[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 说 明 书

[21] ZL 专利号 00810757.2

[45] 授权公告日 2004 年 5 月 5 日

[11] 授权公告号 CN 1148652C

[22] 申请日 2000.7.24 [21] 申请号 00810757.2

[30] 优先权

[32] 1999. 7.23 [33] US [31] 60/145,207

[86] 国际申请 PCT/CA2000/000841 2000.7.24

[87] 国际公布 WO01/008002 英 2001.2.1

[85] 进入国家阶段日期 2002.1.23

[71] 专利权人 加拿大柯达根技术公司

地址 加拿大魁北克省

[72] 发明人 米歇尔·布赖沙德

鲍利斯·申佳罗夫

审查员 刘宇儒

[74] 专利代理机构 中原信达知识产权代理有限责任公司

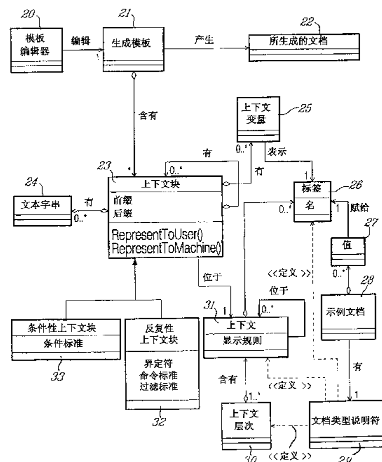
代理人 谷惠敏 袁炳泽

权利要求书 4 页 说明书 22 页 附图 16 页

[54] 发明名称 分层结构的控制信息编辑器

[57] 摘要

第一过程提供了对上下文敏感的编辑器，用于创建、修正和显示分层结构的控制信息。 该编辑器提供适合于当前上下文的动作选项，以此来简化用户操纵协议。 因为编辑器理解上下文之间的关系和多态性，所以其编辑器和显示器代表了上下文块的反复性和条件性，而毋需借助脚本语言。 编辑器根据关联上下文的显示规则显示出控制信息，该呈示清晰地区分出了大段文本、文本的结构和参数化要素。 编辑器把所谓的元控制信息显示为信息，从而高效地操纵元控制信息。 第二过程提供了一种方法，可把控制信息系统地变换成机器可懂的格式，比如代码生成器的脚本数据。



ISSN 1008-4274

知识产权出版社出版

1. 一种用来从分层结构数据中生成数据的系统，该系统包括：

分层信息定义器，用来指定关于所述分层结构数据参数要素的分层信息；

编辑器，用来为所述分层结构数据编辑模板并指定所述数据的非参数要素；

上下文定义器，用来指定上下文参数作为所述数据的参数要素；

过滤定义器，用来为至少一块所述分层结构数据指定过滤选项，该分层结构数据包括至少一个非参数要素和一个参数要素；和

数据生成器，用来至少使用所述分层信息来生成数据。

2. 权利要求 1 的系统，进一步包括：

上下文识别器，用来把显示特性与各所述过滤选项逐一关联起来；其中

所述数据生成器是显示数据生成器，用来使用各所述显示特性和所述分层信息来生成显示数据；其中，所述系统进一步包括：

显示器，用来使用所述显示数据把所识别的所述分层结构数据给显示出来；

藉以显示分层结构数据。

3. 权利要求 2 的系统，其中，所述显示特性是字体格式。

4. 权利要求 2 的系统，其中，所述显示特性是背景颜色。

5. 权利要求 2，3 和 4 之一的系统，其中，用于分层结构数据的所述模板包括基于组件的源代码。

6. 权利要求 2，3 和 4 之一的系统，其中，所述至少一块所述分层结构数据是常数型、条件型和反复型代码块中的一种。

7. 权利要求 2，3 和 4 之一的系统，其中，所述分层信息定义器使用分层信息，它来自统一建模语言的建模应用程序。

5 8. 权利要求 1 的系统，其中，

所述数据生成器是脚本生成器，用来使用所述分层信息、所述模板、所述过滤选项和想要的源代码语言的所述上下文参数来生成代码生成器的脚本数据；

藉以把分层结构数据变换成代码生成器的脚本数据。

10

9. 权利要求 8 的系统，进一步包括翻译数据定义器，用来指定翻译数据，其包括与所述要素相关的源代码表达式。

10. 权利要求 8 和 9 之一的系统，其中，所述代码生成器的脚本

15 数据被代码生成器用来生成源代码，其包括在所述想要的源代码语言中的反复、嵌套和近似反复的源代码中的至少一种。

11. 一种用来从分层结构数据中生成数据的方法，该方法包括：

指定关于所述分层结构数据参数要素的分层信息；

20 为所述分层结构数据编辑模板并指定所述数据的非参数要素；

指定上下文参数作为所述数据的参数要素；

为至少一块所述分层结构数据指定过滤选项，该分层结构数据包括至少一个非参数要素和一个参数要素；以及

至少使用分层信息来生成数据。

25

12. 权利要求 11 的方法，进一步包括：

把显示特性与各所述过滤选项逐一关联起来；其中

所述生成的数据是显示数据，且其生成包括使用各所述显示特性和所述分层信息来生成显示数据；而且，所述方法进一步包括

30 使用所述显示特性把所识别的所述分层结构数据给显示出来；

藉以显示分层结构数据。

13. 权利要求 12 的方法，其中，为所述分层结构数据编辑模板包括在文本窗内输入代码。

5

14. 权利要求 12 的方法，其中，指定上下文参数包括使用上下文参数定义符来定义所述参数要素的参数特性，并在所述数据中插入所述上下文参数的符号代表。

10 15. 权利要求 14 的方法，其中，使用上下文参数定义符包括从所述分层信息获得关于所述数据的对上下文敏感的信息，并提供所指定的可用上下文参数的列表。

16. 权利要求 11 到 15 之一的方法，其中，所述指定分层信息包

15 括使用统一建模语言的建模应用程序来指定所述分层信息。

17. 权利要求 11 到 15 之一的方法，其中，所述显示特性是字体格式。

20 18. 权利要求 11 的方法，其中

所述生成的数据是脚本数据，而且所述生成方法包括：使用所述分层信息、所述模板、所述过滤选项和想要的源代码语言的所述上下文参数来生成代码生成器的脚本数据；

藉以把分层结构数据变换成代码生成器的脚本数据。

25

19. 权利要求 18 的方法，进一步包括：指定翻译数据，其包括与所述要素相关的源代码表达式。

20. 权利要求 18 和 19 之一的方法，其中，所述代码生成器的脚

30 本数据被代码生成器用来生成源代码，其包括在所述想要的源代码语

言中的反复、嵌套和近似反复的源代码中的至少一种。

分层结构的控制信息编辑器

5 技术领域

本发明针对的是一种系统和方法，其在使用了图形用户界面的目标语言中，用来对源代码的模板进行编辑。说得再具体一点，本发明针对的是一种系统和方法，程序员可在其中借助应用域来对带有上下文元数据的目标代码做参数化。程序员确定代码的反复和嵌套结构，
10 该结构清晰直观，容易维护且完全一致。

背景技术

按通常的说法，面向对象的技术较传统技术更为贴近现实世界，且几乎皆靠建模工具来做成。在搭建一个复杂系统时，软件程序员和
15 客户之间容易建立起共同的标准，这有助于他们高效地提炼系统需求。面向对象的技术还有助于他们搭建一个容易扩展的系统，尤其是当使用框架的时候。

程序员和分析员们用建模工具来逻辑地描绘应用程序的商业用
20 途，这种逻辑描绘是对具体编程语言、应用程序藉以运行的操作系统和硬件的“具体实现”的抽像化。

面向对象的编程之所以愈发流行，原因之一是软件的反复使用变得愈发重要了。开发新系统的成本不匪，而维护他们的成本则更高。
25 新近，由全国标准局的 Wilma Osborne.所进行的一项调查显示，用于系统维护的开支占到了软件总成本的 60%到 85%（Ware Meyers.采访 Wilma Osborne.，IEEE 软件 5(3)：104-105，1988 年）。

传统上，程序员要靠“裁剪粘贴”来实现反复或近似反复的方法。
30 有种模板是一块目标源代码，在有改动的地方做了注解，保存在公共

位置并由程序员靠人工干预来反复使用。创建该目标源代码块的程序员或别的程序员把适当的代码行拷贝和粘贴到自己编写的代码中，并按特定用途做出适当的改动，从而反复使用这些代码行。这种方法难免要出错，还很费时，因为程序员不仅要（有时是反复地）拷贝所需的代码行，还得把所有适当的组件名替换成他们相应的等效值。这正是所谓：用带有注解的源代码模板来编程。

有些软件生成工具为程序员提供了文本生成模板，其包括参数化源代码行，用来新建运算或组件。这些文本生成模板不仅允许程序员使用被上下文变量所参数化的目标源代码来定义完整的运算法则，还允许程序员使用过滤变量，从而指定所选的他们藉以生成代码的组件。代码生成器把组件和属性过滤变量作为选择标准，用于过滤来自建模工具的信息，并移植于建模工具的动态生成文本中。上下文变量表示：被代码生成器替换成实际特征值的组件或属性特征名。这些实际特征值与当前组件或动态文本中的属性关联。

上述代码生成器被用来传递生成时刻的控制和定制，以此来整合框架和组件，这些组件是用如今的建模工具和编程语言来开发的。文本生成模板是以程序员的指令来创建的，程序员的指令使用刻板的语法而写成，并被源代码生成器用来替换所给的参数化源代码中的任何上下文变量。例如，某行代码可以是："Persistence Schema = new Persistence Schema("%001"); with(&class name)"，其中的"%001"是参数符，表示生成器要在此填入上下文变量的值，例如上下文变量"<class name>"的值。程序员只能使用这些参数符方可创建文本生成模板，程序员在命名组件时，为求保持一致，必须在脑海中建立一个参数符列表，尤其是当从%001 到%999 需要分别与上下文变量关联的参数符超过一个时。一个参数符可以不止一次地出现在参数化源代码中，因为它总与相同的上下文变量关联。这些参数符不像在非参数化形式中那样，他们不允许程序员键入代码，从这个意义上说，他们不是方便用户的。再有，必须在代码行内定义参数符，以确保生成

器能正确翻译。在上述例子中，只有加上“with(&class name)”指令方可命令生成器：参数符要被替换成上下文变量的当前值，该上下文变量与生成器的动态上下文中的当前类关联。而且，当代码的某部分被不止一次地替换成对过滤变量所提供的选择标准进行寻址的组件的属性时，只有包括类型指令“repeatForEach &attribute private instance using (&attribute name; &attribute attributeClassName;”方可确保近似反复的运算。在此情形中，代码生成器会把模板变换成源代码，这种变换使用了系统和过滤变量的模型说明，定义该过滤变量意在指定某一组件须用何模板。

程序员也可不用这种有赖关键字（例如，%001）和关键字定义（例如，repeatForEach ... using ... or with ...）的语法，而是考虑把所给编程语言的算法所执行的指令与参数化代码链接起来，像在授予 Lindsey 的美国专利号 No. 5,675,801 里所教导的那样，该软件会用别的运算结果来求解参数形式的表达式。说得再具体一点，例如，以 C 语言所写的说明源代码模板，用于面向数据的对象，也即 NumericData-ItemObject，还用来表示名为“IDNumber”的 C 变量，在该例中使用了类型指令“int<self VarName>;”，其中的“int”和“;”是该源代码模板的一段源代码，而生成器的指示则是“<self VarName>”。该例中的“<”和“>”是任意的界定符，标明生成器的起始。在此情形中，对象识别符是记号“self”，表示它是被请求自我生成的对象，即其变量名“IDNumber”；消息识别符是记号“VarName”，即被发送到对象“self”的消息。该消息命令“self”对象：当被用于变量说明时，返回代表它自己的字串。在 Lindsey 的这一示范中，名为“IDNumber”的 NumericDataItemObject 会返回字串“IDNumber”。由此得出的目标语言源代码段是“int IDNumber”，它是 C 语言的源代码段。

