[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 9/44

[12] 发明专利申请公开说明书

[21] 申请号 200510008280.1

[43] 公开日 2005 年 8 月 24 日

[11] 公开号 CN 1658159A

[22] 申请日 2005.2.21
[21] 申请号 200510008280.1
[30] 优先权
[32] 2004.2.19 [33] DE [31] 102004008258.8
[71] 申请人 西门子公司
地址 联邦德国慕尼黑
[72] 发明人 卡尔海因茨·多恩
汉斯-马丁·冯斯托克豪森

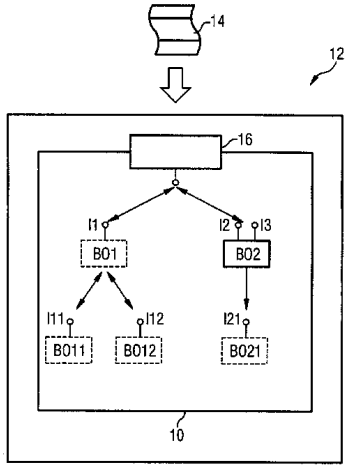
[74] 专利代理机构 北京市柳沈律师事务所
代理人 马 莹 邵亚丽

权利要求书 3 页 说明书 6 页 附图 2 页

[54] 发明名称 可配置和可动态更改的对象模型

[57] 摘要

本发明涉及一种用于在面向对象的环境中为应用(A)产生对象模型(10)的方法和系统，其中，该系统包括至少一个读取配置文件(14)、并从该配置文件中自动产生对象模型(10)的通用组件(12)。此外该通用组件(12)还包括用于管理所产生的对象模型(10)的元素的知识库(16)。



ISSN1008-4274

1. 一种用于在面向对象的环境中为应用（A）或应用（A）的组件产生对象模型（10）的方法，其特征在于，该方法访问读取配置文件（14）、并从该配置文件中自动产生对象模型（10）的通用组件（12）。

2. 根据权利要求 1 所述的方法，其特征在于，所述通用组件（12）独立于，优选在语义上独立于所述应用（A）和/或应用组件（A）。

3. 根据权利要求 1 或 2 所述的方法，其特征在于，所述实施对象不必单独地，或者不必特定于组件地添加到所述对象模型（10）中。

4. 根据上述权利要求之一所述的方法，其特征在于，所述对象模型（10）是层次结构的。

5. 根据上述权利要求之一所述的方法，其特征在于，所述对象模型（10）可以动态配置，优选也可以在运行时动态配置，从而尤其是可以从所产生的对象模型（10）中删除和/或添加对象。

6. 根据上述权利要求之一所述的方法，其特征在于，所述对象模型（10）支持封装原理。

7. 根据上述权利要求之一所述的方法，其特征在于，所述配置文件（14）是 XML 文件。

8. 根据上述权利要求之一所述的方法，其特征在于，该方法访问至少一个知识库（16），该知识库（16）优选由所述对象模型（10）管理，并至少为管理涉及相应对象的实例和接口而确定。

9. 根据上述权利要求之一所述的方法，其特征在于，该方法还包括如下方法步骤：

- 初始化所产生的对象模型（10）的对象。

10. 根据权利要求 9 所述的方法，其特征在于，初始化对象的顺序对于所述对象模型（10）来说是从里到外进行的，而对于同一层级的对象来说是按照配置的顺序进行的。

11. 根据上述权利要求之一所述的方法，其特征在于，通过访问更改例程或者通过读取更改配置文件，可以更改所述对象模型（10）。

12. 一种用于在面向对象的环境中为应用（A）或应用（A）的组件产生

对象模型（10）的系统，其特征在于，该系统包括:

- 至少一个通用组件（12），其读取配置文件（14）并从该配置文件（14）中自动产生对象模型（10）。

13. 根据权利要求 12 所述的系统，其特征在于，所述通用组件（12）独立于，优选在语义上独立于所述应用（A）和/或应用组件（A）。

14. 根据权利要求 12 或 13 所述的系统，其特征在于，所述实施对象不必单独地，或者不必特定于组件地添加到所述对象模型（10）中。

15. 根据权利要求 12 至 14 中任一项所述的系统，其特征在于，所述对象模型（10）是层次结构的。

16. 根据权利要求 12 至 15 中任一项所述的系统，其特征在于，所述对象模型（10）可以动态配置，优选也可以在运行时动态配置，从而尤其是可以从所产生的对象模型（10）中删除和/或添加对象。

