfo: 嵌入字库

1.2.4 navigation: 导航信息

1.2.5 thread: 导读信息

1.2.6 minipage: 微缩版面

1.2.7 signiture: 数字签名

1.2.8 log: 历史痕迹

1.2.9 shareobj: 文档共享对象

2. UOML_PAGE:

2.1 属性：

2.1.1 resolution：逻辑分辨率

2.1.2 size：版心大小，用宽高表示

2.1.3 rotaion：旋转角度

2.1.4 log：历史痕迹

2.2 子元素：

2.2.1 GS：初始图形状态，包括 charstyle（字符风格）、linestyle（线型）、linecap（线头类型）、linejoint（接头类型）、linewidth（线宽）、fillrule（填充规测）、charspace（字间距）、linespace（行间距）、charroate（字符旋转角度）、charslant（字符倾斜方向）、charweight（字色重）、 chardirect（字符方向）、textdirect（文本方向）、shadowwidth（阴影宽度）、shadowdirect（阴影方向）、shadowboderwidth（阴影边线宽度）、outlinewidth（轮廓宽度）、outlineboderwidth（轮廓边线宽度）、linecolor（线的颜色）、fillcolor（填充色）、backcolor（背景色）、textcolor（文字颜色）、shadowcolor（阴影颜色）、outlinecolor（轮廓线颜色）、matrix（变换矩阵）、cliparea（裁减区）。

2.2.2 metadata: 元数据

2.2.3 layerset：属于该页的各层

2.2.4 signiture: 数字签名

2.2.5 log: 历史痕迹

3. UOML_TEXT

3.1 属性：

3.1.1 Encoding： 文字编码方式

3.2 子元素：

3.2.1 TextData：文字内容

3.2.2 CharSpacingList：对非等间距文字的字间距列表

3.2.3 StartPos: 起点位置

4. UOML_CHARSIZE

4.1 属性：

4.1.1 width：字符宽度

4.1.2 height：字符高度

4.2 子元素：无

5. UOML_LINE

5.1 属性：

5.1.1 LineStyle： 线型

5.1.2 LineCap：线的接头类型

5.2 子元素：

5.2.1 StartPoint：线的起点坐标

5.2.2 EndPoint：线的终点坐标

6. UOML_BEIZER

6.1 属性：

6.1.1 LineStyle： 线型

6.2 子元素：

6.2.1 StartPoint：贝塞尔曲线的起点坐标

6.2.2 Control1_Point: 贝塞尔曲线的第一控制点

6.2.3 Control2_Point: 贝塞尔曲线的第二控制点

6.2.4 EndPoint：贝塞尔曲线的终点坐标

7. UOML_ARC

7.1 属性：

7.1.1 ClockWise：弧的方向

7.2 子元素：

7.2.1 StartPoint：弧的起点坐标

7.2.2 EndPoint：弧线的终点坐标

7.2.3 Center：弧的圆心坐标

8. UOML_COLOR

8.1 属性：

8.1.1 Type： 颜色类型，RGB 或 CMYK

8.2 子元素：

RGB 模式

8.2.1 Red：红色

8.2.2 Green: 绿色

8.2.3 Blue: 蓝色

8.2.4 Alpha: 透明度

CMYK 模式

8.2.5 Cyan：青色

8.2.6 Magenta：品红

8.2.7 Yellow：黄色

8.2.8 Black_ink：黑色

以此类推，我们可以用同样的方法来描述所有的 UOML 对象。当应用软件对文档库进行操作时，由上述 UOML 动作与 UOML 对象依照 XML 语法生成相应的 UOML 命令。具体地，在本发明中，采用 XML 元素来描述动作，该元素下的子元素来描述具体对象，采用字符串来表示数值，并且对象的详细信息采用属性描述。通过这种形式，可以实现对功能调用和调用返回结果的 XML 描述。对该 XML 描述语义上的约束，采用 DTD 或 schema。然后，将该 UOML 命令发给文档库系统即代表向文档库系统发出了相应操作指令。XML（eXtensible Markup Language，可扩展置标语言）是由 W3C（World Wide Web Consortium，互联网联合组织）于 1998 年 2 月发布的一种标准，同 HTML 一样是 SGML（Standard Generalized Markup Language，标准通用置标语言）的一个简化子集。XML 语法和各种对象操作指令可参见该标准。同时可以参考"Extensible Markup Language (XML) 1.1，W3C Recommendation 04 February 2004, edited in place 15 April 2004"、"W3C（World Wide Web Consortium）eXtensible Markup Language（XML）1.0（REC-xml-19980210）"、"W3C XML Schema Part 0-4 （REC-xmlschema）"、"W3C Namespaces in XML（REC-xml-names-199090114）"、"W3C XSL Transformations (XSLT) Version 1.0 （REC-xslt-19991116）"、"Document Object Model (DOM) Level 1 Specification (Second Edition) Version 1.0，W3C Working Draft 29 September, 2000"、"美国 Federal CIO Council XML Working Group Draft Federal XML

Developer's Guide"以及"英国 Office of the e-Envoy , e-Government Schema Guidelines for XML"。

例如，对创建文档库操作，可以用以下命令来完成：

```
<UOML_OPEN create="true">
  <path val="f:\\data\\docbase1.sep"/>
</UOML_OPEN>
```

对创建文档集操作，可以用以下命令来完成：

```
<UOML_INSERT >
  <parent val= "123.456.789"/>
  <pos val="1"/>
  <xobj>
      <docset/>
</xobj>
</UOML_INSERT>
```

需要说明的是，虽然 UOML 是用 XML 定义的，但为了显得更加简洁，我们在前面省略了类似“<?xml version="1.0" encoding="UTF-8"?>”以及“xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"”之类的常规 XML 格式，只要是熟悉 XML 语法的实施者都可以自行补充完整。

我们也可以不用 XML 方式定义命令串，例如改用类似 PostScript 那样的方式，这样上例变成这样的：

```
1, "f:\\data\\docbase1.sep", /Open
/docset, 1, “123.456.789” , /Insert
```

根据同样的思路，我们还可以定义出其它类型的命令串格式，甚至我们还可以不用文本方式，而用二进制方式来定义命令串。

除了“动作＋对象”方式外，我们也可以用其它方式定义命令串。例如，对每一个对象的每一个操作都用一个命令来表示，即用“UOML_INSERT_DOCSET”来表示插入一个文档集，用“UOML_INSERT_PAGE”来表示插入一页，我们以这样的方式来定义每个命令：

UOML_INSERT_DOCSET 在文档库中创建一个文档集

属性：无

子元素：

parent：文档库句柄

pos: 插入位置

返回值：如果执行成功，则在 UOML_RET 中包含一个"handle"子元素，表示新插入文档集的句柄

这样上例就变为：

```
<UOML_INSERT_DOCSET >
  <parent val="123.456.789"/>
  <pos val="1"/>
</UOML_INSERT_DOCSET >
```

用这种方法定义命令格式的话就需要对每个对象的每种合法操作都单独定义一条命令，会比较繁琐。

接口标准也可以用函数调用的方式来实施，即通过上接口调用下接口的接口函数的方式来发送操作指令给文档库系统的：

以下以 C++语言为例说明，该实施例称为 UOI。

我们先定义一个 UOI 返回值结构：

```
struct UOI_Ret {
  BOOL    m_bSuccess;        // 操作是否成功
  CString   m_ErrInfo;    // 如果操作不成功，错误信息是什么
};
```

定义所有 UOI 对象的基础类：

```
class UOI_Object {
public:
  enum Type {//类型定义
      TYPE_DOCBASE,//文档库
      TYPE_DOCSET,//文档集
      TYPE_DOC,//文档
```

```
TYPE_PAGE,//页
TYPE_LAYER,//层
TYPE_TEXT,//文字
TYPE_CHARSIZE,//字符尺寸
        ……对文档模型中定义的其它对象的类型的定义与上面类似，以下省略。
    };
    Type      m_Type;//类型

    UOI_Object();//构造函数
virtual ~ UOI_Object();//析构函数
static UOI_Object *Create(Type objType);      //根据指定类型创建相应对象
};
```

然后定义如下几个 UOI 函数，与第一个实施例中的几个 UOML 动作相对应：

打开或创建文档库，成功则将其句柄返回在 pHandle 中：

```
UOI_RET UOI_Open(char *path, BOOL bCreate, HANDLE *pHandle);
```

关闭 db_handle 文档库中的 handle 句柄，如果 handle 为 NULL 则关闭整个文档库：

```
UOI_RET UOI_Close(HANDLE handle, HANDLE db_handle);
```

获取指定子对象句柄：

```
UOI_RET UOI_GetHandle(HANDLE hParent, int nPos, HANDLE *pHandle);
```

获取句柄所指向的对象的类型：

```
UOI_RET UOI_GetObjType(HANDLE handle, UOI_Object ::Type *pType);
```

获取句柄所指向的对象数据：

```
UOI_RET UOI_GetObj(HANDLE handle, UOI_Object *pObj);
```

获取版面位图：

```
UOI_RET UOI_GetPageBmp(HANDLE hPage, RECT rect, void *pBuf);
```

设置对象：

```
UOI_RET UOI_SetObj(HANDLE handle, UOI_Object *pObj);
```

插入对象：

```
UOI_RET UOI_Insert(HANDLE hParent, int nPos, UOI_Object *pObj, HANDLE *pHandle = NULL);
```

删除对象：

```
UOI_RET UOI_Delete(HANDLE handle);
```

检索查询，检索结果的数量返回在 pResultCount 中，检索结果列表的句柄返回在 phResult 中：

```
UOI_RET UOI_Query(HANDLE hDocbase, const char *strCondition, HANDLE *phResult, int *pResultCount);
```

然后定义各 UOI 对象，依然以 UOI_Doc、UOI_Text 和 UOML_CharSize 为例说明：

```
class UOI_Doc : public UOI_Object {
public:
    UOI_MetaData      m_MetaData;//元数据
    int                       m_nPages;//页数
    UOI_Page                  **m_pPages;//页指针
int                           m_nFonts;//字体数
UOI_Font                  **m_pFonts;//字体列表
UOI_Navigation m_Navigation ;//导航对象
UOI_Thread            m_Thread ;//导读
UOI_MiniPage   *m_pMiniPages ;//微缩页面
UOI_Signature   m_Signature ;//签名
int                           m_nShared ;//共享对象数
UOI_Obj                   *m_pShared;//共享列表

UOI_Doc();//构造函数
virtual ~UOI_Doc() ;//解析函数
};

class UOI_Text : public UOI_Object {
```

```
public:
   enum Encoding {
        ENCODE_ASCII,//ascii 编码
        ENCODE_GB13000,//GB13000 编码
        ENCODE_UNICODE,//Unicode 编码
        ......
   } ;
   Encoding m_Encoding;//编码类型
   char       *m_pText ;//文字串
   Point          m_Start ;//起点坐标
   int            *m_CharSpace ;//字符间距数组

UOI_Text();//构造
virtual ~ UOI_Text();//析构
};
//对象 UOI_CharSize 的定义
class UOI_CharSize : public UOI_Object {
public :
   int   m_Width ;//宽度
   int   m_Height ;//高度

   UOI_CharSize();//构造函数
   virtual ~UOI_CharSize();//析构函数
};
```