颜色和画面已被用于源代码编辑器中，以增进对算法流程的了解，但却从未用来说明所提供的信息的结构，比如反复和/或嵌套的上

下文块。例如，在 if-then-else 语句中使用不同颜色表示其条件表达式、基于条件表达式的结果而执行的真表达式和假表达式。画面从未被用来阐释这一事实：即，同一 if-then-else 表达式对于各符合选择标准集的属性在所给的组件算法内反复使用。

5

美国专利 5,603,018 教导了一种用于程序开发的系统，它带有能分层地代表指定对象的图形编辑器，并能够在不同图形编辑器的代表格式之间转换。

10 发明内容

于是，本发明旨在提供一种编辑器，使程序员能创建清晰直观而高效的模板。

本发明还旨在提供一种编辑器，其中反复和嵌套的代码清晰可

15 辨。

本发明还旨在提供一种用于创建源代码模板的编辑器，它以最少的程序员人工干预来获得想要的模板。本发明又旨在提供一种模板编辑器，以满足编辑模板源代码之需。该模板编辑器适用于各种环境，

20 比如基于万维网的服务器，框架环境，代码生成及其它。

本发明亦旨在使用编辑器来生成脚本模板。

本发明亦旨在提供一种用于显示分层结构数据的系统和方法，从

25 而帮助程序员以图形用户界面来编辑源代码。

为在计算环境中达到上述意图，并根据在此处所强调的发明目的，提供了一种系统，用来从分层结构数据中生成数据，包括：分层信息定义符，用来指定关于分层结构数据参数要素的分层信息；编辑

30 器，用来为分层结构数据编辑模板并指定所述数据的非参数要素；上

下文定义符，用来指定上下文参数作为数据的参数要素；过滤定义符，用来为至少一块分层结构数据指定过滤选项，该分层结构数据包括至少一个非参数要素和一个参数要素；和数据生成器，用来至少使用分层信息来生成数据。

5

在一种实施例中，优选地，本系统进一步包括上下文识别符，用来把显示特性与各过滤选项逐一关联起来；而其中的数据生成器是显示数据生成器，用来使用各显示特性和分层信息来生成显示数据；而且，本系统进一步包括显示器，用来使用显示数据把所识别的分层结构数据给显示出来；藉以显示分层结构数据。

10

在另一种实施例中，优选地，数据生成器是脚本生成器，用来使用分层信息、模板、过滤选项和想要的源代码语言的上下文参数来生成代码生成器的脚本数据；藉以把分层结构数据变换成代码生成器的脚本数据。

15

根据本发明的另一方面，提供了一种方法，用来从分层结构数据中生成数据，包括的步骤有：指定关于分层结构数据参数要素的分层信息；为分层结构数据编辑模板并指定所述数据的非参数要素；指定上下文参数作为数据的参数要素；为至少一块分层结构数据指定过滤选项，该分层结构数据包括至少一个非参数要素和一个参数要素；以及至少使用分层信息来生成数据。

20

在一种实施例中，优选地，本方法进一步包括：把显示特性与各过滤选项逐一关联起来；而其中所生成的数据是显示数据，且其生成包括使用各显示特性和分层信息来生成显示数据；而且，本方法进一步包括使用显示数据把所识别的分层结构数据给显示出来；藉以显示分层结构数据。

25

在另一种实施例中，所生成的数据是脚本数据，而且优选地，本

30

方法进一步包括使用分层信息、模板、过滤选项和想要的源代码语言的上下文参数来生成代码生成器的脚本数据；藉以把分层结构数据变换成代码生成器的脚本数据。

5 提供了一种系统，用于编辑基于组件的源代码模板。该系统包括上下文编辑器，用来键入源代码和参数化时的上下文变量。上下文变量代表了组件和属性特征名，属性特征名在生成时被生成器替换成其实际特征值。源代码其实是被捕入能相互嵌套的上下文块里的，他们代表了常数型、条件型或反复型上下文块，在反复型上下文块的情形

10 中条件表达式用于确定是否生成了上下文块和上下文块出现的次数。

还提供了另一方法，用于创建源代码模板、上下文捕获控制信息，并以视觉效果来代表上下文变量和上下文块。这当中，视觉效果包括颜色、特殊字体、修正的光标形状或乃至音效。

15

为综观本发明而提供了两个独特的实施例，它们皆涉及对分层结构的控制信息的操纵。第一过程使人能高效创建、修正控制信息并使之可视化；第二过程可使该控制信息经确定性变换而成为机器可懂的格式。

20

第一过程提供了对上下文敏感的编辑器，用于创建、修正和显示分层结构的控制信息。该编辑器呈示出适合于当前上下文的动作选项，以此来简化用户操纵协议。因为编辑器理解上下文之间的关系和多态性，所以其图形用户界面代表了上下文块的反复性和条件性，而

25 毋需借助脚本语言。编辑器根据关联上下文的呈示规则显示出控制信息，该呈示清晰地区分出了大段文本、文本的结构和参数化要素。编辑器把所谓的元控制信息显示为信息，从而高效地操纵元控制信息。

根据本发明的优选实施例，提供了一种方法和系统，用于显示分

30 层结构数据。

5 该方法和系统包括：指定关于分层结构数据参数要素的控制信息；为分层结构数据编辑模板并指定数据的非参数要素；指定上下文参数作为数据的参数要素；为至少一块分层结构数据指定过滤选项，该分层结构数据包括至少一个非参数要素和一个参数要素；把显示特性与各过滤选项逐一关联起来；使用显示特性把所识别的分层结构数据给显示出来。

10 第二过程提供了一种方法，可把控制信息系统地变换成机器可懂的格式。

15 根据本发明的另一优选实施例，提供了一种方法，用于把分层结构数据变换成代码生成器的脚本数据。该方法包括：为分层结构数据的有关参数要素指定控制信息；为分层结构数据编辑模板和指定数据的非参数要素；指定上下文参数作为数据的参数要素；为至少一块分层结构数据指定过滤选项，该分层结构数据包括至少一个非参数要素和一个参数要素；使用控制信息、模板、过滤选项和想要的源代码语言的上下文参数来生成代码生成器的脚本数据。

20 根据本发明的第三种优选实施例，提供了一种系统和方法，用来显示分层结构数据，这就需要指定参数、编辑模板、指定上下文、与显示特性关联、指定过滤选项。

25 实际上，程序员以应用域的上下文元数据来参数化源代码，并确定代码的反复和嵌套结构，以实现清晰直观和容易维护。

附图说明

至此已概述了发明的内容；下面将参考附图来说明优选实施例。在图中：

30 图1是分层结构数据编辑器的类原理图；

图 2 是新模板的示范；
图 3 是把模板译成脚本语言的示范；
图 4 是由脚本语言生成源代码的示范；
图 5 是显示和翻译系统的框图；
5 图 6 是显示分层结构数据的方法的步骤流程图；
图 7 是把分层结构数据翻译成脚本语言的方法的步骤流程图；
图 8 示出了新运算署名工具；
图 9 示出了在模板编辑器内所创建的署名；
图 10 示出了在注释中插入的上下文变量；
10 图 11 示出了算法的创建；
图 12 示出了在算法中插入的上下文变量；
图 13 示出了如何存取属性级别；
图 14 示出了如何为属性编写递归运算的代码；
图 15 示出了如何在该属性级别中插入上下文变量；和
15 图 16 示出了完整的模板。

具体实施方式

为促进模板程序员编写代码的过程，创建了一种新的分层结构的控制信息编辑器（下文谓之“模板编辑器”），该模板编辑器使用编
20 辑器来显示分层结构和参数化代码。

分层结构的控制信息编辑器的主要目的是要使程序员以一种酷似生成输出的直观形式来创建和编辑生成模板，这样，对于生成器的任何所给输入，一眼就能看出会生成何种输出。为使之成为可能，模板
25 编辑器提供了图形编辑器（参看图 2），程序员可在其中键入文本字串 24，插入上下文变量 25 或插入上下文块 23。用各种视觉效果，比如文本/背景颜色，光标/文本形状等等（看图 1 的上下文显示规则 31），来显示用户所输入的文本与各种上下文变量 25 或上下文块 23 之间的区别。上下文变量 25 无法被修正，它们是只读字串。对上下
30 文变量的可能操作只有替换或删除。

图 1 示出了编辑器实现的统一建模语言(UML)模型。主要的类是上下文块 23、上下文变量 25 和文本字串 24，随后的文本将说明他们在图形用户界面(GUI)编辑和脚本语言翻译中的作用。

5

文本字串 24 代表上下文块 23 里的静态数据（即字符集），文本的外观会依上下文块而变化。再有，各字符皆有自己的一套编辑手续，当程序员企图改动该字符时就要履行这些手续。文本字串 24 的各字符也有自己的一套行为特征。由于字符的行为特征往往是不自觉地反复变成许多连续的字符，故使用稀疏的代表以节省电脑内存。

10

上下文变量 25 被其上下文块 23 链接至模型对象，因而能代表该对象的各种特性。在模板编辑器 20 中，上下文变量 25 的各字符皆与文本字串 24 的字符结构相同，模板编辑器 20 不允许编辑代表，以此来保持上下文变量 25 的完整性。

15

上下文块 23 含有文本字串 24 和/或上下文变量 25 和/或别的嵌套上下文块 23，上下文块 23 的作用根据附加是根据附加条件反复地或有条件地产生其内容。对于需要反复的上下文块 23 来说，程序员必须提供选择标准（即，条件和过滤变量），这是确定反复次数所不可或缺的。程序员通过使用标准而控制每次反复的上下文块排序。

20

当提供了上下文块 23 的过滤器或条件标准后，除非满足了标准，否则不生成块。对于各符合选择标准的对象，上下文块 23 将被实例化。

25

上下文块 23 有三种格式：条件型 33，常数型和反复型 32。对于每个被其过滤器（即标准）所捕获的对象，皆会生成一个独一无二的条件型上下文块 33；常数型上下文块 像是条件永远为真的条件型上下文块 33；反复型上下文块 32 则对于每个被上下文块条件（例如，

30

类中各目标的作用）所捕获的项目在对象中反复生成。反复型上下文块 32 还能在块的每次反复之间插入用户定义的界定文本，比如后面跟着一个空格的逗号。

5 在以面向对象的语言所开发的实际应用中，文本字串 24 的组成为：一个字串对象和一个与字串大小相等的阵列对象。文本的阵列对象代表文本的行为特征。对于字串对象中的各字符，皆有一个对象在相应的阵列位置上，该对象含有字符的图形代表的信息，比如颜色、字号、加粗或斜体特征。该行为特征还存储了字符的类型，无论是上10 下文变量 25 还是文本字串 24。这些特性包括行为特征，并把荧幕上的文本与模板编辑类原理图中的对象链接起来。

下面解释如何把键入到模板编辑器中去的生成模板译成脚本语言的格式。如类原理图（图 1）所示，生成模板是上下文块 23 的一个序15 列。上下文块含有文本字串 24 的一个序列、上下文变量 25 和上下文块 23。

文本字串 24 代表静态字符，像在脚本语言格式中所代表的那样。

20 上下文变量被置于脚本语言格式中，在该处形象地显示出该变量的值。在脚本语言格式中，这是通过一个前面带‘%’的数字（如：%001）来显示的。上下文变量的值取决于使用它的上下文。

当译成脚本语言格式时，依一个标准集把上下文块写入脚本语言25 格式中去。在反复型上下文块的情形中，写入脚本语言格式以代表上下文块的反复。

示范

下面通过一个示范来帮助说明本发明的优选实施例。该示范将示30 出如何定义上下文层次（参看图 1）和各层次的某些呈示规则（参看

表 1）、如何使用表 2 定义的控制信息数据、如何定义代表 Java 方法（图 2）的模板、如何生成模板并获得脚本、如何使用已公布的题为“与组件研发关联的源代码模板生成器”（PCT 专利申请号 WO 00/22517）的系统来执行该脚本，以获得 Java 源代码（图 4）。