17. 根据权利要求 12 至 16 中任一项所述的系统，其特征在于，所述对象模型（10）支持封装原理。

18. 根据权利要求 12 至 17 中任一项所述的系统，其特征在于，所述配置文件（14）是 XML 文件。

19. 根据权利要求 12 至 18 中任一项所述的系统，其特征在于，该系统还包括:

- 至少一个知识库（16），该知识库（16）优选由所述对象模型（10）管理，并至少为管理涉及相应对象的实例和接口而确定。

20. 根据权利要求 12 至 19 中任一项所述的系统，其特征在于，该系统还包括:

- 至少一个初始化单元，其为初始化所产生的对象模型（10）的对象而确定。

21. 根据权利要求 20 所述的系统，其特征在于，初始化对象的顺序对于所述对象模型（10）来说是从里到外进行的，而对于同一层级的对象来说是按照配置的顺序进行的。

22. 根据权利要求 12 至 21 中任一项所述的系统，其特征在于，通过访问更改例程或者通过读取更改配置文件，可以更改所述对象模型（10）。

23. 一种在面向对象的环境中用于应用或应用组件的对象模型，其特征在于，所述对象模型可以通过访问读取配置文件的通用组件自动产生。

24. 根据权利要求 23 所述的对象模型，其特征在于，该对象模型（10）可动态配置和/或至少就所产生的对象来说可更换和/或可扩展。

可配置和可动态更改的对象模型

技术领域

本发明涉及面向对象地开发基于软件的系统的领域，并尤其是涉及一种用于为应用或应用组件产生对象模型的方法。

背景技术

与诸如 LISP、PROLOG 的面向问题的函数和逻辑编程语言相反，诸如 JAVA、C++的面向对象的编程语言从建立问题的元素、即对象及其在解空间中的表示出发。该应用不限于特定的问题类型。面向对象的规则由于其高度的抽象性而对问题的解决提供了广泛的使用领域。

面向对象的范例建立在由问题的解空间构成的抽象结构之上，所述问题的解空间即是通过特征（或者说是属性）以及特性（或者说是方法）表现的对象。

面向对象编程的基本原理称为封装。该原理说明来自其它类的方法只能通过对象方法来访问属性。由此可以定义谁或者说哪些实例可以访问所定义的对象属性和方法，而谁不可以访问。这些属性和方法例如可以是“public”（公用）并可用于所有对象，或者是“private”（专有）并只允许在对象内部使用。

另一个基本原理称为继承。这意味着通过继承其它对象的方法和属性可以定义基于其它对象的对象。由此出现了类的树结构。

在为不同的应用开发软件组件时，尤其是在对软件组件进行面向对象的编程时，总是在各组件之间形成交叉区域，它们对问题的解的结构要求是类似的，而迄今为止对它们仍须单独实现。这导致由现有技术公知的解决方案产生的冗余的缺陷。

此外还需要基于软件的系统可以灵活地对所需的更改做出反应，同时不会降低产品的质量。目前需要在匹配系统时，必须通过将最初实施的组件替换成新的匹配实施的组件来具体地匹配对象模型。

对象模型组件的可更换性到目前为止还必须在该对象模型中自行解决。这导致很高的开发和测试费用，因为为了加载被更改的模块、产生更改的对象或

对象实例并将这些对象或对象实例添加到对象模型中，始终都需要单独匹配并且必须实现单独的编码。由于这些更改又需要重新测试整个系统。

发明内容

从上述现有技术出发，本发明要解决的技术问题在于提供一种途径，可以动态配置对象模型并且可以在运行时更换和扩展对象模型的对象，而无需更改已实现的对象模型。

该技术问题是通过本发明的特征解决的，并尤其是通过一种用于在面向对象的环境中为应用或应用组件产生对象模型的方法和系统来解决的，其中该方法访问读取配置文件的通用组件，并从中产生对象模型。

另一种解决技术问题的方案是设置了一种在面向对象的环境中用于应用或应用的组件的对象模型，该模型访问本身读取配置文件的通用组件，其中由于对该配置文件的处理而产生该对象模型。

根据本发明的解决方案在于，还可以在实际设计之后进行对象模型的更改和/或匹配。由此可以在运行时更改对象模型。

更改代码的单独实施不再需要。只是从该对象模型中删除单个对象或添加单个对象。

重要的是，根据本发明的通用组件完全独立于所述应用和/或应用组件。首先，通用组件不包含涉及应用的语义，并因此在语义上独立于待解决的各个应用问题和/或独立于待产生的对象模型。通用组件基于一种普遍适用的机制，该机制这样由问题抽象而来，即该机制可以用于不同的应用组件。