以下示例说明 UOI 的使用方法。首先是创建文档库操作：

```
ret = UOI_Open("f:\\data\\docbase1.sep", TRUE, &hDocBase);
```

然后是构建一个创建新对象的函数：

```
HANDLE InsertNewObj(HANDLE hParent, int nPos, UOI_Object ::Type type)
{
```

```
UOI_Ret          ret;//返回值
HADNLE    handle ;//对象句柄
UOI_Obj     *pNewObj = UOI_Obj::Create(type);//创建对象
if (pNewObj == NULL)
   return NULL;//创建失败返回空值
ret = UOI_Insert(hParent, nPos, pNewObj, &handle) ;//插入到父结点，返回对象句柄 handle
delete pNewObj ;//删除临时对象
return ret.m_bSuccess ? handle : NULL;//成功返回对象句柄 handle，失败返回空值
   }
```

然后是直接获取对象的函数：

```
UOI_Obj *GetObj(HANDLE handle)
{
UOI_Ret          ret;//返回值
UOI_Object ::Type          type;//对象类型
UOI_Obj     *pObj;//对象指针
ret = UOI_GetObjType(handle, &type);//根据对象类型 type
if ( !ret. m_bSuccess )
   return NULL;//如果返回失败，本函数返回空值
pObj = UOI_Obj::Create(type);//创建 type 类型的对象
if (pObj == NULL)
   return NULL;//如果新建对象为空，返回空值
ret = UOI_GetObj(handle, pObj);获取对象
if ( !ret. m_bSuccess ) {
   delete pObj;//失败删除临时指针
   return NULL;
}
return pObj;//返回对象指针
}
```

我们还可以用非“动作＋对象”的函数方式来定义接口标准，例如对每一个对象的每一种操作都定义一个接口函数，这样插入文档集的操作指令就是上接口以下列方式调用下接口的接口函数来发送给文档库系统的：

```
UOI_InsertDocset(pDocbase, 0);
```

我们还可以封装各个对象类（如文档库类）,把该对象可以进行的操作定义成该类的方法，如：

```
class UOI_DocBase : public UOI_Obj
{
public:
/*!
 *  \brief        创建文档库
 *  \param      szPath:    文档库全路径
 *  \param      bOverride:是否覆盖原文件
 *  \return    UOI_DocBase 对象
 */
  BOOL Create(const char *szPath, bool bOverride = false);
/*!
 *  \brief        打开文档库
 *  \param      szPath: 文档库全路径
 *  \return    UOI_DocBase 对象
 */
  BOOL Open(const char *szPath);
/*!
*  \brief          关闭文档库
*  \param      无
*  \return      无
*/
  void Close();
/*!
```

```
 *  \brief          获取角色列表
 *  \param      无
 *  \return       UOI_RoleList 对象
 *  \sa             UOI_RoleList
 */
  UOI_RoleList GetRoleList();
/*!
 *  \brief          存储文档库
 *  \param      szPath: 存储文档库全路径
 *  \return       无
 */
  void Save(char *szPath = 0);

/*!
 *  \brief          插入文档集
 *  \param      nPos:插入文档集的位置
 *  \return       UOI_DocSet 对象
 *  \sa             UOI_DocSet
 */
  UOI_DocSet InsertDocSet(int nPos);
 /*!
 *  \brief          获取指定索引的文档集
 *  \param      nIndex: 文档列表的索引号
 *  \return       UOI_DocSet 对象
 *  \sa             UOI_DocSet
 */
  UOI_DocSet GetDocSet(int nIndex);
 /*!
 *  \brief          获取文档集的总数
```

```
*  \param      无
*  \return     文档集个数
*/
 int GetDocSetCount();
/*!
*  \brief      设置文档库的名称
*  \param      nLen:       文档库名称长度
*  \param      szName:  文档库名称
*  \return     无
*/
 void SetName(int nLen, const char* szName);
/*!
*  \brief      获取文档库名称长度
*  \param      无
*  \return     长度
*/
 int GetNameLen();
/*!
*  \brief      获取文档库名称
*  \param      无
*  \return     文档库名称
*/
 const char* GetName();
/*!
*  \brief      获取文档库 id 长度
*  \param      无
*  \return     长度
*/
 int GetIDLen();
```

```
/*!
*  \brief          获取文档库 id
*  \param      无
*  \return        id
*/
  const char* GetID();

  //! 构造函数
  UOI_DocBase();
  //! 析构函数
  virtual ~UOI_DocBase();

};

    class   UOI_Text   : public UOI_Obj
    {
    public:
        //!  构造函数
        UOI_Text();
        //!  析构函数
        virtual ~UOI_Text();
        //!  表示文本编码的枚举类型
        enum UOI_TextEncoding
        {
            CHARSET_GB2312 ,   /*!< GB2312, a1a1-fefe */
            CHARSET_HZ2312 ,   /*!< GB2312 except GBFH, b0a1-fefe */
            CHARSET_GB12345,/*!< GB12345, traditional char of GB2312, a1a1-fefe */
            CHARSET_HZ12345,/*!< GB12345 except GBFH, traditional char of HZ2312, b0a1-fefe */
            CHARSET_GB13000,/*!< GBK, 8141-fefe */
```

```
        CHARSET_HZ13000,/*!< GBK except GBFH, 8141-fefe except a1a1-affe */
        CHARSET_GB18030,/*!< GB18030 except GBFH, unsupported in this version */
        CHARSET_HZ18030,/*!< GB18030 except GBFH, unsupported in this version */
        CHARSET_UNICODE,/*!< UniCode, unsupported in this version */
        CHARSET_ASCII ,     /*!< ASCII 编码 */
};

//! 获得文本的编码
UOI_TextEncoding GetEncoding();
//! 设置文本的编码
void SetEncoding(UOI_TextEncoding nEncoding );
//! 获得文本的数据
const char * GetTextData();
//! 获得文本的数据长度
int GetTextDataLen();
//! 设置文本的数据
/*!
\param pData 文本数据
\param nLen 数据长度
*/
void SetTextData(const char * pData, int nLen);
//! 获得起点位置
Point GetStartPoint();
//! 设置起点位置
void SetStartPoint(Point startPoint);
//! 获得字符间距表大小
int GetCharSpacingCount();
//! 获得字符间距表中指定位置的字符间距
float GetCharSpacing(int nIndex);
```

```
    //! 设置字符间距表大小
    bool SetCharSpacingCount(int nLen);
    //! 设置字符间距
    bool SetCharSpacing (int nIndex, float charSpace );
    //! 获得文本的外框
    UOI_Rect GetExtentArea() ;
};

class   UOI_Arc : public UOI_Obj   {     // 圆弧对象及其操作
public:
     //! 构造函数
    UOI_Arc();
     //! 析构函数
    virtual ~UOI_Arc();
     //! 获得圆弧起点
     /*!
     \return 圆弧起点
     */
    UOI_Point GetStartPoint();
     //! 获得圆弧终点
     /*!
     \return 圆弧终点
     */
    UOI_Point GetEndPoint();
     //! 获得圆弧旋转角
     /*!
     \return 椭圆横轴与坐标系 X 轴的夹角，单位为弧度
     */
    float GetRotAng();
```

```
    //! 设置圆弧旋转角
    /*!
    \param fRotAng 新的旋转角
    \sa GetRotAng()
    */
    void SetRotAng(float fRotAng);
    //!获得 X 半轴长度
    float GetRadiusX();
    //!设置 X 半轴长度
    void SetRadiusX(float fRx);
    //!获得 Y 半轴长度
    float GetRadiusY();
    //!设置 Y 半轴长度
    void SetRadiusY(float fRy);
    //!获得弧线方向（是否为顺时针）
    bool GetClockWise();
    //!设置弧线方向（是否为顺时针）
    void SetClockWise(bool bClockWise);
    //!获得（由圆弧起点->圆心->圆弧终点，是否为顺时针方向）
    bool GetGreatArcFlag();
    //!设置（由圆弧起点->圆心->圆弧终点，是否为顺时针方向）
    void SetGreatArcFlag(bool bGreat);
    //!计算圆心，由参数返回，如果数据无效，则返回 false，否则返回 true
    bool GetCenter(float &fCx, float &fCy);
};

class   UOI_RoleList : public UOI_Obj      // 文档库中的角色列表
{
public:
```

```
    //! 获得列表中角色的数目
    int GetRoleCount();
    //! 按指定索引获得角色
    UOI_Role *GetRole(int nIndex);
    //! 创建角色
    /*!
    \param pPrivKey 私钥缓冲区
    \param pnKeyLen 用于返回实际私钥的长度
    \return 新创建的角色
    */
    UOI_Role AddRole(unsigned char *pPrivKey, int *pnKeyLen);
    //! 构造函数
    UOI_RoleList();
    //! 析构函数
    virtual ~UOI_RoleList();
};

class  UOI_Role  : public UOI_Obj  //  文档库中的角色
{
public:
    //! 构造函数
    UOI_Role();
    //! 析构函数
    virtual ~UOI_Role();

    //! 获得角色 ID
    int GetRoleID();
    //! 设置角色 ID
    /*!
```

```
    \param nID 角色 ID
    */
    void SetRoleID(int nID);
    //! 获得角色名称
    const char * GetRoleName();
    //! 设置角色名称
    /*!
    \param szName  角色名称
    */
    void SetRoleName(const char *szName);
};

class   UOI_PrivList : public UOI_Obj   // 权限列表，每个权限列表由若干角色权限项组成
{
public:
    //! 获得指定角色对应的权限
    UOI_RolePriv *GetRolePriv (UOI_Role *pRole);
    //! 新建某角色的权限项
    UOI_RolePriv *AddRole (UOI_Role *pRole);
    //! 获得列表中角色权限项的数目
    int GetRolePrivCount();
    //! 按索引值，获得角色权限项
    UOI_RolePriv *GetRolePriv (int nIndex);
    //! 构造函数
    UOI_PrivList();
    //! 析构函数
    virtual ~UOI_PrivList();
};
```

```
class  UOI_RolePriv : public UOI_Obj  // 角色权限项，对应于某一个角色的所有权限，由若干针对某个对象的权限组成
{
public:
    //! 获得角色
    UOI_Role *GetRole();
    //! 设置对某个对象的权限,当权限超过该角色对该对象的当前权限时为授权，小于时为收回授权。当前登录的角色必须有相应的再授权或收回授权权限
    bool SetPriv(UOI_Obj *pObj, UOI_Priv *pPriv);
    //!获得权限设置数量
    int GetPrivCount();
    //! 获得索引值对应的权限设置的对象
    UOI_Obj *GetObj(int nIndex);
    //! 获得索引值对应的权限设置的权限
    UOI_Priv *GetPriv(int nIndex);
  //! 获得对应于某一个对象的的权限
    UOI_Priv *GetPriv(UOI_Obj *pObj);
    //! 构造函数
    UOI_RolePriv ();
    //! 析构函数
    virtual ~UOI_RolePriv ();
};

class  UOI_Priv : public UOI_Obj  // 权限的定义
{
public:
enum PrivType {      // 各权限类型定义
    PRIV_READ,          // 读权限
    PRIV_WRITE,         // 写权限
```

```
    PRIV_RELICENSE,     // 再授权权限
    PRIV_BEREAVE,       // 收回授权权限
    PRIV_PRINT,         // 打印权限
    其它权限定义
}
    //! 是否有相应权限
    bool GetPriv(PrivType privType);
    //! 设置相应权限
    void SetPriv(PrivType privType, bool bPriv);
//! 构造函数
    UOI_Priv ();
    //! 析构函数
    virtual ~UOI_Priv ();
};

class   UOI_SignList : public UOI_Obj   // 数字签名列表
{
public:
    //! 构造函数
    UOI_SignList();
    //! 析构函数
    virtual ~UOI_SignList();

    //! 添加新的数字签名,返回其索引值
    int   AddSign(UOI_Sign *pSign);
    //! 按指定索引值，获得指定数字签名
    UOI_Sign   *GetSign(int index);
    //! 按索引值，删除指定数字签名
    void DelSign(int index);
```

```
    //! 获得列表中数字签名的数目
    int   GetSignCount();
};

class   UOI_Sign : public UOI_Obj   // 数字签名
{
public:
    //! 构造函数
    UOI_Sign();
    //! 析构函数
    virtual ~UOI_Sign();

    //! 执行签名
    /*!
    \param pDepList 签名所依赖的列表
    \param pRole 用于签名的角色
    \param pObj 被签名的对象
    */
    void Sign(UOI_SignDepList *pDepList, UOI_Role *pRole , UOI_Obj *pObj);
    //! 验证签名
    bool Verify();
    //! 获得签名的依赖列表
    UOI_SignDepList *GetDepList();
};

class   UOI_SignDepList : public UOI_Obj   // 签名的依赖列表
{
public:
    //! 构造函数
```

```
    UOI_SignDepList();
    //! 析构函数
    virtual ~UOI_SignDepList();

    //! 加入一个依赖项
    void InsertSignDep(UOI_Sign *pSign);
    //! 获得依赖项的数目
    int    GetDepSignCount();
    //! 按指定索引值，获得依赖项
    UOI_Sign *GetDepSign(int nIndex);
    //! 按索引值，删除指定依赖项
    bool *DelDepSign(int nIndex);
};
```