5

下面的上下文层次说明了表 2、3 和图 2、3、4 中所使用的可能的上下文。示范中所使用的所有对象皆属于层次的某个级别，并获得与该级别关联的上下文特征。

10　下表定义了各上下文。对于各上下文，还定义了可能的上下文变量、子上下文和对象可用的特征。此定义使模板编辑器 20 可以显示不同上下文块 23 的子菜单、字体和颜色。

级别	实体类型	显示设定	特征
1.1	Class	白底黑字	Name: String Attributes: {attribute}
1.1.1	Attributes	橙底黑字	Name: String Type: class

表 1. 层次的呈示规则

15

表 2 示出了（来自 UML 模型的）用于驱动模板实例化的信息。

级别	名称	实体类型	特征	值
1.1	Cityinfo	Class	class name	Cityinfo
			Attributes	{name country population}
1.1.1	Name	Attribute	Attribute name	name
			Attribute Name	Name
			Attribute type	String
1.1.1	Country	Attribute	Attribute name	country
			Attribute Name	Country
			Attribute type	String

1.1.1	Population	Attribute	Attribute name	population
			Attribute Name	Population
			Attribute type	Integer

表 2. 从 UML 模型获得的信息

图 2 示出了在分层结构的控制信息编辑器中的模板，该示范是基于各类而实例化的。

5

示出了类原理图和示范之间的关系。生成模板的示例对应于该模板，该示例包括上下文块，它表示类上下文；该上下文块包括：

文本字串	值：	/*** Take Properties result set received from database, and populate
上下文变量	值：	<class name/>(Class name)
文本字串	值：	's Data fields. *@param resultSet The data used in populating the
上下文变量	值：	<class name/>(Class name)
文本字串	值：	's Data fields. */ public void setPropObjectData (java.util.Properties resultSet){
上下文块	表示属性上下文并包括：	If(resultSet.get("<Attribute Name/>")!=null && !(resultSet.get(etc...
文本字串	值：	}

表 3. 上下文块的内容

10

使用下面的注释会更好地理解图 2：

34. 这些块是基于各类而实例化的，各类皆要对所有控制信息（文本段、上下文变量或从属块）实例化一次。

15

35. 这些是上下文变量，在未被实例化时，他们形如被“<”和“>”括起来的特征名。

36. 该块是基于各属性而实例化的，由于包含它的块是基于各类而实例化的，故该块可以在各类的实例化中任意次地反复。优选地，该反复次数可以是任何大于或等于零的整数（若类中有反复次数为零的属性，则跳过该块）。

37. 各类的各属性皆要对这些上下文变量实例化一次。

通过把表 2 中所说明的控制数据传递给图 2 所示的模板，则生成了图 3 所示的脚本代码。

参考图 2 和图 3，创建模板的过程如下所述：

生成模板的示例标出了已知信息，该已知信息合起来即构成模板的序言，序言提供了模板名、过滤器（在此情形中，过滤器是“所有类”）和模板特性（在此情形中，模板创建了一个 Java 运算）。

```
templateName^!setPropObjectData! definedAs^(&class)
    generate^(
        operation^public^instance^(!setPropObjectData!)
        definedAs^(
```

接着，由模板的类上下文块来负责脚本。在脚本上下文中，按下面的格式嵌入上下文块：

```
(!<block contents>!)
```

于是，该块为模板序言写入了如下脚本：

```
(!
```

这之后，由它的各个下级来继续负责脚本。第一个下级是文本字串，文本字串按它在脚本中所出现的那样生成，于是，第一文本字串被写成：

```
/**
*Take Properties result set received from database, and populate
*
```

下一要素是上下文变量。在脚本语言中，如前所述，上下文变量必须写入对于对象的引用。在脚本中，按上下文变量出现的次序来引用参数。该次序由上下文块来确定，上下文块赋予各变量一个序号，从而注册了自己的上下文变量列表。

在此情形中，对于一个类名有上下文变量。上下文变量向其上下文块查询自己的出现次序。

由于该上下文变量是第一个变量，它获得了自己的次序并写入：

```
%001
```

对于所有的块要素，皆反复执行相同的过程，即得出了下面的脚本文本：

```
's Data fields.
*@param resultSet The data used in populating the %001's Data fields
*/
public void setPropObjectData(java.util.Properties resultSet){
```

然后由上下文块来负责脚本，上下文块知道只有用惊叹号(!)方可

结束自己。若上下文块不是反复性的，且至少包括一个上下文变量，则会写入：

! with^

5

要是上下文块是反复性的（像对于属性的情形那样），则本该写入：

repeatForEach^attributeOrTargetRoleFilterExpression^&attribute

10

上述情形中没有过滤器，但要是属性为 private，我们本该在 &attribute 后面写入 private 的。

下一步，上下文块请求各上下文变量写入自己的初始行为特征。

15　于是上下文块写入：

(

各上下文变量再依次写入自己。类名上下文变量写入了：

20

&class name^

由于不再有别的上下文变量了，故上下文块为参数加上反括号，并接着为自己也加上反括号：

25

))

对于当前生成模板所属的附加块，依此类推地继续生成脚本，直到以脚本语言完整地生成了模板。

30

图 3 中的粗体文本直接源自图 2 模板中的文本，下划线文本涉及上下文变量，而其它文本则涉及脚本语言的结构代码。

需要注释下面几点：

5

75. 主块是基于各类而实例化的；各类皆要对所有控制信息——文本段、上下文变量或从属块实例化一次。

各类皆要对上下文变量 76 和 77 替换一次。

10

76. 于此处接收被替换的上下文变量。

77. 这是驱动上下文变量替换的指令。

15

78. 该块是基于各属性而实例化的，由于包含它的块是基于各类而实例化的，故该块可以在各类的实例化中任意次地反复。优选地，该反复次数可以是任意大于或等于零的整数（若类中有反复次数为零的属性，则跳过该块）。

20

各类的各属性皆要对上下文变量 79 和 80 实例化一次。

79. 于此处接收被替换的上下文变量。

80. 这是驱动上下文变量替换的指令。

25

图 4 包括所生成的代码，其以目标语言（在此情形中是 Java 语言）写成。该代码只不过是用图 2 所示的模板生成代码的一个示例罢了。图 2 所示的用户界面不言而喻地适用于所有可能的示例。

30

需要注释下面几点：

81. 选择基于各类而生成的代码。

82. 上下文变量<Class Name>被替换成“cityifno”。

83. 以来自示例的数据来替换上下文变量；参考表 2 和图 2 以弄清是如何进行替换的。

在各属性的区域中：

84. 由 Country 属性所生成的上下文块。

85. 由 Population 属性所生成的上下文块。

86. 由 Name 属性所生成的上下文块。

下面将使用框图 5 和流程图 6、7 来详细说明本发明的优选实施例。

图 5 是系统框图，该系统是用于分层结构数据的，并用于生成代码生成器的脚本数据。编辑器 120 用来键入并修正模板。编辑器使用上下文定义符 121 来引入上下文变量和参数要素。接着，上下文定义符 121 使用分层信息定义符 122 来获得有关模板上下文的信息，该上下文可从（例如）UML 建模工具获得。编辑器 120 还使用过滤定义符 123 来为生成源代码而定义过滤选项。上下文定义符 121 还与上下文识别符 125 通讯，从而为各上下文变量分配显示特性。最后，编辑器 120 把模板发送到显示器 126 上显示出来。显示器 126 使用显示特性 124 来显示模板的各参数和非参数要素。作为选择，系统还有代码生成器的脚本生成器 127，它使用编辑器 120 的模板、过滤定义选项 123 和上下文定义符 121 的上下文参数来生成代码生成器的脚本数据。

图 6 是根据本发明优选实施例的方法的步骤流程图。步骤 130 指定分层信息；步骤 131 编辑模板；步骤 132 指定上下文参数；步骤 133 还要指定过滤选项；步骤 134 把显示特性与上下文参数关联起来；最后，步骤 135 把数据显示给用户。

图 7 是根据本发明优选实施例的另一方法的流程图。步骤 140 指定分层信息；步骤 141 编辑模板；步骤 142 指定上下文参数；步骤 143 还要指定过滤选项；接着步骤 使用模板、上下文参数和过滤选项生成代码生成器的脚本数据。

固然，从本发明的优选实施例中得知，代码生成器的脚本数据是一个单独的文件，其包括代码生成器的所有相关信息并使允许代码生成器以目标语言来产生源代码；但是也要明白，代码生成器的脚本数据可能不止包括一个被代码生成器用来以目标语言产生源代码的数据文件。例如，代码生成器的脚本数据可以包括第一文件，它独立于上下文，并含有要生成的方法和类；还包括第二文件，它含有取决于上下文的信息，如模型的细节。代码生成器的脚本数据也可只包括独立于上下文的数据，而代码生成器则另觅别的能提供取决于上下文的信息的来源。

下面给出了一个具体的示范，其中从荧幕上抓拍了用户界面，它是根据本发明的优选实施例而创建的。

图 8 示出了新运算署名工具 38。模板名 40 须与运算的可见性 41、承袭性 43 和运算是否为静态 一起给出。返回类型须由其类名 44 和其分组 45 来定义。运算名于 46 处键入。完成上述信息后，按 OK 按钮 39。

图 9 示出了带有方才所创建的运算的署名 52 的模板编辑器 50。

该文本无法直接修正，因为它是由组织器来管理的。为进行修正，可以双击署名字串并重新弹出运算署名工具。为表明该文本是由运算署名工具所创建的，它以不同的视觉效果而显示。在附图中，由于附图是黑白的，故字体和背景的颜色皆是灰色阴影；不过，可以使用任何视觉特性以突出模板中不同性质的文本的区别。例如，署名可以用蓝色的背景来显示，表示程序员只有通过双击方可改动该字串里包括的内容。空间 53 是留给为运算编写代码用的，用不同于署名的背景颜色来显示它，表示程序员可以向该空间填写内容。若（假如）程序员有视力障碍，则荧幕上的视觉效果可替换成音效。事实上可以使用任何手段，只要能标明代码部分的不同特性即可。指定处理类过滤器 51 以确定哪个/些组件是要由该图形生成模板来生成的（附图中未示出这些步骤）。

图 10 示出了如何在运算中创建注释。可见，在注释中可以加入上下文变量 59，比如属性名 60。图 11 示出了完成后的注释 65 和一些由程序员所键入的目标源代码。如图 12 所示，使用如图 10 所说明的过程，在指令的代码 71 中加入了上下文变量“attribute name”。

图 13 示出了如何创建单名运算。同样地，用不同的背景颜色来代表署名 95。其中有注释 96，而运算的代码在 97 处。若需要为组件的一些或所有属性而反复执行该运算的一部分代码，则可能键入属性级别 98 并创建一个循环。图 14 示出了为某些属性而反复的代码行 104，图中，为代码的循环部分使用了不同的背景颜色 103。同样地，可能在属性级别处插入上下文变量 109，例如属性名 110（图 15）。最后，在图 16 中完整地创建了模板。若保存了该模板，则代码生成器（比如上文提到的 PCT 公开号 WO 00/22517）即可使用该模板和模型说明来创建源代码。

本发明可用于不同环境中，下面将对某些使用环境加以说明。

本发明的用途之一是通过创建模板的代码行而隐藏来自开发者的脚本语言，从而用生成工具来生成代码，这样的生成工具例如已转让的与本申请一起待审的题为“与组件研发关联的源代码模板生成器”（PCT 专利申请号 WO 00/22517）所述的生成工具。在此情形中，使用模板来为组件生成近似反复或反复的源代码，这些组件符合生成模板的选择标准，而且是在建模工具或集成开发环境中的。

当程序员键入将被用于系统所有类中的上下文块时，只消键入生成代码时他要用到的方法即可。他可以在欲写入类名的地方，用滑鼠右击上下文编辑器，从所有可能的上下文变量列表中选择他想要的上下文变量。在此情形中，他会选择“<class name>”。参考现有技术一节所述的示范，新一行代码即成为：

```
PersistenceSchema = new PersistenceSchema("<class name>");
```