这使得可以在对对象模型进行更改时不必实施单独的代码。

为了该实施或者说是该实施对象而产生的对象不必单独地，或者说是不必特定于组件地添加到该对象模型中。通用组件在起始时刻读取配置文件，并访问至少一个本身由对象模型管理的知识库。该知识库用于管理各对象的实例和接口。这可以显著简化开发并明显减少开发费用。

根据本发明产生的对象模型优选为层次结构的。一个对象可以访问所有位于该对象所在层或者位于其以下层的接口。由此产生了这样的优点，即根据本发明的解决方案支持封装原理。但是也可以采用另一种实施方式，即不考虑封装原理。

本发明解决方案的另一优点在于，对象模型是可动态可配置的，从而尤其

是可以从所产生的对象模型中删除和移去对象。这甚至可以在运行时进行。由此可以非常灵活地对更改要求做出反应。

为了使根据本发明的系统，尤其是使通用组件能产生对象模型，该系统/通用组件读取配置文件。该文件优选包含按照可扩展标记语言（XML）形式的说明。但是其它实施方式也可以利用其它语言，例如基于 HTML 的语法。

由通用组件读取的配置文件至少包括涉及以下数据组的数据:

- 模块名称，
- 待产生的对象实例的名称，
- 至少通过对象相互之间的关系可见的对象实例和/或数据之间的接口。

对象以及必要时对象的属性和方法优选自动地由通用组件产生。根据本发明，这在访问配置文件的情况下进行。但是在另一实施方式中也可以让应用者介入该过程，对对象模型的产生进行控制地监控和/或半自动地实现对象模型的产生。

本发明的方法还包括另一方法步骤，即初始化所产生的对象模型的对象。在此可以设置是否应当初始化所有对象以及如果不能则初始化哪些对象。

初始化对象的顺序对于对象模型来说是从里到外进行，而对于同一层级的对象来说是按照配置的顺序进行的。由此初始化可以通过采用继承的概念而半自动地进行。

通过访问更改例程或者通过除了初始配置文件之外还读取另一个配置文件、即更改配置文件，可以完成必要的更改。这些更改不必是单独的，也就是说不是针对每个组件单独进行。

除了通用组件之外还有一个重要元素就是为管理对象模型的元素或模块的对象知识库。在此也就是管理对象实例和可见的接口。在本发明的一个更为复杂的实施方式中，在对象知识库中还管理得更多，例如通过类结构的其它接口和/或数据、消息等。在优选实施方式中设置了更多的对象知识库。在该优选实施方式中，该对象知识库是通用组件的组成部分，并由该通用组件管理。

上述根据本发明方法的实施方式还可以构成为计算机程序产品，具有计算机可读介质和计算机程序以及所属的程序代码块，其中计算机在加载该计算机程序之后可以实施上述根据本发明的方法。

另一解决方案是提供一种存储介质，其用于存储上述利用计算机实施的方法，并且可由计算机读取。

附图说明

在以下对附图的详细描述中，借助附图非限制性地描述具有其特征和其它优点的实施例。在附图中示出：

图 1 示出根据本发明产生的对象模型的例子，以及

图 2 示出对根据本发明方法的原理流程和根据本发明的系统的重要元素的概览。

具体实施方式

根据本发明，用 10 一般性地表示的对象模型是为应用 A 或为应用 A 的一部分或者说一个组件而产生的。其中采用通用组件 12，其访问通用的机制并由此可用于不同的应用 A 和/或所属的对象模型 10。

如图 2 所示，通用组件 12 在起始时刻读取配置文件 14。该配置文件 14 优选具有 XML 文件的形式。在配置文件 14 中存储着模块的名称、待产生的对象实例的名称及其通过对象相互之间的关系的可见接口和数据。在此尤其是要管理可以从哪些对象出发访问哪些接口以及不能访问哪些接口。

在读取和处理这些数据之后，通用组件 12 产生对象模型 10。通用组件 12 还包括知识库 16。该知识库 16 可以物理地设置在通用组件内。或者还可以将知识库 16 设置在通用组件之外并具有可访问途经。

知识库 16 用于管理对象模型 10、尤其是每个对象的实例和可见接口。在对象模型 10 内，每个对象都获得一个唯一的逻辑名，知识库 16 可以通过该逻辑名对该对象进行响应。同一个类可以在不同的逻辑名下分别用不同的接口进行配置。

在图 1 中示出根据本发明产生对象模型 10 的例子。从只示例性示出的配置文件 14 中产生通用组件 12 内的对象模型 10。由对象登记的接口可由起源相同的各对象以及源对象可见。所有对象的起源都由对象知识库 16 形成。对象知识库 16 由通用组件 12 管理。