这样插入文档集的操作指令就是上接口以下列方式调用下接口的接口函数来发送给文档库系统的：

pDocBase.InsertDocset(0);

我们还可以用同样的方法为 Java、C#、VB、Delphi 等各种编程语言开发的应用软件设计各种不同的接口标准。

只要在接口标准中不含有与特定的操作系统（如 WINDOWS、UNIX/LINUX、MAC OS、SYMBIAN）或特定的硬件平台（如 x86CPU、MIPS、POWER PC 等）相关连的特征，该接口标准就可以具有跨平台性，使得不同平台上运行的应用软件和文档库系统都可以统一使用同样的接口标准，特别是可以让一个平台上运行的应用软件可以调用另一个平台上运行的文档库系统来执行相应操作。例如，应用软件部署在客户端，使用的是 PC 机，Windows 操作系统，文档库系统部署在服务器端，使用的是大型机，Linux 操作系统，但应用软件依然可以像调用本地文档库系统一样调用服务器上的文档库系统来执行相应文档操作。

如果在接口标准中不含有与特定编程语言相关的特征，则该接口标准还

能做到与编程语言无关。可以看出，用命令串的方式容易构造与平台无关、与编程语言无关的接口标准，更具有通用性。特别是用 XML 来构造命令串的话，由于目前在各种不同平台、不同编程语言都存在易于获得的 XML 生成解析工具，因此不仅该接口标准具有很好的跨平台性和与编程语言无关性，也非常便于工程师开发上接口部和下接口部。

以上列举了多种接口标准的实施方法，按照类似的思路，不难设计出更多种类的接口标准。

接口标准可以在上述实施例的基础上按同样的思路增加操作指令，也可以简化操作指令，特别是文档模型被简化时操作指令也会相应被简化。最简化情况下只有文档的创建、页面的创建、各版面对象的创建这几个操作指令。

现在，返回图 1，继续描述依照本发明的文档操作系统的工作过程。

应用软件可以是具有符合接口标准的上接口部的任意软件，例如 Office 软件、内容管理、资源采集等。任一应用软件在需要对文档进行操作时，依照前述方法将指令传递给文档库系统，文档库系统根据指令来完成具体操作过程。

文档库系统可以自由地存储、组织文档库数据，例如可以把一个文档库的文件全部都存储在一个磁盘文件中；可以一个文档对应一个磁盘文件，利用操作系统中的文件系统功能实现多文档组织；也可以一页对应一个磁盘文件；还可以完全抛开操作系统，在磁盘上留出一块空间后直接对磁道、扇区进行管理。对文档库数据的存储格式，可以用二进制格式保存，可以用 XML，还可以用二进制 XML。页面描述语言（定义页面上的文字、图形、图像等对象的方法）可以用 PostScript，可以用 PDF，可以用 SPD(书生公司使用的页面描述语言)，当然也可以自定义。总之，只要能够实现接口标准所定义的功能，任何实现方式都是可以的。

例如，我们可以用 XML 来描述文档库数据，当文档模型是层次型的时候，可以完全对照建立相应的 XML 树。执行创建操作时就在 XML 树中增加一个结点（可以将每个节点对应的数据看作是一组，例如，对应每层的子节点下的数据可以看作是一组），执行删除操作就删掉相应结点，执行设置操作就设置相应结点的属性，执行获取操作就取出相应结点的属性并返回给应用

软件，执行查询操作时就遍历相关结点查找。

以下是该实施例的进一步说明：

1. 用 XML 来描述每个对象。也就是说，为每个对象按照其 XML Schema 建立一个对应的 XML 树。有的对象属性比较简单，其对应的 XML 树就只有根结点，有的对象比较复杂，其对应的 XML 树还有子结点。具体描述方法可以参见前面用 XML 来定义操作对象的说明。

2. 当新建一个文档库时就新建一个根结点为文档库对象的 XML 文件。

3. 每当在文档库中插入一个对象时（如文字对象），就将该对象对应的 XML 树插入到插入位置的父结点（如层）之下。这样，文档库中的每个对象都在文档库为根结点的 XML 树中有一个对应的结点。

4. 当删除一个对象时，就删除该对象对应的结点，其下属所有子结点也都被删除。删除过程是从叶子结点开始自下而上遍历的。

5. 设置一个对象属性时，将该对象对应的结点的属性设置成该属性。如果该属性是用子结点表示的，则设置对应的子结点。

6. 获取一个对象属性时，访问该对象对应的结点，根据该结点的属性和子结点获得该对象的属性。

7. 获取一个对象的句柄时，返回该对象对应结点的 XML 路径。

8. 复制一个对象（如页面）到指定位置时，就将该对象对应的结点开始的整个子树都复制到目标位置对应的父结点（如文档）之下。如果是复制到另一个文档库中，则需要将该子树引用的对象（如嵌入字库）也一起复制过去。

9. 执行获取版面信息指令时，先生成一个指定位图格式的空白位图，其尺寸和指定区域相同，然后遍历指定页面的所有版面对象，凡是位于指定区域内（包括只有一部分在该区域内）的版面对象，都解释其含义，并在版面上相应体现。具体过程虽然比较复杂比较专业，但均属于现有 RIP 技术范畴，不在此赘述。

10. 在创建角色对象时，生成一对随机 PKI 钥对（例如 512 位的 RSA 密钥），将公钥存储在角色对象中，将私钥返回给应用软件。

11. 当应用软件登录时，随机生成一块（例如 128 字节）数据，用相应

角色对象中的公钥加密该数据发给应用软件，应用软件解密后比较验证，如果正确则表明应用软件确实拥有该角色对应的私钥，登录成功。为保险起见，该认证过程可以重复三次，三次全部通过才算登录成功。

12. 当对某一对象进行签名时，也就是对其对应的结点开始的子树进行签名。为了能够使签名不受具体物理存储方式的影响，需要先做一个正则化，使得逻辑上等效的变化（例如存储位置的改变导致相应指针的变化）不会影响签名有效性。该正则化的方法如下：

a）对树的某一结点，先将该结点的子结点数计算 HASH 值，然后再依次计算其类型和各个属性的 HASH 值，按顺序连接在子结点数 HASH 值的后面。对连接的结果再计算其 HASH 值，得到该结点的正则结果；

b）从子树的根节点开始，按照上述方法计算该结点的正则结果，并对其所有子结点，按照从左到右顺序依次计算其正则结果，将子结点的正则结果按顺序附加到父结点正则结果之后；

c）这是一个深度优先的递归过程。递归结束之后，即得到最终结果。

d）如果需要对被引用的对象也一起做签名，则可以将被引用对象也作为一个子结点处理，方法同上。

正则化以后，再做 HASH 并用角色的私钥进行签名就属于现有技术了。

在上述正则化过程中，我们可以把 a)改成如下方案：对树的某一结点，将该结点的子结点数、类型及其各属性用分隔符隔开后按照顺序连接起来，对连接的结果计算其 HASH 值，得到该结点的正则结果；