一旦在滑鼠右击弹出的菜单中选择了上下文变量“<class name>”以后，编辑器就自动把它加入到代码中去了。

当某些上下文块需要在同一算法中反复一次以上时，必须使用过滤变量来创建循环。过滤变量确定参与循环的项目的选择标准，这些项目例如：组件、组件子类、超类、或属性特征。在上述示范中，程序员只有为循环定义指令方可覆盖所有属性。在新的模板编辑器中，用视觉特性来识别对于所有属性皆紧密相关的块上下文。

例如，可以用不同的背景颜色来识别上下文块，接着便自然要键入代码行了，并通过滑鼠右击来引入上下文变量。既然编辑器很明白：对于所有属性皆要复制这些代码行，故在滑鼠右击时弹出的上下文菜单中的上下文变量的选择势必不同于前面的菜单。这些上下文变量不尽属于同一种类的上下文变量，例如，上下文变量“<class name>”可位于反复性上下文块中，该上下文块在当前组件的属性级别上选

代；或可位于内部上下文块中，该上下文块在所给的外部特征名上迭代，该外部特征名涉及生成器动态上下文中当前类的当前属性。这是因为，生成器要从外层级别到合适的级别递归地查询，以获取特征值。总是用视觉特性来识别循环，这样就突出了编程的外观。务必注意到：甚至是在属性级别上编写代码时也可能插入（例如）表示类的上下文变量。这就带来了极大的灵活性，因为程序员可以使用所有上下文变量来为在属性级别所需的运算而编写代码。

在另一环境中，编辑器被用于可反复使用的代码行库。例如，可以用模板来创建个人或联网的可反复使用的代码。在此情形中，由于模板已被参数化了，故程序员毋需对代码行做任何修正即可用于自己的工程了。该方法替程序员省却了大量的工作。

使用这些模板的又一示范是面向万维网的。迄今为止，超文本标记语言(HTML)的开发者们只能使用正确信息的指针或记号来编写网页代码。他们想引入的 HTML 代码可以处理页面的呈示，并包括用于存取服务器数据库中的合适信息的代码。同样地，插入他的网页编辑工具中的模板编辑器使程序员能创建网页的视觉外观，而毋需费神考虑藉以存取数据库数据的代码了。程序员可以用传统的 HTML 语言来编写网页代码并加入上下文变量，该上下文变量反映出当需要为特定页面生成特定代码时，对来自数据库的所需信息的说明。例如，当程序员创建页面以使顾客能存取他们的帐户结余时，可以使用模板编辑器创建一个 HTML 模板，并为实际结余使用参数化组件，比如<account balance>。当生成该代码时，生成器会为所有顾客单独地把<account balance>替换成正确的量。于是该网页在观察者看来就是完全相同的了，除了几乎毋需程序员一方编程的帐户结余以外。本发明可提供用于该环境中的图形用户界面，而不改变本发明的本质。

按相同方式，模板编辑器的改进版允许程序员将源代码衍变为模板。程序员毋需以参数化的语法来书写代码了，他可以把他的源代码

保存为一个文件。当他想起可以反复利用该代码时，他可以指定以何组件、何框架在使用该代码的场合中生成。生成器由于清楚被建模系统的所有组件和所有要素之间的关系，故可以提取关于如何开发代码的信息。弄清了模型的类名和属性名并辨认出了代码的近似反复块，编辑器方可把原始文件变换成模板的参数化源代码，它既可以被显示出来以征求程序员的同意，也可以自动生成并存盘。然后，生成器即可使用该代码并为新应用而产生代码。程序员毋需书写实际的参数化代码了。

而且，还可以使用嵌入在字处理应用中的模板编辑器来产生模板，从而生成文本文件，比如参数化的邮寄列表，其带有姓名、地址的记号和客户的账户代表。

事实上，几乎可以使用模板编辑器来生成任何东西，只要有了想要生成的东西的信息即可。可以使用扩展标记语言(XML)的模板编辑器。模板编辑器可以使用 XML 协议来提取想要生成的东西的信息。可以使用 XML 文档类型定义(dtd)来把模板编辑器所使用的元数据定义为上下文变量，并可以使用 XML 的 dtd 树结构定义来了解参数化源文本的不同级别（也即，用来在兼容 XML 文档的各嵌套级别上定义合适的过滤变量）。可以使用被写入 XML 的数据时间定义(dtd)文件，而非以建模工具而搭建的模型说明。在此情形中，XML 的 dtd 可以取代对于代码生成所不可或缺的模型说明，因为他说明了被处理的数据。若两个文件皆使用同一 dtd 文件，则可互换他们的数据文件，因为数据具有相同的组织，并导致模板也具有相同的行为。