在抽象层面上，XML 配置文件 14 可以如下所示，以产生如图 1 所示的对象模型：

<APPLICATION> <BUSINESS_OBJECT LOGID="BO1" ...>

<INTERFACES> <INTERFACE TYPE= "IBO1" />

```
</INTERFACES>                    <BUSINESS_OBJECT LOGID= “BO11”
...>                                          <INTERFACES>
                                  <INTERFACE TYPE= “IBO11”/>
                                          </ INTERFACES>
                                      </BUSINESS_OBJECT>
                          <BUSINESS_OBJECT LOGID= “BO12” ...>
                                          <INTERFACES>
                                   <INTERFACE TYPE= “IBO12” />
                                          </ INTERFACES>
                                      </BUSINESS_OBJECT>
                                      </BUSINESS_OBJECT>
                         <BUSINESS_OBJECT LOGID=”BO2” ...>
                                          <INTERFACES>
                                  <INTERFACE TYPE= “IBO2”/>
                                  <INTERFACE TYPE= “IBO3”/>
                                          </INTERFACES>
                          <BUSINESS_OBJECT LOGID= “BO21” ...>
                                          <INTERFACES>
                                  <INTERFACE TYPE= “IBO21”/>
                                          </ INTERFACES>
            </BUSINESS_OBJECT>
      </ BUSINESS_OBJECT>
</APPLICATION>
```

利用注释和为实现基于.NET 的方法和/或系统读取 XML 代码，该代码产生图 1 中示出的对象模型 10:

```
<?xml version “l. 0” ?><APPLICATION>          <!--- 用逻辑名称、类型（类名）和包含商业对象的DLL来定义商业对象 -->
 <BUSINESS_OBJECT LOGID= “BO1” TYPE= “MyBussinessObjSpace. BO1”
                  ASSEMBLY= “MyBussinessObj1 .dll”>
                  <!--- 由对象实现并应当在对象模型中可见的接口 -->
                  <INTERFACES>
```

```
                    <INTERFACE TYPE= "IBO1" />
                </INTERFACES>
                <!--- 可封装的商业对象（在父对象之外不可见）-->
            <BUSINESS_OBJECT LOGID "BO11"
TYPE= "MyBussinessObjSpace. BO11"
                    ASSEMBLY ="MyBussinessObj11 .dll">
                    <INTERFACES>
                        <INTERFACE TYPE= "IBO11"/>
                    </INTERFACES>
                </BUSINESS_OBJECT>
            <BUSINESS_OBJECT LOGID= "BO12"
TYPE= "MyBussinessObjSpace. BO12"
                    ASSEMBLY ="MyBussinessObj12. dll">
                    <INTERFACES>
                        <INTERFACE TYPE= "IBO12"/>
                    </INTERFACES>
                </BUSINESS_OBJECT>
            </BUSINESS_OBJECT>
            <BUSINESS OBJECT LOGID= "BO2" TYPE= "BO2NameSpace . BO2"
                ASSEMBLY= "BO2 .dll">
                <INTERFACES>
                    <INTERFACE TYPE= "IBO2"/>
                    <INTERFACE TYPE= "IBO3"/>
                </INTERFACES>
            <BUSINESS_OBJECT LOGID= "BO21" TYPE= "MeinBONeu"
                    ASSEMBLY ="MeinBO. dll">
                    <INTERFACES>
                        <INTERFACE TYPE= "IBO21"/>
                    </INTERFACES>
                </BUSINESS_OBJECT>
        </BUSINESS_OBJECT>
</APPLICATION>
```