我们还可以把 a)改成如下方案：对树的某一结点，其子结点数、类型及其各属性的长度用分隔符隔开后按照顺序连接起来，再与子结点数、类型、各属性连接起来，即为该结点的正则结果；

总之，a)可以是以下各种方案中的任意一种：对树的某一结点，其子结点数、类型、各属性，子结点数 / 类型 / 各属性的长度（可选的），原值或经过特定变换（如 HASH、压缩），按照预定顺序连接起来（直接连接或用分隔符隔开）。

上述预定顺序的意思是，子结点数长度、类型长度、各属性长度、子结点数、类型、各属性可以按任意顺序排列，只要是预定的顺序即可，b)、c)

步骤也可以改为宽度优先。

我们不难给出上述方案的各种变化方式，如每个结点的子结点数用分隔符隔开后按照深度优先的顺序连接起来，再与各结点其它数据的正则结果连接起来。总之，只要对该子树中的所有结点的子结点数、类型和各属性，按照确定的方法排列在一起就属于本实施例的变形。

13. 当对某一对象设置权限时，最简单的实现方式是简单记录各角色对该对象（及其子对象）的权限，并在今后各角色访问时加以比较，符合权限的则允许相应操作，否则报错返回。更好的实现方式是对相应数据加密，并用密钥来控制权限，如果该角色没有相应密钥就没有对应的权限，这种方式抗攻击能力要更强。具体方案为：

a)对受保护的数据区域(通常为一个子树,对应某对象及其所有子对象)，有一对对应的 PKI 密钥对，用其中的加密密钥对该数据区域进行加密；

b）对具有读权限的角色，授予其解密密钥，该角色可以用该密钥解密该数据区域，从而正确读取这些数据；

c）对具有写权限的角色，将授予其加密密钥，该角色可以将修改后的数据用该密钥加密，从而可以正确写入该区域的数据；

d）鉴于 PKI 的加密／解密效率较低，为提高运行效率，也可以用对称密钥来对该数据区域加密，加密密钥用于对该对称密钥进行加密，解密密钥用于解密经过加密后的密钥数据，从而获得正确的对称密钥。为防止只有读权限的角色在获得对称密钥后用其修改数据，可以用加密密钥来对该数据区域进行数字签名，每次拥有写权限的角色修改该数据区域后都重新做一次签名，从而确保数据不会被没有写权限的角色篡改；

e）当授予某一角色加密密钥或解密密钥时，可以用该角色的公钥对该密钥加密后存储，这样只有拥有该角色的私钥时才能取出该密钥。

以下进一步说明增强系统安全性和文档安全性的技术实施方案：

角色由一个唯一的 ID 号和一对唯一的 PKI 密钥组成，但在角色对象中只存储其 ID 号和公钥，私钥由应用软件掌握。ID 号可以是任意的编号或字符串，只要不同角色都分配了不同的 ID 即可。PKI 算法可以是 ECC、RSA 中的一种。

安全管理功能由角色管理单元、安全会话通道单元、身份认证单元、访问控制单元、签名单元组成。

以某个角色（或多个角色）登录、执行一系列操作、最后注销的整个过程称为会话。会话包括会话标志、登录角色列表。会话可以通过一个安全会话通道进行。安全会话通道有一个会话密钥，用于加密双方之间传递的数据。会话密钥可以用非对称密钥，但一般常用效率更高的对称密钥。

身份认证单元用于当角色登录时，对登录的身份进行认证。身份认证的单位是角色，只有拥有某个角色的私钥才能以这个角色的身份登录。在登录时，身份认证单元根据登录角色的 ID 取出存储在角色对象中的角色公钥，按照前述的“挑战－应答”机制进行认证。

角色管理单元包括角色的创建、各角色的权限的授权、收回授权等。

访问控制单元，用于对文档数据设置访问控制权限，角色只能根据自己的访问控制权限访问文档数据。我们甚至可以连权限数据都可以置于访问控制的管理之下，这样有的角色可以获取其他人的权限，有的角色不能。但只有拥有再授权或收回授权权限的角色才能按照正常的再授权或收回授权方式改变角色的权限，而不允许直接写入权限数据。

以下详细说明各操作步骤：

1. 新建一文档库时，角色管理单元自动将该文档库的缺省角色的权限设置为拥有所有权限，包括对所有对象的读、写、再授权和收回授权权限

2. 建立安全会话通道，启动会话

a）根据会话标志判断是否已经启动会话，如果是，则完成建立安全会话通道的过程，否则继续；

b）一方生成一对随机 PKI 钥对；

c）将公钥发送给对方；

d）对方生成随机对称密钥作为会话密钥，并用该公钥加密会话密钥后传回；

e）用私钥解密出会话密钥；

f）设置会话标志；

g）将登录角色列表设置为缺省角色；

3. 角色登录

a）应用软件提供所要登录角色的 ID 和所登录的文档库；

b）身份认证单元检查会话中的登录角色列表，如果该角色已经登录（包括缺省角色），则该步骤已经完成，否则继续；

c）身份认证单元取出存储在角色对象中的角色公钥；

d）身份认证单元生成一段随机数据块，用该角色的公钥对该数据块进行加密；

e）身份认证单元将加密后的数据块发送给应用软件；

f）应用软件用该角色的私钥进行解密，将解密后的数据发送给身份认证单元；

g）身份认证单元判断传回的数据是否正确，如果不正确则登录失败，否则继续；

h）在会话的登录角色列表中增加该角色。

4. 创建新角色

a）应用软件发出创建新角色指令；

b）角色管理单元生成一个唯一的角色 ID 号；

c）角色管理单元生成一对随机的 PKI 钥对；

d）角色管理单元在文档库中创建一个角色对象，在角色对象中存储上述 ID 号和公钥，该角色的权限为空，即对所有对象都不拥有任何权限；

e）将 ID 号和私钥返回给应用软件。

5. 对角色 R 授与对对象 O 的权限 P

a）应用软件发出授权请求；

b）角色管理单元计算登录角色列表中所有角色对 O 的权限的并集，判断该并集是否是 P 的超集并同时拥有再授权权限。如果否则授权失败（所有角色都加在一起也没有授权所需要的权限），否则继续；

c）角色管理单元将对 O 的权限 P 增加到角色 R 的权限列表中。如果 P 不包含读或写的权限，则授权完成，否则继续；

d）访问控制单元单元检查对象 O 是否已经设置了读写的访问控制权限。如果否，则：

i. 生成随机对称密钥和随机 PKI 密钥

ii. 用对称密钥对 O 进行加密。如果 O 的各级子对象中有已经设置了读写访问控制权限的，则该子对象保持不变

iii. 用 PKI 加密密钥加密对称密钥，存储加密后的密文，并对 O 进行签名

iv. 检查文档库中的所有角色，凡是对 O 具有读权限的（这时 O 是该角色拥有读权限的某个对象的子对象），用该角色的公钥对解密密钥进行加密，将加密后的密文存储到该角色的权限列表中；凡是对 O 具有写权限的（这时 O 是该角色拥有读权限的某个对象的子对象），用该角色的公钥对加密密钥进行加密，将加密后的密文存储到该角色的权限列表中

v. 转到步骤 h

e）从当前登录的角色中，选择对 O 具备相应权限的角色；

f）将该角色权限列表中 O 的对应密钥（读权限对应解密密钥，写权限对应加密密钥，可读可写则包含两个密钥）的密文，发送给应用软件；

g）应用软件用该角色的私钥解密出密钥，返回给访问控制单元；

h）根据 P 的设定，使用目标角色 R 的公钥，加密相应的密钥，生成对应的密文，并存储到 R 的权限列表中。

6. 收回 R 对对象 O 的权限 P

a）应用软件发出收回授权请求；

b）角色管理单元查找登录角色列表中的所有角色，是否有对 O 的收回授权的权限。如果都没有，则收回授权失败，否则继续；

c）从 R 对 O 的权限中去掉 P；

d）如果 P 包含读或写权限，从 R 的权限列表中删除对 O 的相应解密密钥和／或加密密钥。

7. 读取对象 O

a）应用软件发出需要读取 O 的操作的指令；

b）访问控制单元检查登录角色列表中所有角色对 O 的权限，确认是否至少有一个角色对 O 有读权限。如果均无，则失败，否则继续；

c）检查对象 O 是否已经设置了读写的访问控制权限。如果否，则检查其父对象，还不是的话则再检查父对象的父对象，直到找到了设置读写访问控制权限的对象；

d）选择一个对该对象有读权限的角色；

e）将该角色权限列表中保存的该对象的解密密钥的密文，发送给应用软件；

f）应用软件用该角色的私钥解密出解密密钥，返回给访问控制单元；

g）访问控制单元用该解密密钥解密出该对象的对称密钥；

h）用该对称密钥解密出对象 O 的数据；

i）将解密后的数据返回给应用软件。

8. 写对象 O

a）应用软件发出需要修改 O 的操作的指令；

b）访问控制单元检查登录角色列表中所有角色对 O 的权限，确认是否至少有一个角色对 O 有写权限。如果均无，则失败，否则继续；

c）检查对象 O 是否已经设置了读写的访问控制权限。如果否，则检查其父对象，还不是的话则再检查父对象的父对象，直到找到了设置读写访问控制权限的对象 O1；

d）选择一个对 O1 有写权限的角色；

e）将该角色权限列表中保存的 O1 的加密密钥的密文，发送给应用软件；

f）应用软件用该角色的私钥解密出 O1 的加密密钥，返回给访问控制单元；

g）用该加密密钥加密 O 的新数据(如果 O 的各级子对象中有已经设置了读写访问控制权限的，则仍然用其密钥对该子对象加密)；

h）用加密后的数据覆盖原数据，完成写入过程。

9. 对对象 O 进行签名

a）应用软件发出对 O 进行签名的指令；

b）签名单元用前面所述的方法对对象 O 的数据进行正则化；

c）计算正则化结果的 HASH 值；

d）将 HASH 值发给应用软件；

e）应用软件用登录角色列表中所有角色的私钥对该 HASH 值进行加密（即签名）；

f）应用软件将签名结果返回给签名单元；

g）签名单元将签名结果保存在数字签名对象中。

10. 注销登录角色

a）应用软件发出注销某个登录角色的指令；

b）如果登录角色列表中存在该角色，安全会话通道单元将该角色从登录角色列表中去掉。

11. 结束会话

a）一方发出结束会话请求；

b）停止一切与当前会话相关的线程，消除会话标志，删除登录角色列表。

为了提高工作效率，在实施时还可以对上述方法进行增强、简化和变化，例如分解或合并各组成单元、将某个组成单元的某个功能调整为由另一各组成单元来完成、将各角色私钥缓存在会话数据中（会话结束后删除），而不用每次都需要发到应用软件进行解密，或者省略一些安全措施，或者减少一些功能。总之，任何对上述方法进行简化、变化的方法都是本方法的变形。