尽管具体地参考所述的实施例而说明了本发明，但是应该理解：对本领域的技术人员还会出现许多修正。相应地，以上说明书和附图应作为对本发明的说明，而无限制之意。

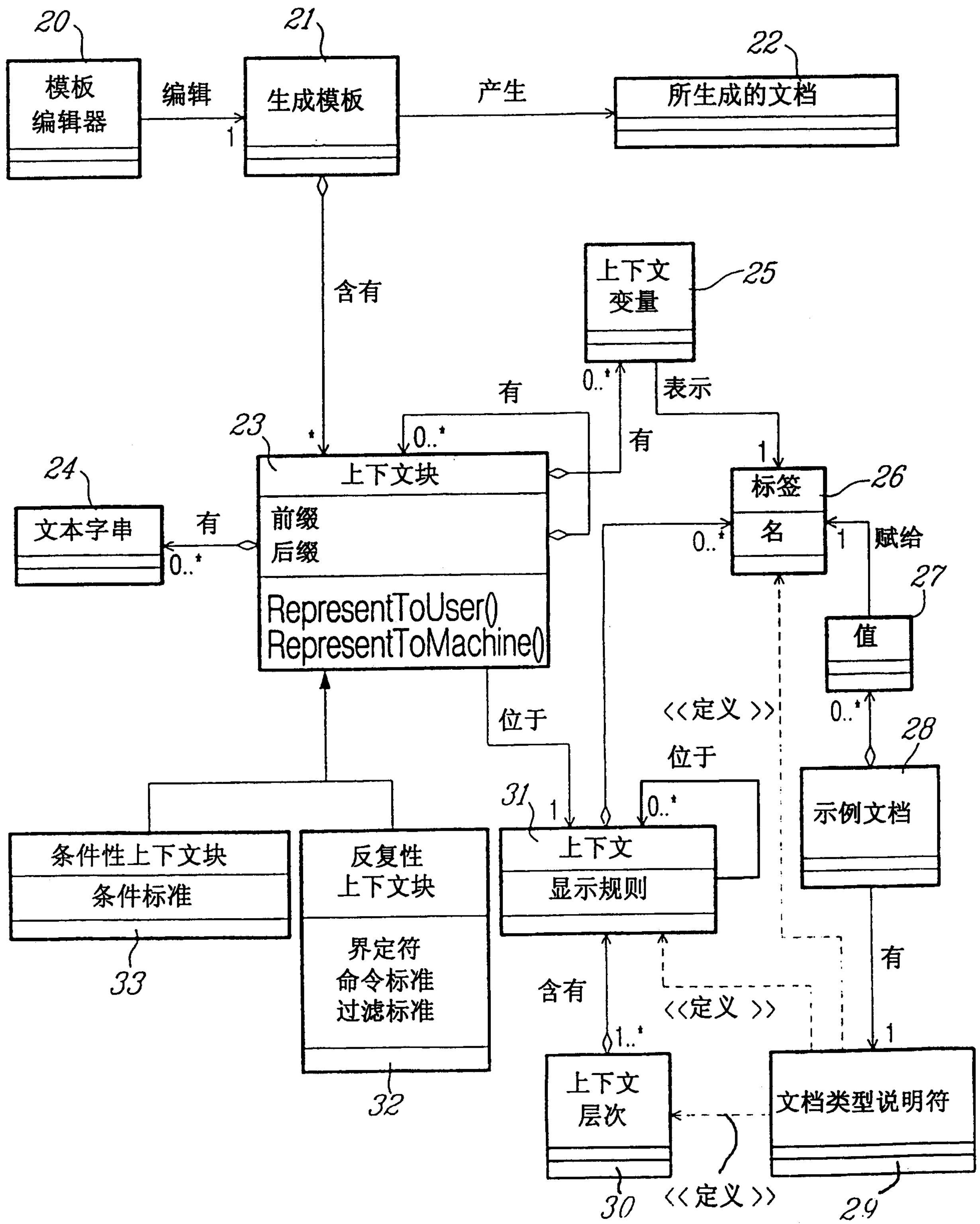


图1

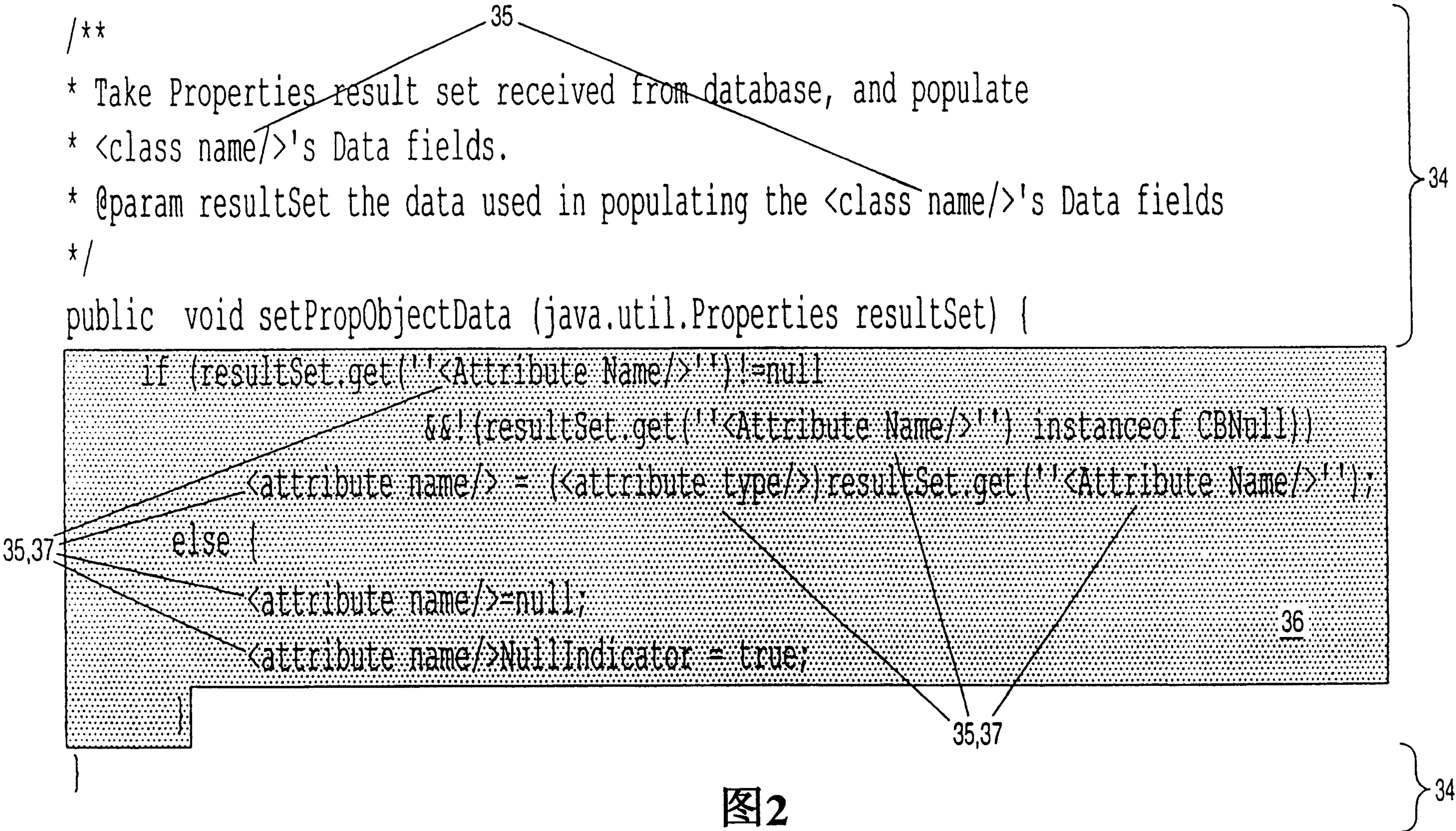


图2

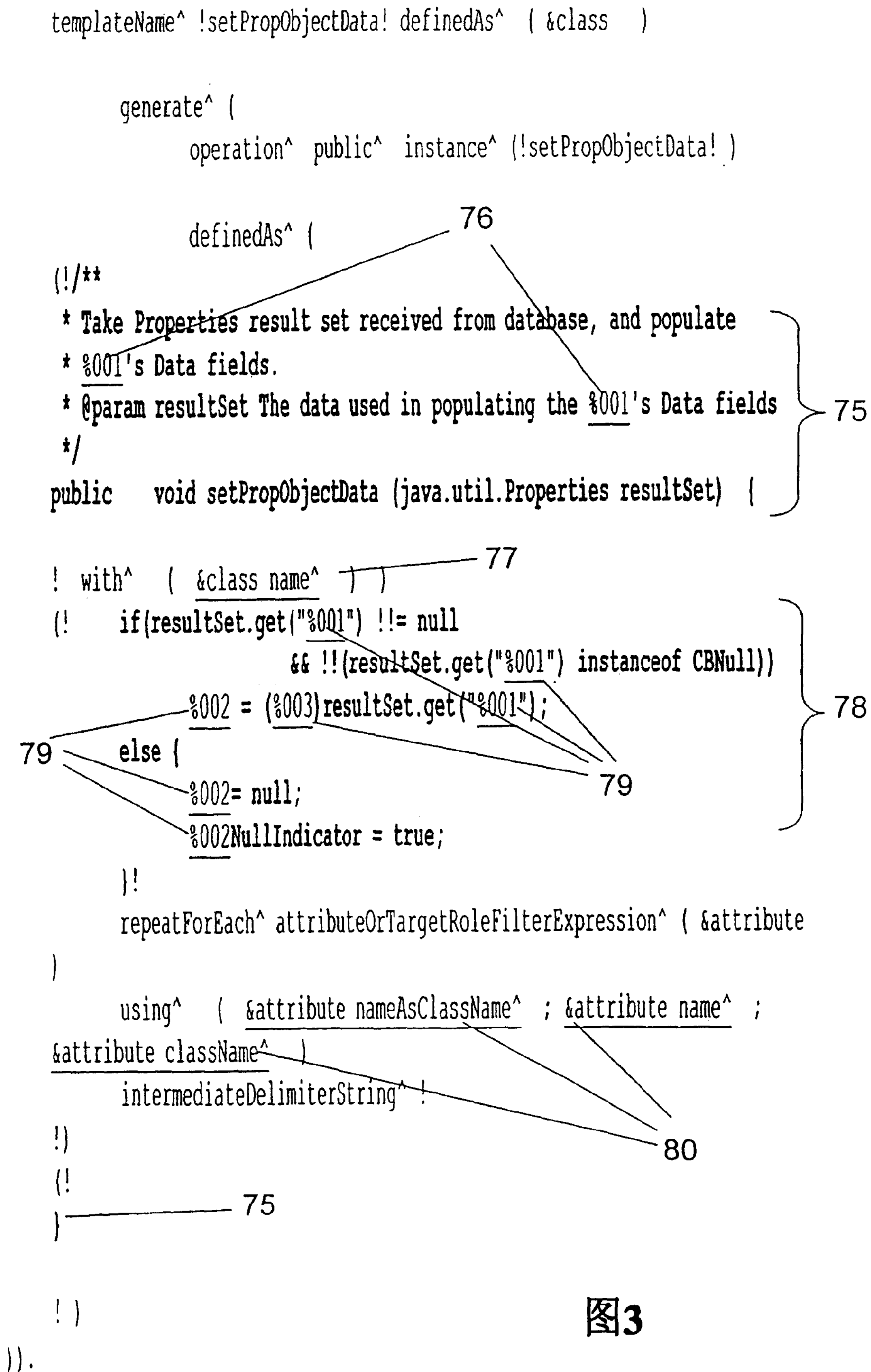


图3

```
/**
 * Take Properties result set received from database, and populate
 * cityinfo's Data fields.
 * @param resultSet The data used in populating the cityinfo's Data fields
 */
public    void setPropObjectData (java.util.Properties resultSet)  {
    if(resultSet.get("Country") != null
                && !(resultSet.get("Country") instanceof CBNull))
        country = (String)resultSet.get("Country");
    else {
        country= null;
        countryNullIndicator = true;
    }
    if(resultSet.get("Population") != null
                && !(resultSet.get("Population") instanceof CBNull))
        population = (Integer)resultSet.get("Population");
    else {
        population= null;
        populationNullIndicator = true;
    }
    if(resultSet.get("Name") != null
                && !(resultSet.get("Name") instanceof CBNull))
        name = (String)resultSet.get("Name");
    else {
        name= null;
        nameNullIndicator = true;
    }
}
```