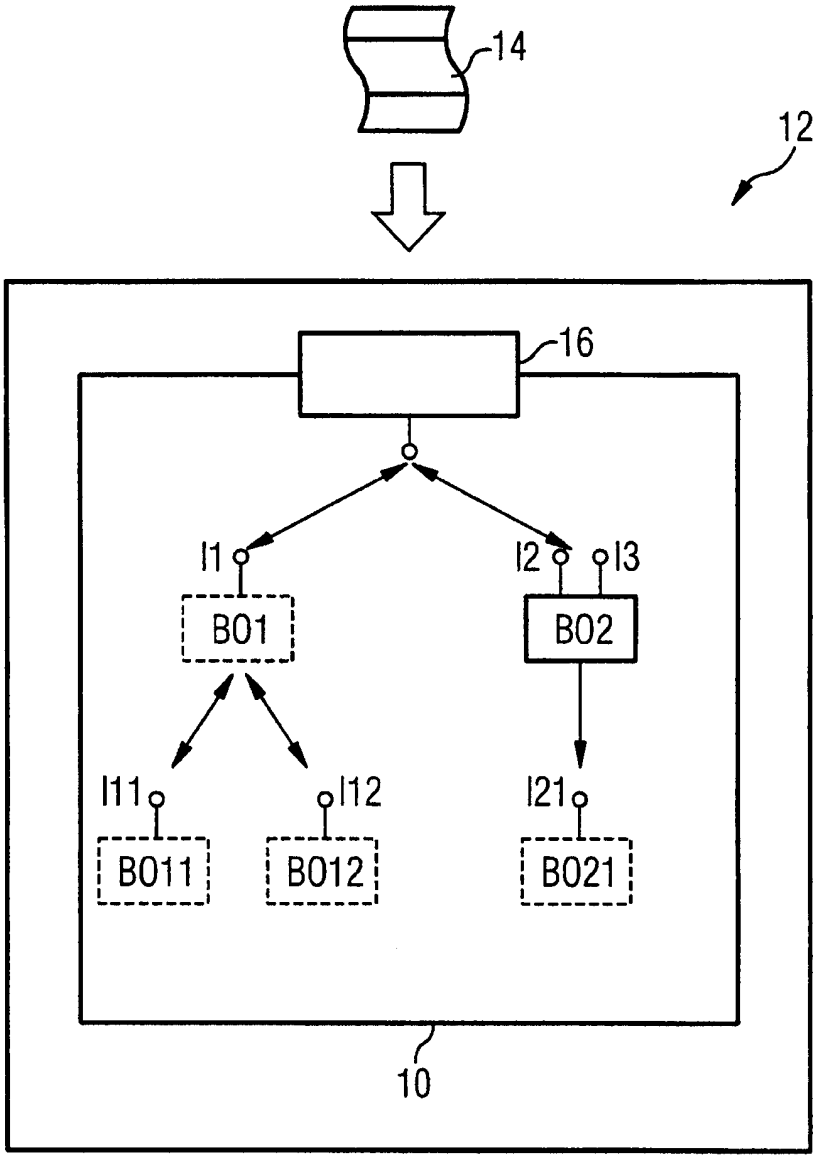


图 1

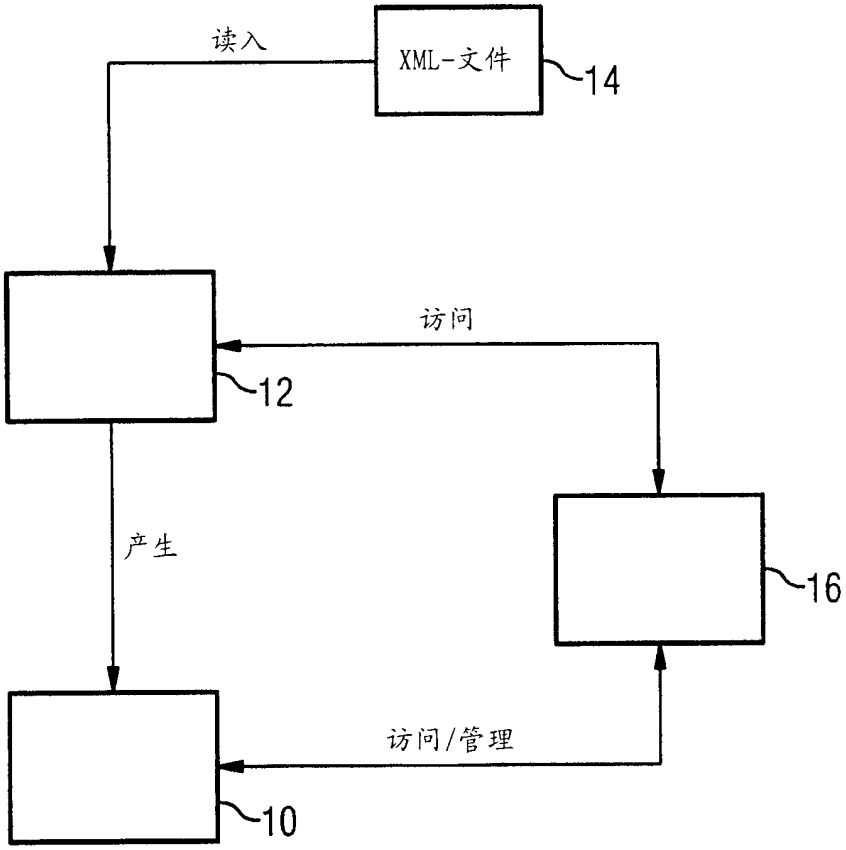


图 2