本发明中所说明的文档安全技术，例如基于角色的权限管理、安全会话通道、角色的认证方式、多重角色登陆、对树结构的正则化技术、细粒度的权限管理单元、基于加密的权限设置等，都不仅适用于本发明所述的文档处理系统，还可以运用于更为广泛的其它应用场合。

在本发明中，为了使本文档处理系统能很好地模拟纸张的特性，提供了一种“只加不改”的技术方案。也就是说，每个应用软件都只在现有文档内容基础上添加新的内容，但不修改、不删除已有的内容，使文档的一个页面就象一张纸一样，可以由不同的人用不同的笔在纸上不断写写画画，但谁都不能修改、删除已有内容。具体方法是每个文档的每一层只由一个应用软件来管理和维护，即每一个应用软件在编辑其它软件生成的文档时，都在现有文档基础上新增加一层，将本软件新编辑的内容都放到这一层中，不修改和删除前面各层的内容。由于现有社会就是基于纸张来运转的，因此只要能符

合纸张的特性就能满足现有应用的需求，具备足够的实用价值。

为了确保每一层内容在生成后没有被修改、删除，我们可以利用每一层的数字签名对象。数字签名可以是对本层内容进行签名，更可以是对本层以及本层下面（即更早创建的）的所有层的内容一起签名。签名以后并不妨碍对文档做进一步的批注等编辑，只要新的内容是位于新建的层，没有修改破坏签名时存在的各层，签名依然是有效的，但签名者只对签名以前的内容负责，不对签名以后的内容负责。这是一个非常符合应用需求的技术方案，具有很大的实用价值。相比之下，现有的其它技术或者签名后不允许编辑，或者编辑后（尽管是“只加不改”的编辑）签名被破坏。

前述技术方案不允许修改文档中的已有内容，即使不考虑与纸张特性的兼容以及数字签名问题，需要修改的话也只能做版面级编辑，即对每个版面对象的编辑（增、删、改）都不会改变其它版面对象（这是由于通用文档模型是基于可见部分为基础构建的，不包含大量不可见的、关于版面对象之间的关系，因此修改任何一个版面对象时，其它版面对象不会产生相应的调整，例如删掉一个字，就会在其位置留下空白，右边的文字不会自动左移）。如果用户需要对文档中的已有内容进行编辑，并且还希望能像在原来那样编辑的话，有一个技术方案可以很好地满足这个应用需求。该方案是当应用软件完成初始编辑时，除了新建一层存放当前编辑的内容外，还将源文件（按照应用软件自有的格式存储，记录了各对象之间完整关系的文件，例如.doc 文件）嵌入到文档中。当下次需要进行继续编辑时，从文档中取出该源文件，并使用该源文件继续编辑。编辑完成后清除该软件所管理的那一层，重新生成该层的内容，并继续将新修改的源文件嵌入到文档中。

具体方法如下：

1. 应用软件第一次处理该文档时，新建一层，将新编辑内容对应的版面对象插入到新建层中，同时用自身格式另存一份新编辑的内容（即源文件）；

2. 在文档对象中新建一个源文件子对象，用来嵌入源文件（例如用二进制数据的方式整体嵌入），并记录是哪一层对应该源文件对象；

3. 用同一应用软件再次编辑该文档时，从对应的源文件对象中取出对应的源文件；

4. 使用该源文件继续编辑该层内容。由于该源文件是该应用软件自身的格式，可以按照该应用软件自身的功能继续对该层内容进行编辑；

5. 再次编辑结束后，根据新编辑后的结果更新该层内容（例如用全部清除后全部重新生成的方式），同时将新修改后的源文件重新嵌入到文档对象中；

6. 如此循环往复，就可以用原有应用软件按照原有方式对文档中的已有内容进行编辑。

采用上述技术方案，可以最大程度地实现文档的互操作性。在应用软件、文档都采用本发明技术时，可以实现（如果有足够安全权限的话）：

1. 对任何文档，用任何应用软件都可以正确打开、显示、打印；

2. 对任何文档，用任何应用软件都可以新添加任何内容，而且不会破坏文档已有签名；

3. 对任何文档，在不必考虑文档已有签名（没有签名或者虽有签名但允许破坏）的前提下，用任何应用软件都可以对文档已有内容进行版面级编辑；

4. 对任何文档，使用文档已有内容的原始编辑软件可以对该内容进行正常编辑。

由此可见，通过本发明中对层的管理，对文档的管理、互操作、安全设置都带来极大的便利。

下面我们以A软件创建一个文档并且B软件对其进行编辑为例说明其工作过程。为了节约篇幅起见，在本例中我们选用UOI作为接口标准：

1. A软件发出指令，创建文档库c:\sample\mydocbase.sep，将其句柄存放在hDocBase：

```
UOI_Open("c:\\sample\\mydocbase.sep", TRUE, &hDocBase);
```

文档库系统执行该指令，创建一个文件名为c:\sample\mydocbase.sep的新的XML文件，其根结点为文档库对象，将其句柄返回给应用软件。

2. A软件发出指令，在文档库hDocBase中新建文档集，将其句柄存放在hDocSet：

```
hDocSet = InsertNewObj(hDocBase, 0, UOI_Obj:: TYPE_DOCSET);
```

文档库系统执行该指令，在文档库对象结点下添加一个文档集子结点，，

将其句柄返回给应用软件。

3. A 软件发出指令，在文档集 hDocBase 中新建文档，将其句柄存放在 hDoc：

```
hDoc = InsertNewObj(hDocSet, 0, UOI_Obj:: TYPE_DOC);
```

文档库系统执行该指令，在 hDocBase 对应的结点下添加一个文档结点，将其句柄返回给应用软件。

4. A 软件发出指令，在文档 hDoc 中新建一页，版心大小是宽 w，高 h，将其句柄存放在 hPage：

```
UOI_Page     page;
page.size.w = w;
page.size.h = h;
UOI_Insert(hDoc, 0, &page, &hPage);
```

文档库系统执行该指令，在 hDoc 对应的结点下添加一个页结点，其版心属性按宽 w 高 h 设置，将其句柄返回给应用软件。

5. A 软件发出指令，在页 hPage 中创建一层，将其句柄存放在 hLayer：

```
hLayer = InertNewObj(hPage, 0, UOI_Obj::TYPE_LAYER);
```

文档库系统执行该指令，在 hPage 对应的结点下添加一个层结点，将其句柄返回给应用软件。

6. A 软件发出指令， 设置字号为 s：

```
UOI_CharSize      charSize;
charSize.m_Width = charSize.m_Height = s;
UOI_Insert(hLayer, 0, &charSize);
```

文档库系统执行该指令，在 hLayer 对应的结点下添加一个字号结点，其宽高属性都设置为 s，将其句柄返回给应用软件。

7. A 软件发出指令，在坐标（x1, y1）位置插入文字串“书生意气挥斥方遒”：

```
UOI_Text      text;
text.m_pText = Duplicate("书生意气挥斥方遒");
text.m_Encoding = UOI_Text:: ENCODE_GB13000;
```

```
text.m_Start.x = x1;//起点坐标的 X 坐标
text.m_Start.y = y1;// 起点坐标的 Y 坐标
UOI_Insert(hLayer, 1, &text);
```

文档库系统执行该指令，在 hLayer 对应的结点下再添加一个文字子结点，其文字数据为“书生意气挥斥方遒”，其字符编码方式为 GB13000，文字起点为（x1, y1），并将其句柄返回给应用软件。

8. A 软件发出指令，关闭文档库 hDocBase：

```
UOI_Close(NULL, hDocBase);
```

文档库系统执行该指令，关闭文件 c:\sample\mydocbase.sep。

9. B 软件发出指令，打开文档库 c:\sample\mydocbase.sep, 将其句柄存放在 hDocBase：

```
UOI_Open("c:\\sample\\mydocbase.sep", FALSE, &hDocBase);
```

文档库系统执行该指令，打开文件 c:\sample\mydocbase.sep。

10. B 软件发出指令，获取文档库 hDocBase 第一个文档集的指针，将其句柄存放在 hDocSet：

```
UOI_GetHandle(hDocBase, 0, &hDocSet);
```

文档库系统执行该指令，寻找根结点下第一个类型为文档集的子结点，将其句柄返回给应用软件。

11. B 软件发出指令，获取文档集 hDocSet 第一个文档的指针，将其句柄存放在 hDoc：

```
UOI_GetHandle(hDocSet, 0, &hDoc);
```

文档库系统执行该指令，寻找 hDocSet 对应的结点下第一个类型为文档的子结点，将其句柄返回给应用软件。

12. B 软件发出指令，获取文档 hDoc 第一页的指针，将其句柄存放在 hPage：

```
UOI_GetHandle(hDoc, 0, &hPage);
```

文档库系统执行该指令，寻找 hDoc 对应的结点下第一个类型为页的子结点，将其句柄返回给应用软件。

13. B 软件获取该页版面位图，用于显示该页

```
UOI_GetPageBmp(hPage, rect, buf);
```

文档库系统执行该指令，按照前述方法（文档库系统实施例说明第 9 条）生成 hPage 对应的页的版面位图，将其返回给应用软件。

14. B 软件发出指令，获取 hPage 第一层的指针，将其句柄存放在 hLayer：

```
UOI_GetHandle(hPage, 0, &hLayer);
```

文档库系统执行该指令，寻找 hPage 对应的结点下第一个类型为层的子结点，将其句柄返回给应用软件。

15. B 软件发出指令，获取第一个版面对象的句柄 hObj:

```
UOI_GetHandle(hLayer, 0, &hObj);
```

文档库系统执行该指令，寻找 hDocSet 对应的结点下第一个子结点，将其句柄返回给应用软件。

16. B 软件发出指令，获取 hObj 的类型：

```
UOI_GetObjType(hObj, &type);
```

文档库系统执行该指令，获取 hObj 对应的结点的类型，将其返回给应用软件。

17. B 软件发现这是一个字号对象，获取该对象：

```
UOI_GetObj(hObj, &charSize);
```

文档库系统执行该指令，获取 hObj 对应的结点的数据，将其返回给应用软件。

18. B 软件将字高放大一倍：

```
charSize.m_Height *= 2;
UOI_SetObj(hObj, &charSize);
```

文档库系统执行该指令，设置 hObj 对应的结点的属性。

19. B 软件重新获取版面位图并显示，这时会发现屏幕上的“书生意气挥斥方遒”变成长体字了。

下面，参照图 18 描述依照本发明的文档操作系统执行一操作的一个例子。在该例子中，应用软件通过统一的接口标准（UOML 接口）请求对文档的操作。文档库系统可能会有不同厂商的不同型号，但是对于应用开发厂商来说面向的都是同一个接口标准，因此都可以与之配套使用。