图4

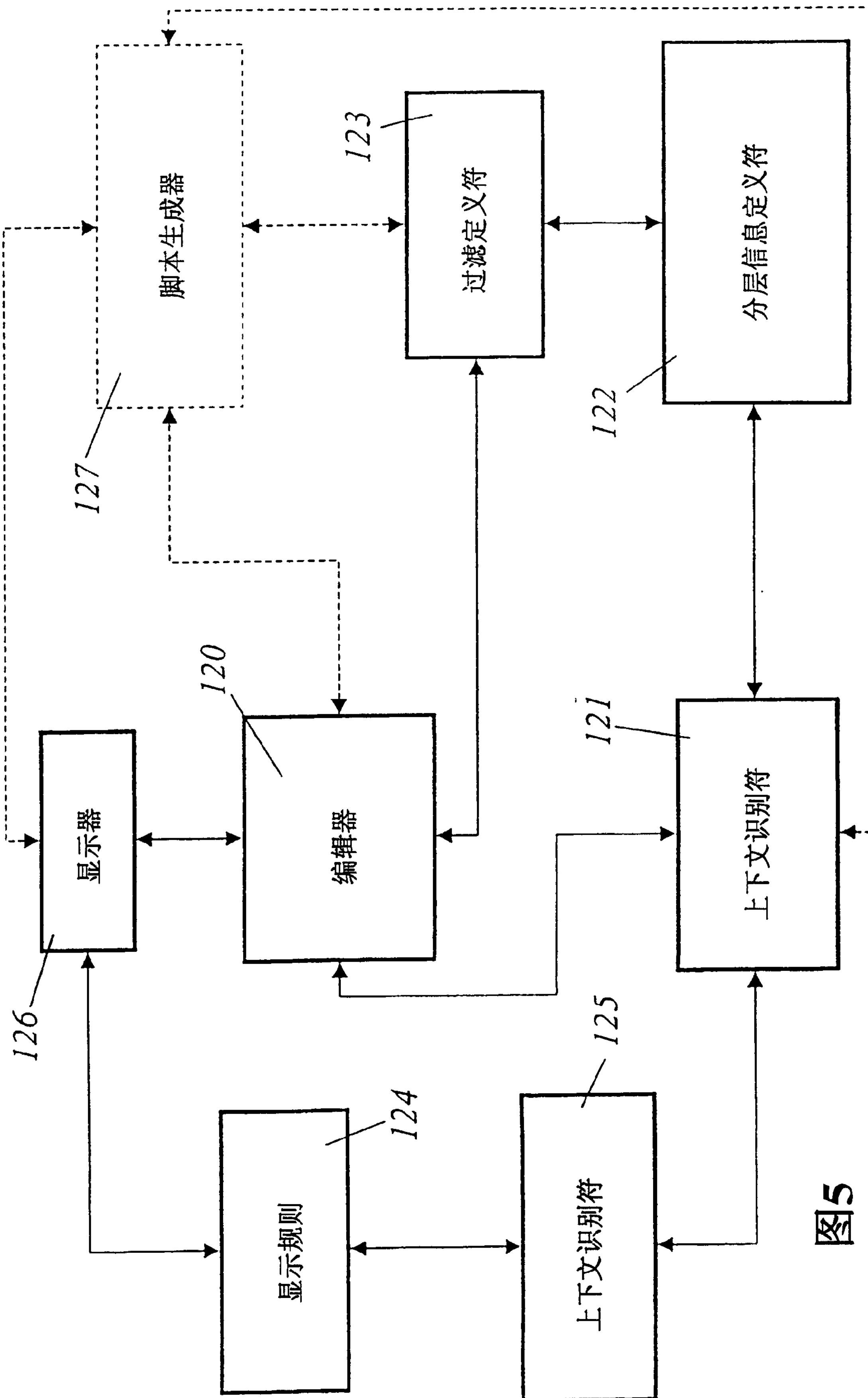


图5

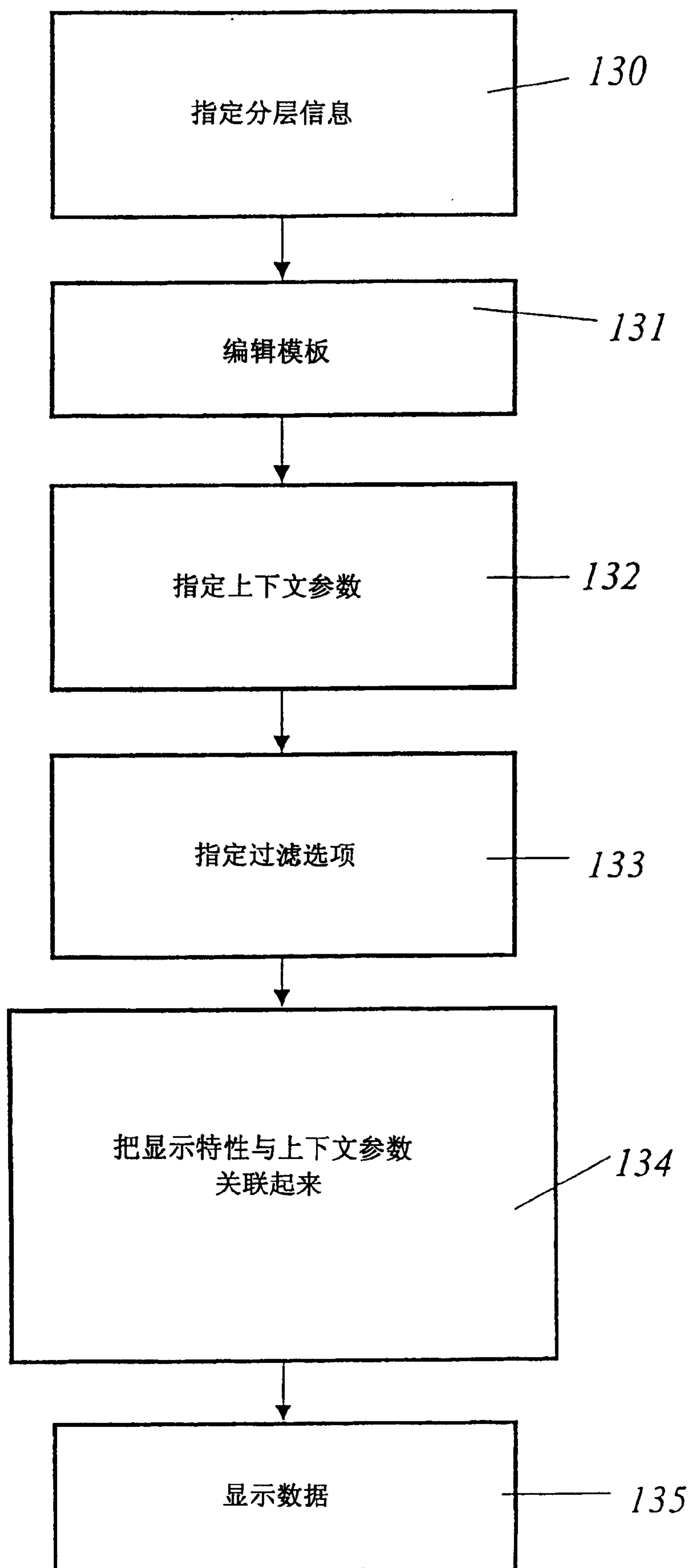


图6

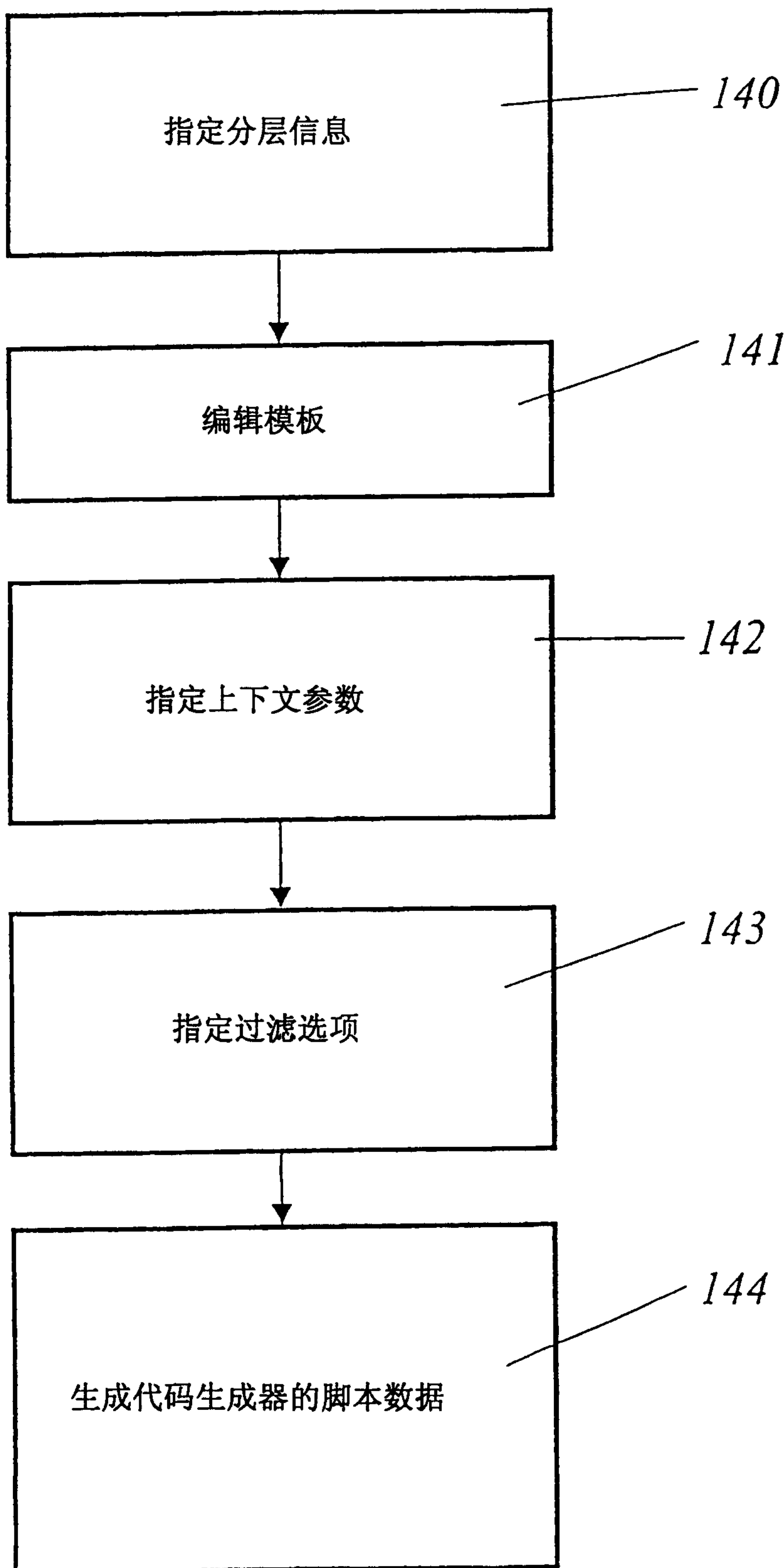


图7

<New Operation Signature> or <Operation Signature: [template name]>

Edit

Template name: 40

Operation signature

Visibility: public 41

static

Inheritance: 43

Return Type

Class Name: <attribute type> 44

Package: 45

Name: get<Attribute Name> Capitalize first letter 46

Paremeters

Name	Class	Package

Add...

Remove

Exceptions

Class	Package

Add...

Remove

OK Cancel ?

Save changes and close win

38

39

图8

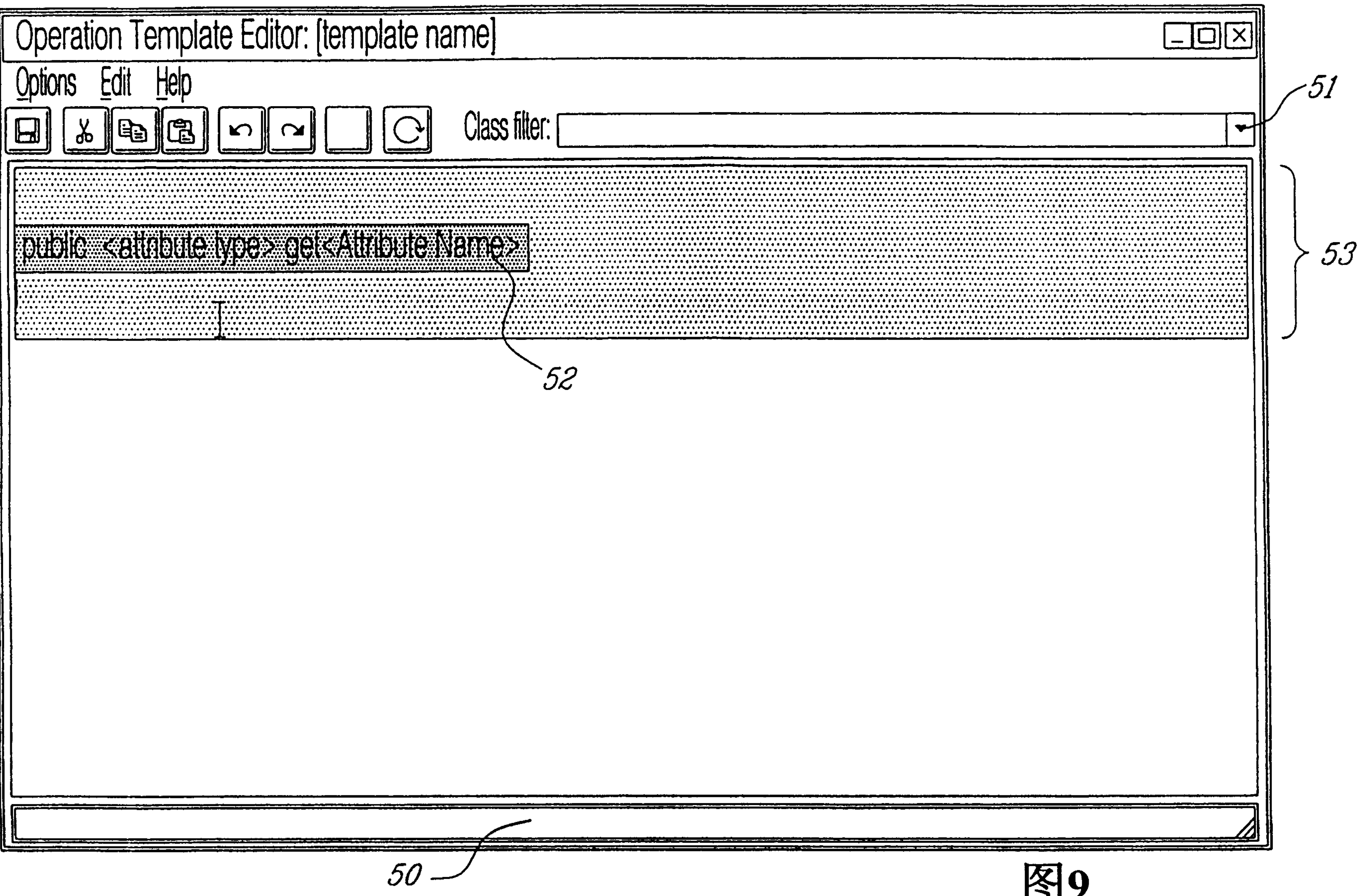


图9

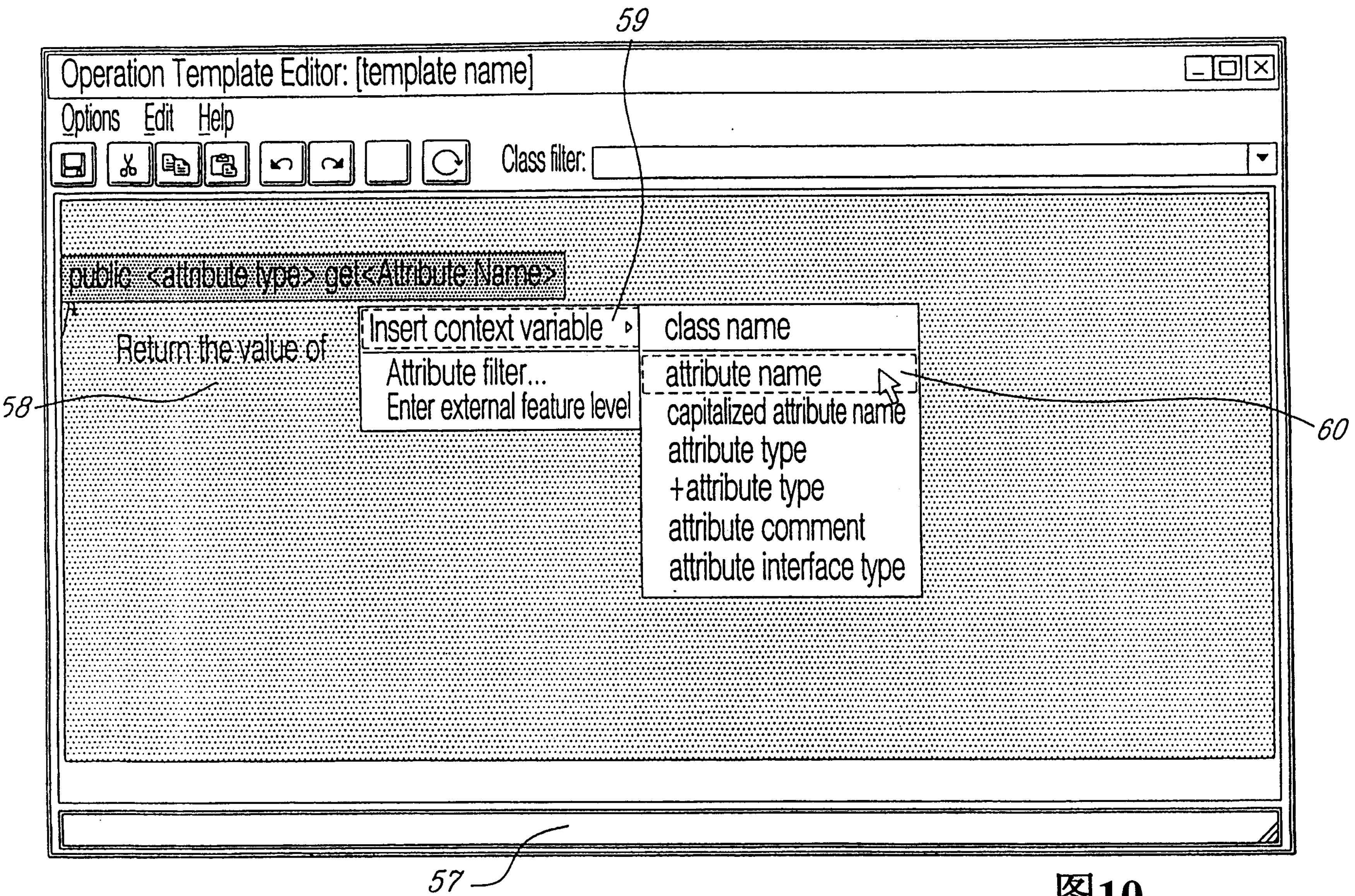


图10

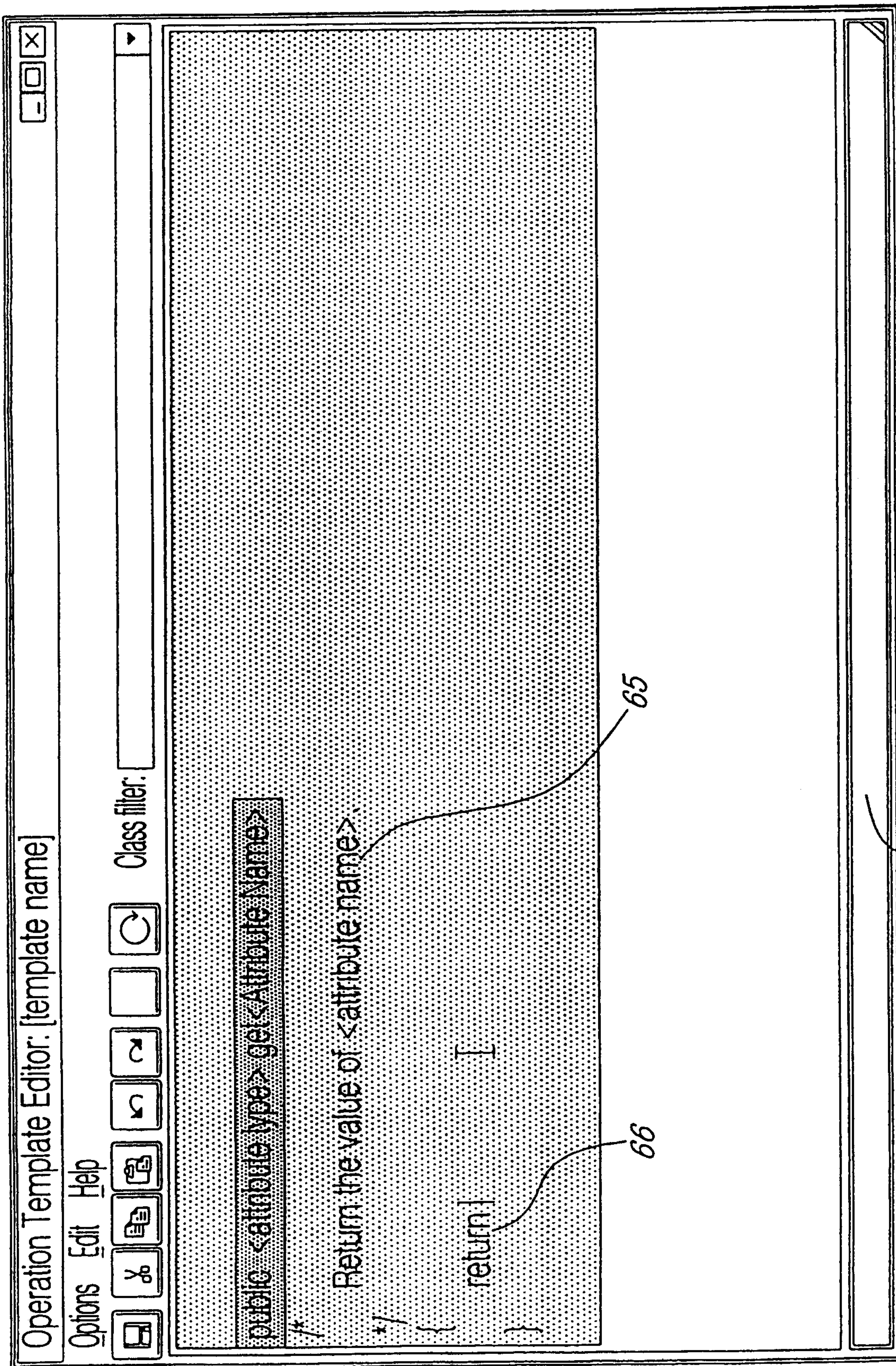


图11

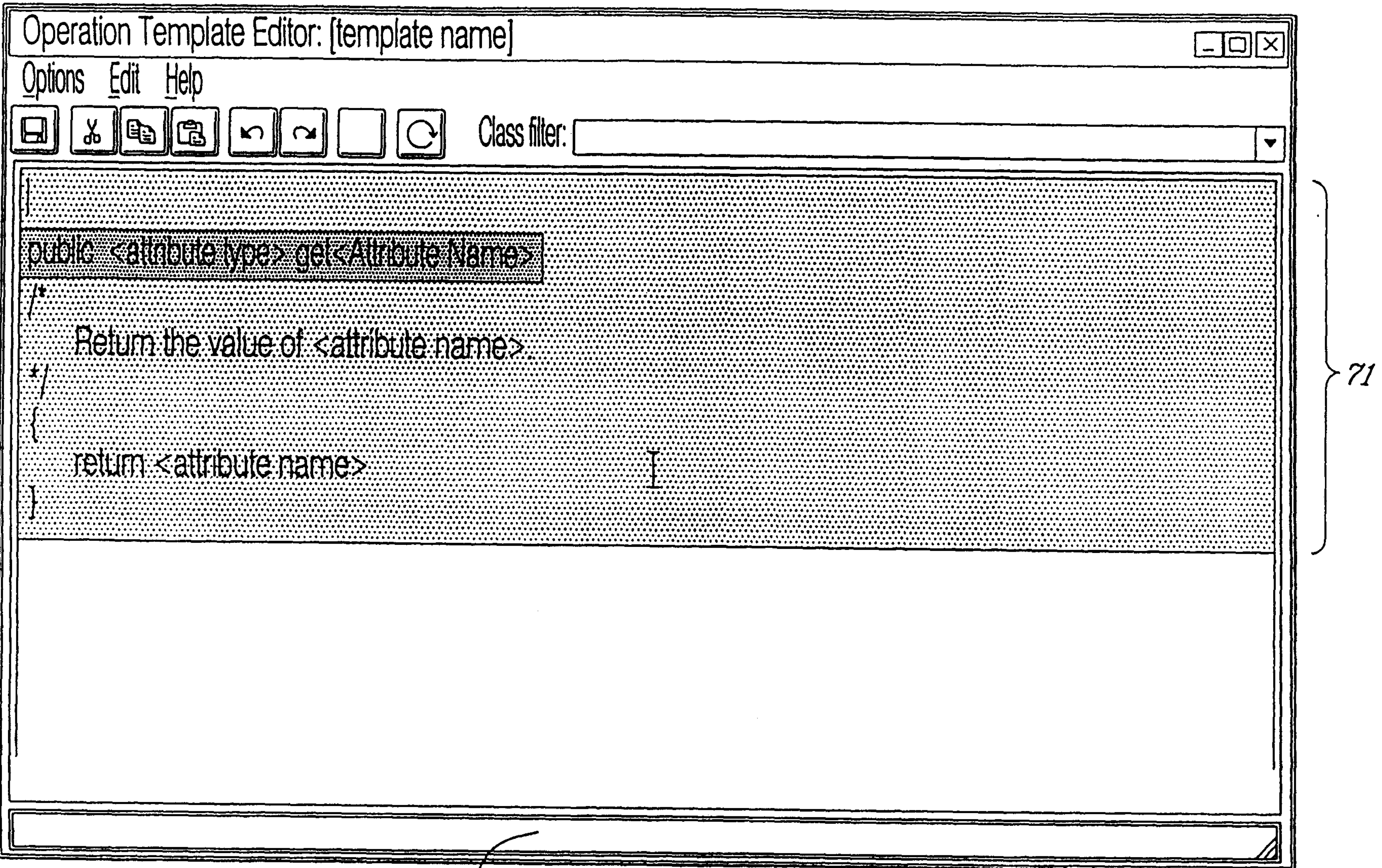