在本发明中，不同的应用软件可以同时或不同时调用同一个文档库系统，同一应用软件可以同时或不同时调用不同的文档库系统。

依照本发明，使得应用层和数据处理层分离，使得同一文档能在不同的应用软件之间通用，使不同应用软件之间具有良好的文档互操作性。

依照本发明，形成产业分工，减少重复开发，并更加专业、完备、正确；对文档的基本操作都在文档库系统中处理，各应用软件不必重复开发。而且由于文档库系统是由专业厂商开发，相关技术的专业性、完备性、正确性较有保障，而且应用软件厂商和用户可以选择做的最好的一家文档库系统厂商，从而保证处理效果的正确性和一致性。

依照本发明，提供多文档甚至海量文档的管理机制，使文档之间能够有效组织起来，便于检索、查询、保管，便于嵌入较强的信息安全机制。

依照本发明，提供更好的安全机制，可以设置多种角色，细粒度地设置每个角色的权限。其中细粒度是双重的，一方面可以对整个文档或文档的一个细微之处进行权限设置，另一方面可以设置种类非常多的权限，而不仅仅是传统的读／写／不可访问三级。

依照本发明，鼓励创新，合理竞争。形成合理的产业分工后，各文档库系统厂商和各应用软件厂商就会在领域展开竞争，而不会再出现 Microsoft Word 一样靠文档格式来垄断应用软件的情形发生。各文档库系统厂商也可以在标准之外增加新的功能以吸引用户，标准并不会对创新形成束缚。

依照本发明，便于优化性能，有更好的可移植性和可伸缩性。无论是什么平台，什么样的性能，都可以遵循同样的调用接口，使得在不改变接口标准的情况下可以不断优化性能，并移植到不同的平台。

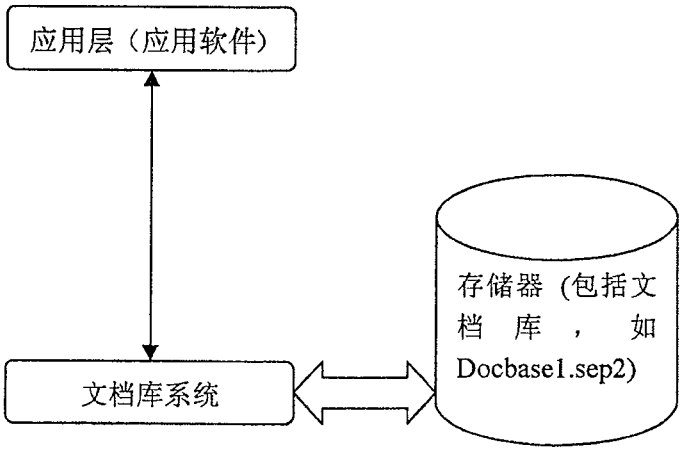


图 1

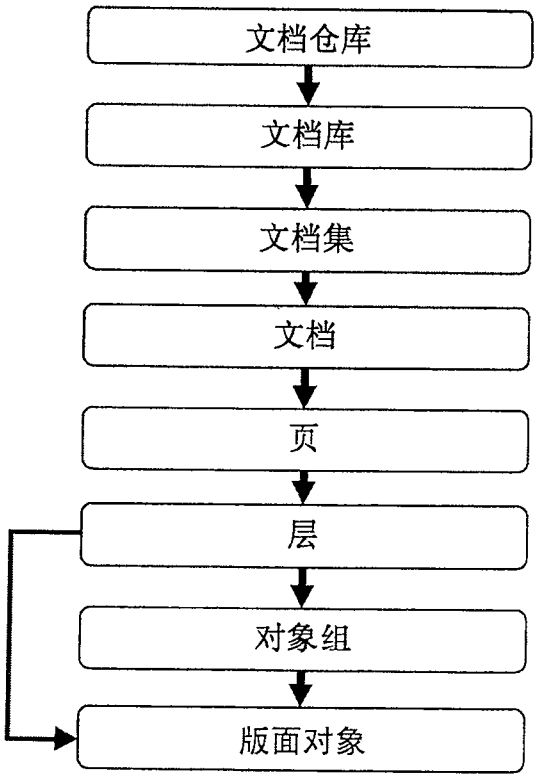


图 2

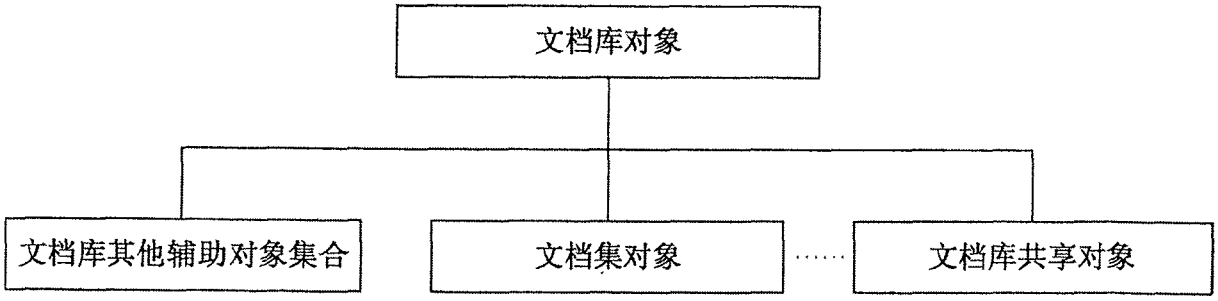

文档库对象
文档库其他辅助对象集合
文档集对象
……
文档库共享对象

图 3

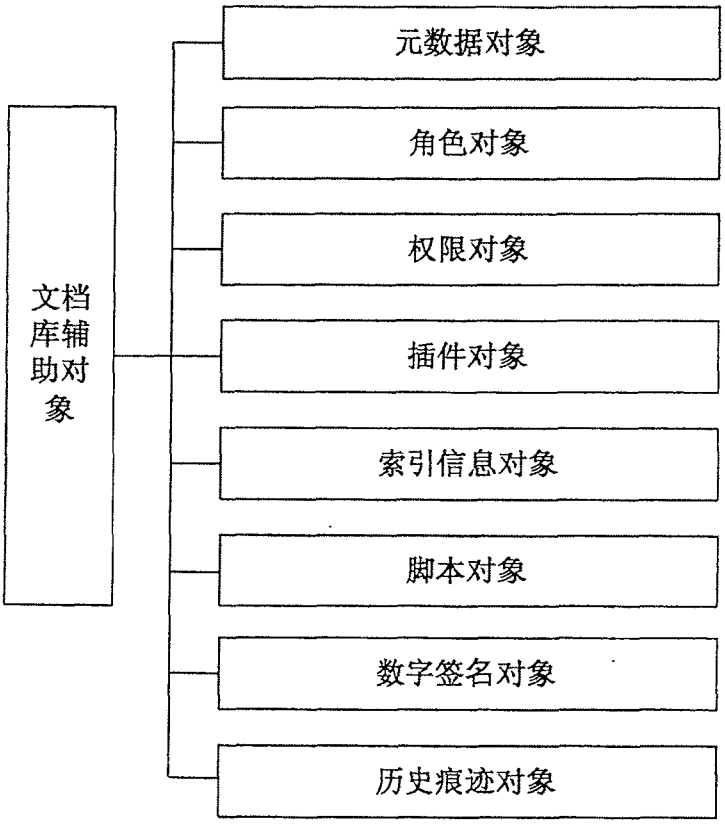

文档库辅助对象
元数据对象
角色对象
权限对象
插件对象
索引信息对象
脚本对象
数字签名对象
历史痕迹对象

图 4

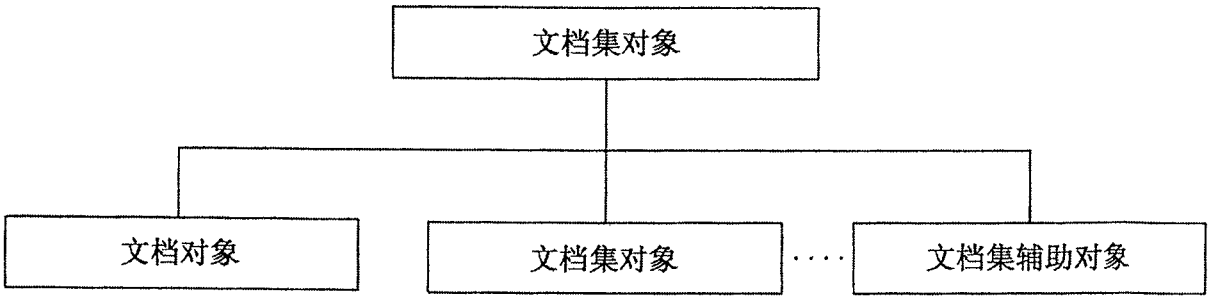


图 5

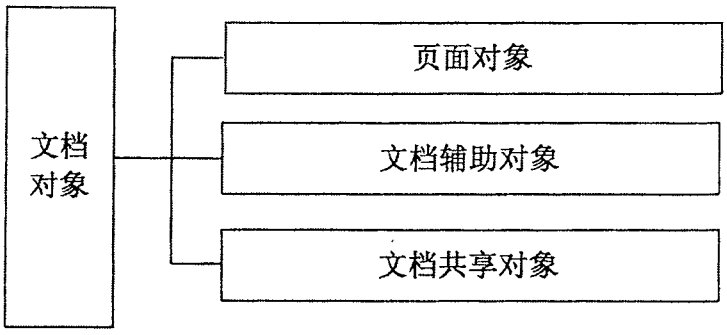


图 6

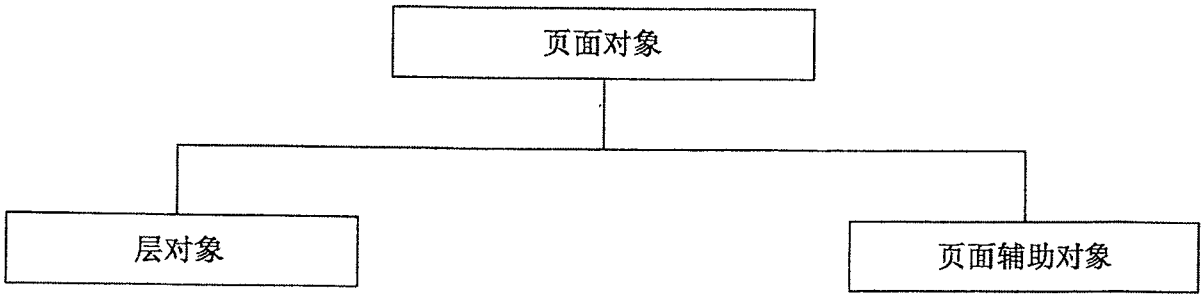


图 7

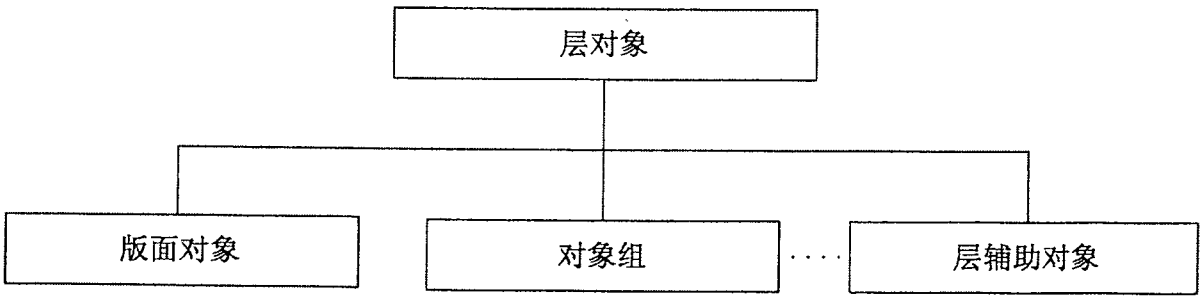


图 8

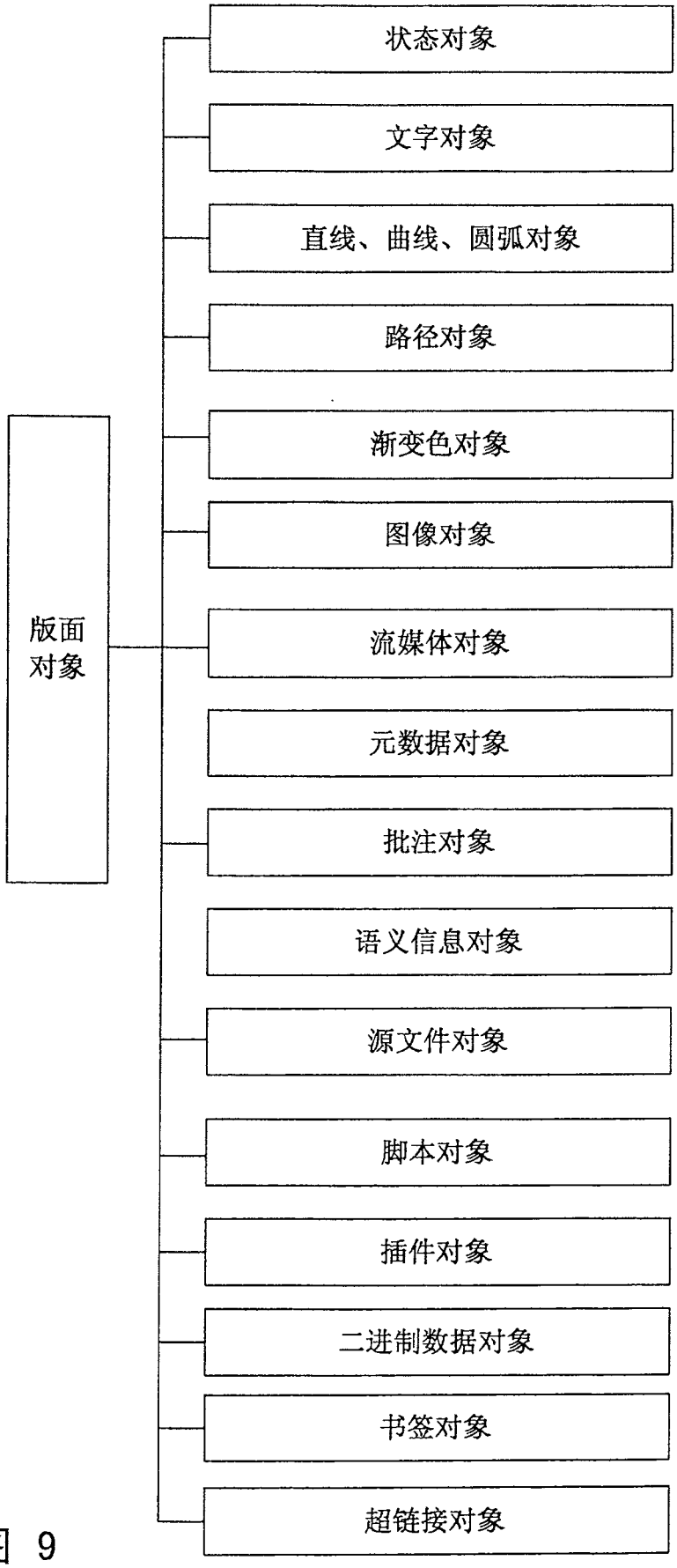

版面对象
状态对象
文字对象
直线、曲线、圆弧对象
路径对象
渐变色对象
图像对象
流媒体对象
元数据对象
批注对象
语义信息对象
源文件对象
脚本对象
插件对象
二进制数据对象
书签对象
超链接对象