图12

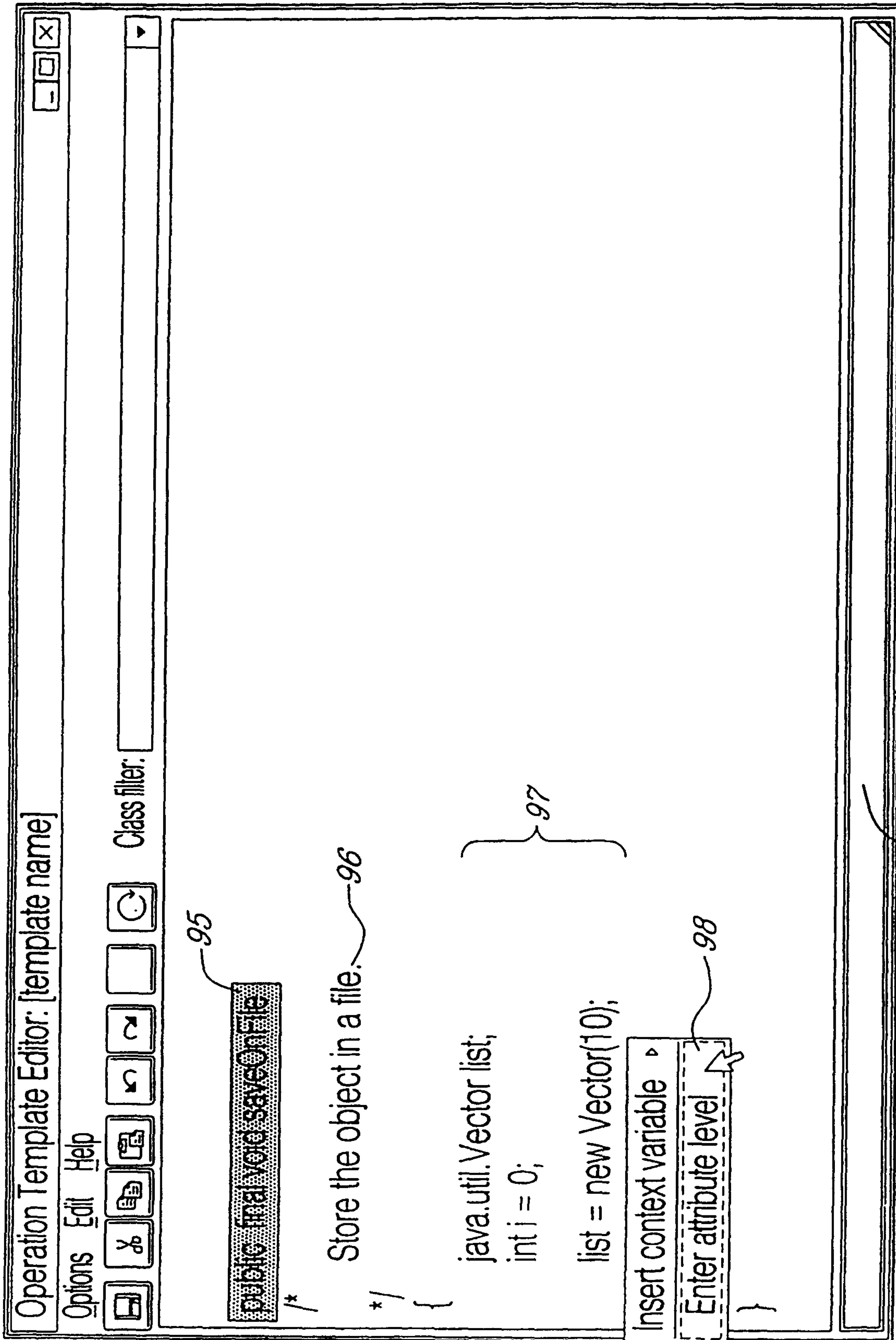


图13

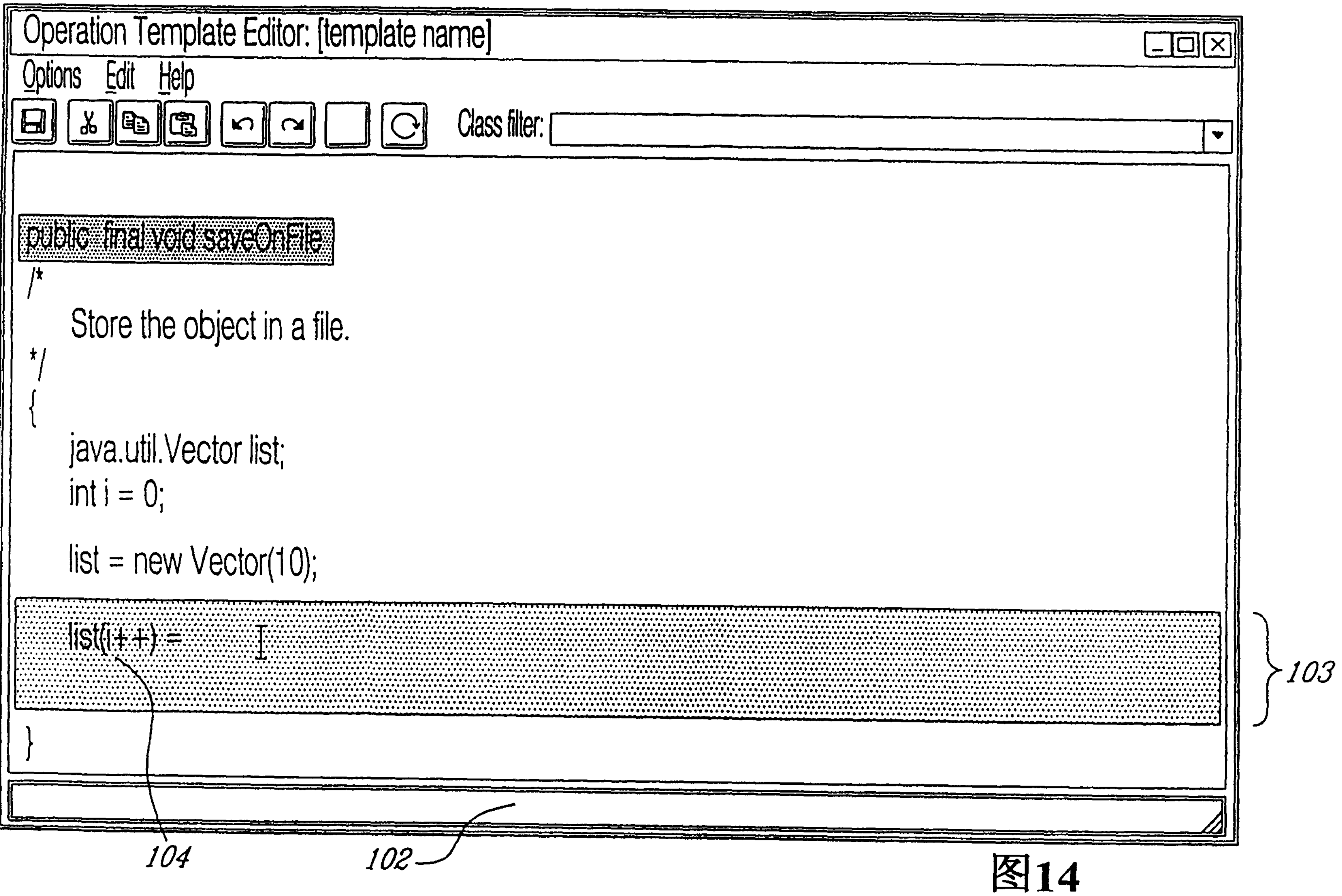


图14

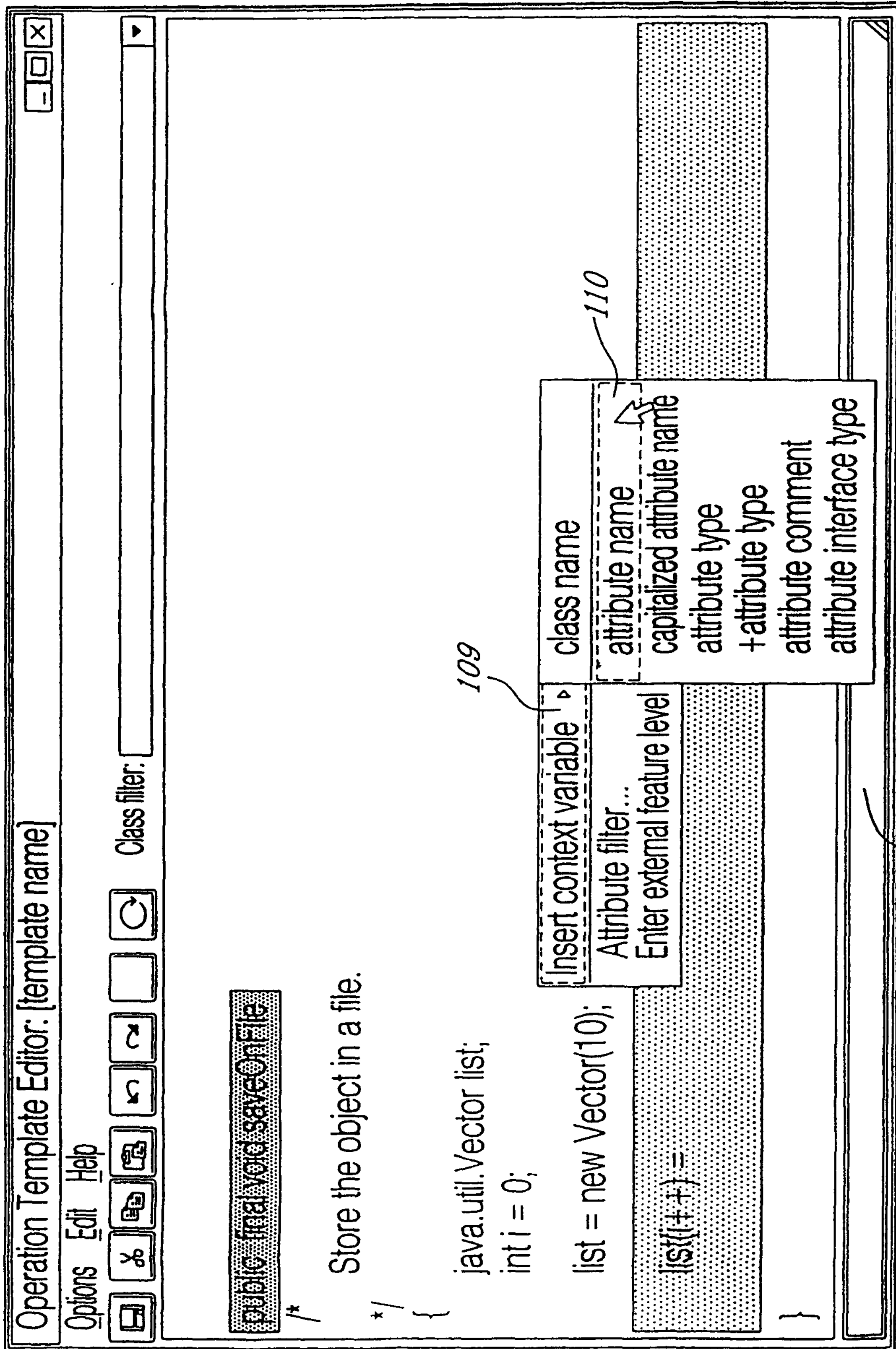


图15

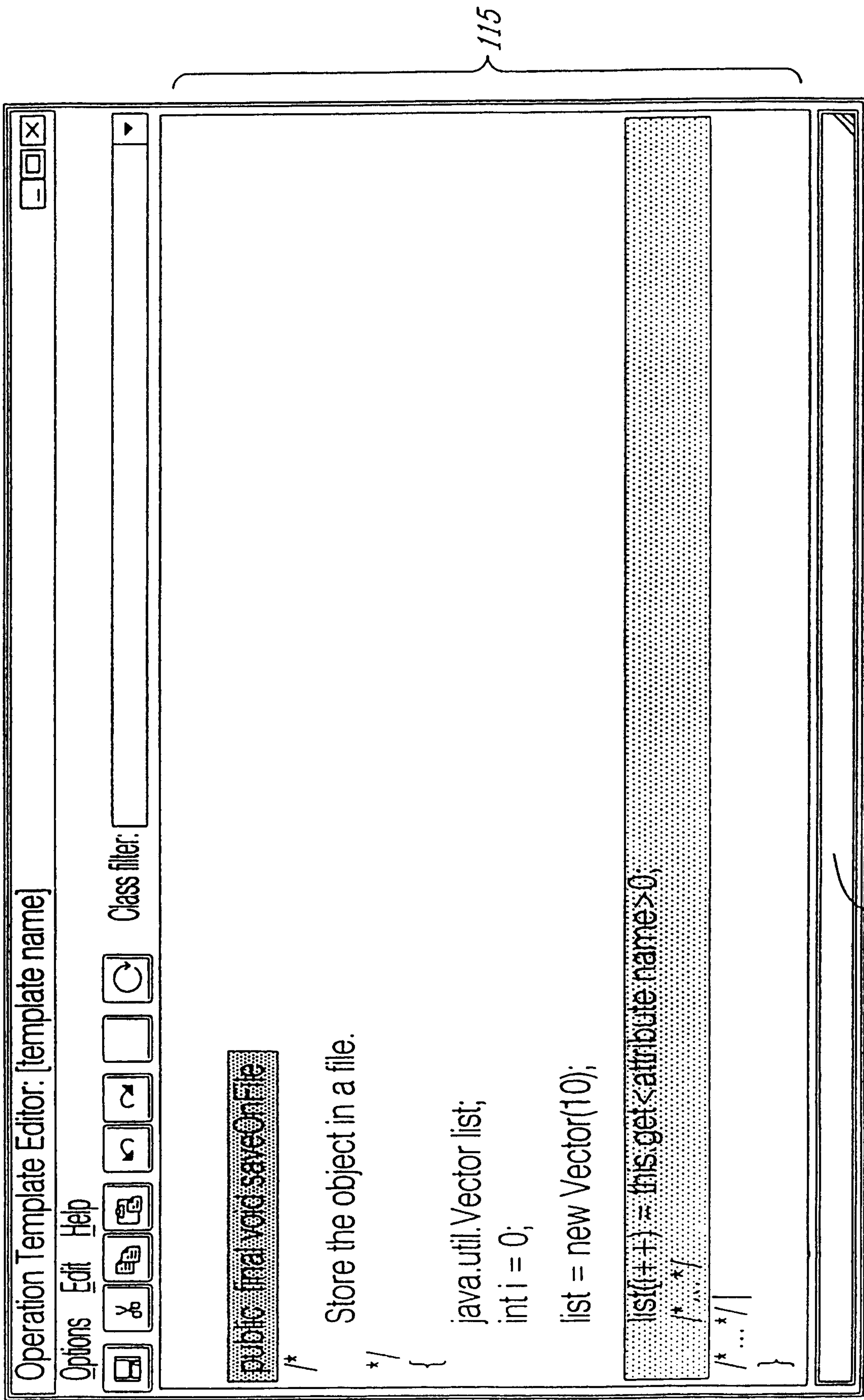


图16