图 9

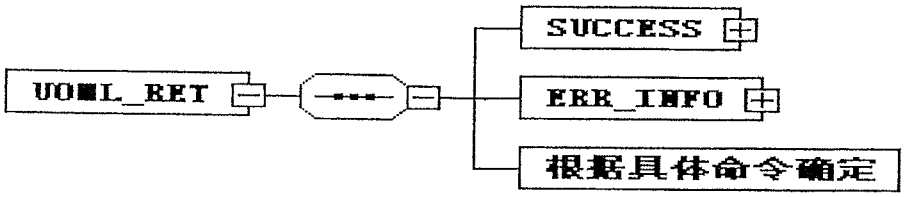

UOML_RET
SUCCESS
ERR_INFO
根据具体命令确定

图 10

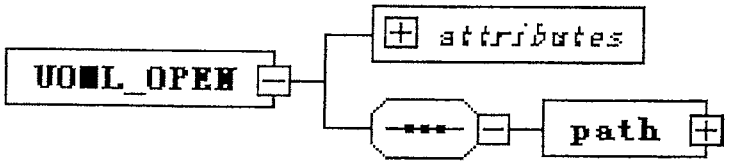

UOML_OPEN
attributes
path

图 11

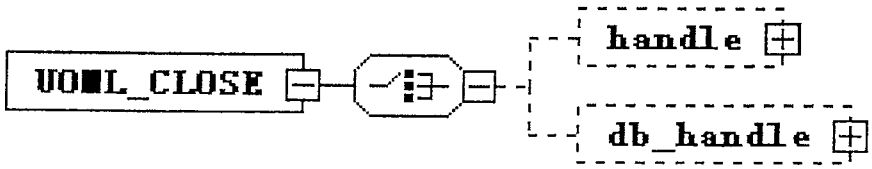

UOML_CLOSE
handle
db_handle

图 12

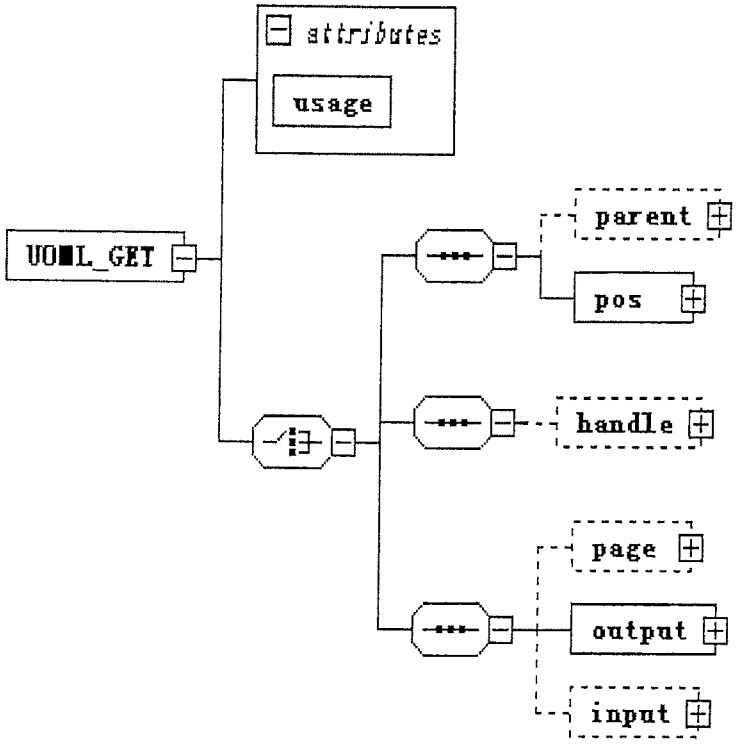

attributes
usage
UOML_GET
parent
pos
handle
page
output
input

图 13

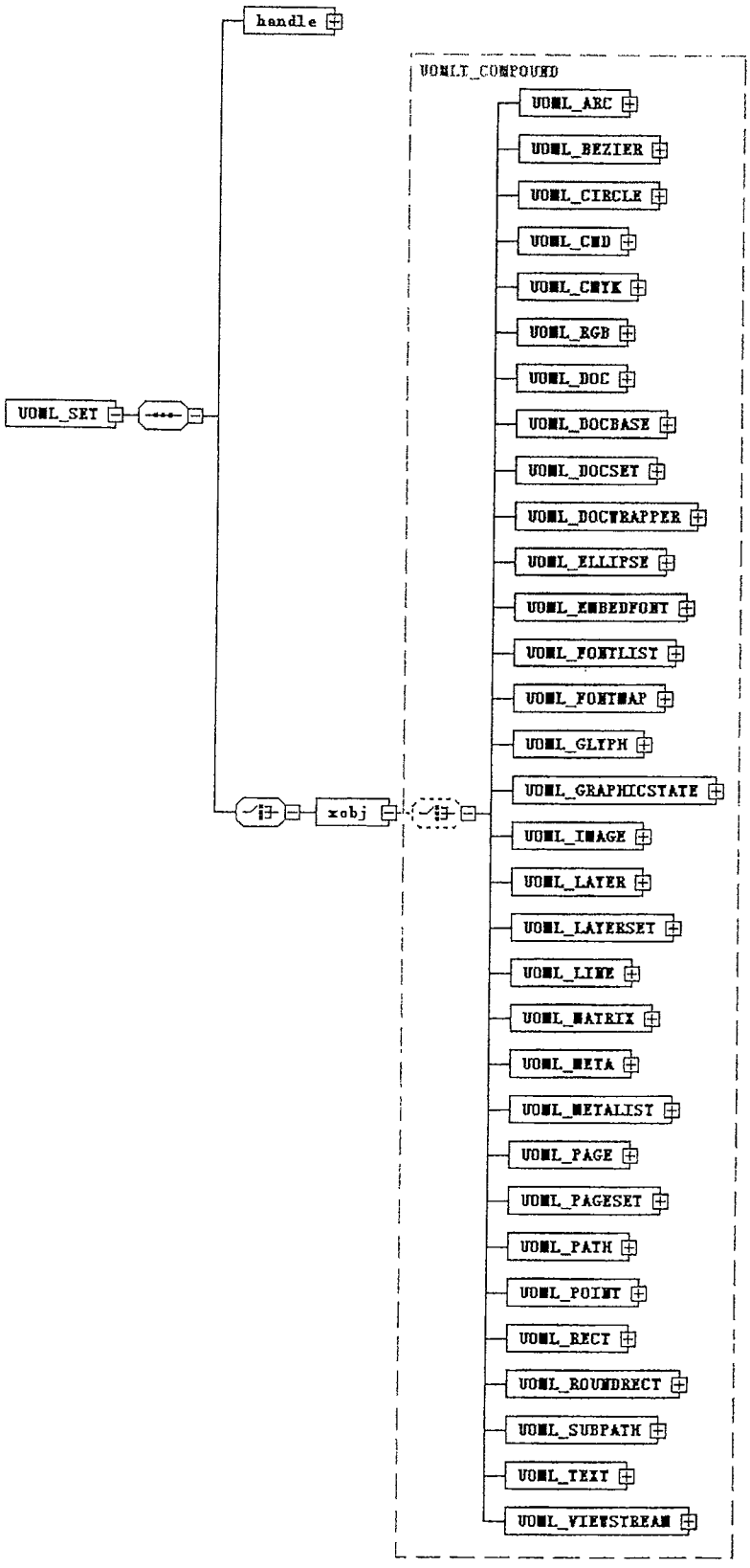

handle
VOMLT_COMPOUND
VOML_ARC
VOML_BEZIER
VOML_CIRCLE
VOML_CMD
VOML_CMYK
VOML_RGB
VOML_DOC
VOML_SET
VOML_DOCBASE
VOML_DOCSET
VOML_DOCWRAPPER
VOML_ELLIPSE
VOML_EMBEDFONT
VOML_FONTLIST
VOML_FONTMAP
VOML_GLYPH
VOML_GRAPHICSTATE
xobj
VOML_IMAGE
VOML_LAYER
VOML_LAYERSET
VOML_LINE
VOML_MATRIX
VOML_META
VOML_METALIST
VOML_PAGE
VOML_PAGESET
VOML_PATH
VOML_POINT
VOML_RECT
VOML_ROUNDRECT
VOML_SUBPATH
VOML_TEXT
VOML_VIEWSTREAM

图 14

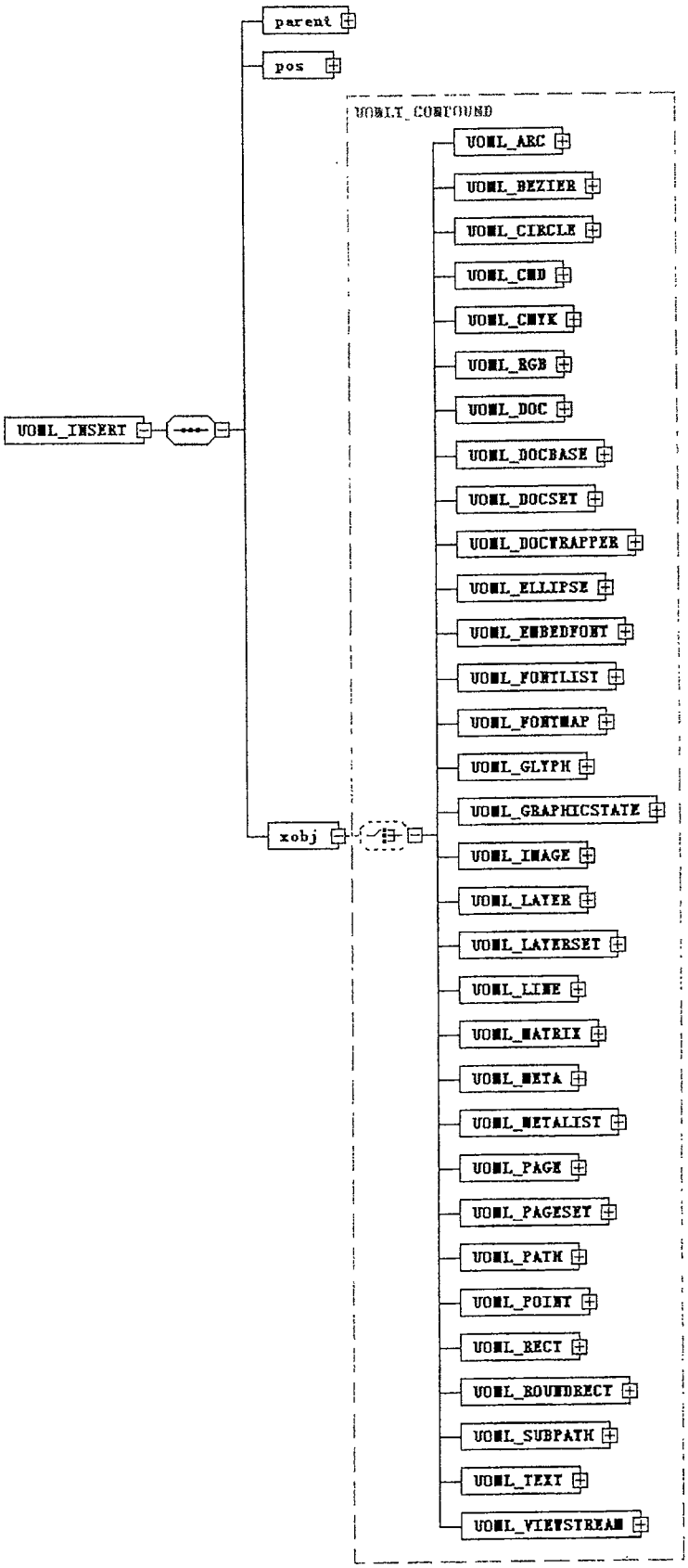

parent
pos
VOML_COMPOUND
VOML_ARC
VOML_BEZIER
VOML_CIRCLE
VOML_CMD
VOML_CMYK
VOML_RGB
VOML_DOC
VOML_DOCBASE
VOML_DOCSET
VOML_DOCWRAPPER
VOML_ELLIPSE
VOML_EMBEDFONT
VOML_FONTLIST
VOML_FONTMAP
VOML_GLYPH
VOML_GRAPHICSTATE
VOML_IMAGE
VOML_LAYER
VOML_LAYERSET
VOML_LINE
VOML_MATRIX
VOML_META
VOML_METALIST
VOML_PAGE
VOML_PAGESET
VOML_PATH
VOML_POINT
VOML_RECT
VOML_ROUNDRECT
VOML_SUBPATH
VOML_TEXT
VOML_VIEWSTREAM
VOML_INSERT
xobj

图 15

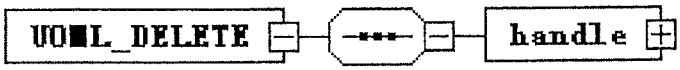

UOML_DELETE
handle

图 16

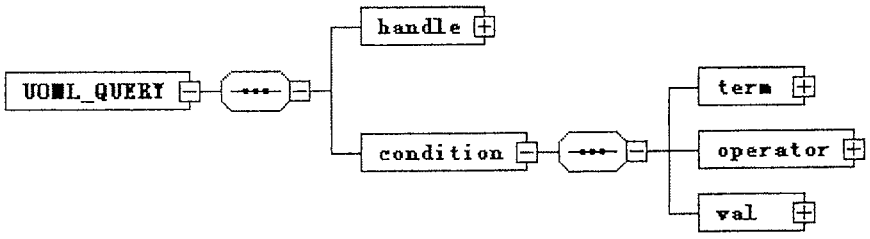

UOML_QUERY
handle
condition
term
operator
val

图 17

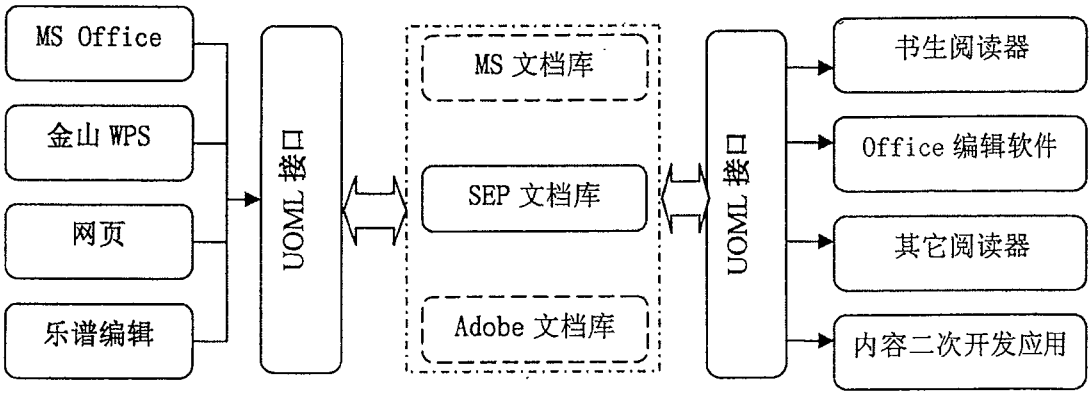

MS Office
金山 WPS
网页
乐谱编辑
UOML 接口
MS 文档库
SEP 文档库
Adobe 文档库
UOML 接口
书生阅读器
Office 编辑软件
其它阅读器
内容二次开发应用

图 18