[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 9/44 (2006.01)
G06F 17/30 (2006.01)
H04L 29/06 (2006.01)

[12] 发明专利申请公布说明书

[21] 申请号 200610093114.0

[43] 公开日 2007 年 1 月 10 日

[11] 公开号 CN 1892593A

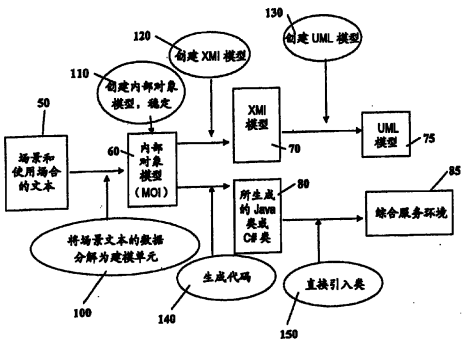
[22] 申请日 2006.6.21
[21] 申请号 200610093114.0
[30] 优先权
[32] 2005. 6.21 [33] FR [31] 0551697
[71] 申请人 阿尔卡特公司
地址 法国巴黎市
[72] 发明人 菲拉普·拉尔韦 帕特里克·方丹
[74] 专利代理机构 北京市金杜律师事务所
代理人 王茂华
权利要求书 3 页 说明书 30 页 附图 1 页

[54] 发明名称

与对象建模形式体系兼容的数据处理方法

[57] 摘要

本发明涉及一种产生新的 Web 服务的方法，包括步骤：将例如场景(50)和/或使用场合(50)之类的数据分解(100)为基于对象建模形式体系的新的 Web 服务的建模单元；根据该建模单元创建(110)内部对象模型(60)；以及将内部对象模型(60)的建模单元转换(120、140)为外部元模型的建模单元。后一步骤可以伴随生成代码(80)的步骤(140)或者经由 XMI 生成器创建 XMI 模型(70)的步骤(120)。本发明还涉及一种适合于实现本发明方法的计算机程序产品，以及一种通过该方法得到的计算机文件。



1. 一种产生新的 Web 服务的方法，包括步骤：

- 将文本描述分解（100）为符合对象建模形式体系的所述新的 Web 服务的建模单元；

- 根据所述建模单元创建（110）内部对象模型（60）；以及

- 将所述内部对象模型（60）的建模单元转换（120、140）为外部元模型的建模单元。

2. 根据权利要求 1 所述的方法，其中，在所述分解步骤（100）中，所述文本描述包括场景（50）和/或使用场合（50）。

3. 根据权利要求 2 所述的方法，其中所述分解步骤（100）包括对数据的语法和/或语义的分析。

4. 根据权利要求 3 所述的处理方法，其中所述分解步骤还包括在所述使用场合中识别：

- 对所述新的 Web 服务的一般描述；以及

- 应该由该服务使用的现有 Web 服务。

5. 根据权利要求 4 所述的处理方法，其中所述分解步骤还包括：

- 从所述使用场合识别所述新服务中所涉及的参与者；以及

- 从所述新的 Web 服务的所述一般描述提取所述新服务的输入－输出。

6. 根据权利要求 4 或 5 所述的处理方法，其中所述分解步骤还包括重新组织所述场景的短语。

7. 根据权利要求 6 所述的处理方法，其中所述分解步骤还包括根据重新组织的所述场景的短语来产生逻辑短语。

8. 根据权利要求 1-7 中的任一项所述的方法，其中所述外部元模型是 XMI 模型，并且所述转换步骤（120、140）是经由 XMI 生成器创建所述 XMI 模型（70）的步骤（120）。

9. 根据权利要求 8 所述的方法，在创建所述 XMI 模型的所述步骤（120）之后包括：

- 根据所创建的所述 XMI 模型（70）创建 UML 模型（75）的步骤（120）。

10. 根据权利要求 1-9 中任一项所述的方法，在所述转换步骤（120、140）期间或之后包括:

- 生成代码（80）的步骤（140）。

11. 根据权利要求 1-10 中任一项所述的方法，其中创建所述内部对象模块（60）的所述步骤（110）包括:

- 稳定所述内部对象模块（60）的数据结构（60）的步骤;

并且其中:

- 所述数据结构（60）包括多个子句;

- 将这些子句中的每个子句与所述形式体系的一个或多个建模单元相关联; 以及

- 这些子句中的每个子句全部是声明性的。

12. 根据权利要求 11 所述的方法，其中每个所述子句与一个单独的建模单元有关。

13. 根据权利要求 12 所述的方法，其中每个所述子句包括与其所关联的所述建模单元有关的代码以及与该建模单元有关的特征。

14. 根据权利要求 11、12 或 13 所述的方法，其中一些所述子句包括与其所关联的所述建模单元的容器有关的特征。

15. 根据权利要求 1-14 中任一项所述的方法，其中所述对象建模的形式体系是 UML 形式体系。

16. 根据权利要求 1-15 中任一项所述的方法，其中所述形式体系的所述建模单元中的一个建模单元是:

- 封装;

- 使用场合;

- 类;

- 属性;

- 关联; 或

- 继承。

17. 一种计算机程序产品，其适合于实现根据权利要求 1-16 中任一项所述的方法。

18. 一种计算机文件，通过根据权利要求 1-16 中任一项所述的方法来得到所述计算机文件。

与对象建模形式体系兼容的数据处理方法

技术领域

本发明涉及一种提供新的 Web（网络）服务的方法，一种适合于实现该方法的计算机程序产品(特别是使用场合辅助程序(use case assistant））），以及一种通过该方法得到的计算机文件。

背景技术

在下文中，除非以其他方式指明，否则就按以下所指明的意思使用以下术语:

- “电子数据处理应用程序”：适合于联合运行的任意程序或任意一组程序，其目的是向其用户呈递服务。

- “类”：类的概念通常应用于建模形式体系和面向对象的编程语言并且特别是适合于在电子数据处理应用程序中共同工作的数据和处理的集合（分别是属性和方法）。在面向对象的语言中，每种类型的对象由其类定义。一个类表示共享相同属性、操作、方法、关系和语义的所有对象。

- “行”：向目标程序，例如向编译器提供特定信息的符号语言的一个表达或一系列表达。

- “容器”：在面向对象的编程中，包括诸如目录之类的其他对象并提供用于访问该其内容的操作的对象。它还可以是包括其他组件的组件。

- “源代码”：用编程语言或描述语言编写的文本，该文本随后被编译为机器语言以便由硬件电子数据处理平台执行或者由软件环境或解译硬件来解译。

- “代码”：该术语可互换地意指源代码和编译代码。

- “编译”：在由电子数据处理硬件平台执行用编程语言表达的

描述之前，将该描述转换为该电子数据处理硬件平台的本机语言的动作。

- “声明性的”：未将数据与指明的或必需的使用该数据的顺序相结合。如果指明了该顺序，则这种表示就不再是声明性的而是过程性的（procedural）。在语言可以具有声明部分和过程部分的意义下，这两个概念一般会重叠。

- “DTD”：文档类型定义，指定文档逻辑结构模型，用于固定（fix）构成文档的单元以及连接这些单元的链接的性质。也可以使用术语“方案”。

- “代码生成”：通过生成器对用于电子数据处理应用程序的源代码的自动或半自动产生，从提供给生成器并控制生成处理的软件应用程序的抽象描述中得到。在分析该描述之后，该处理构造期望的输出代码。该描述通常用比待产生的代码中所用的语言更高级的语言表达。因此，通常选择与自然人类语言尽可能接近的图形形式或文本形式的语言。因此，可以在不需要获知所生成的代码中使用的编程语言的形式体系的情况下使用生成器。

- “继承”：链接两个类的关系，其中“基类”从“母类”继承特性（属性和方法），使得基类能够利用母类的定义来丰富其自身的定义，同时保持所有其原始的特征。继承是丰富（将所继承的类的特性添加到继承类）而不是耗尽（并非母类的所有特性都可以被继承）。

- “HUTN”：人类可用的文本的通知，意指用于利用设计为对用户来说通用的、可自动化的且可理解的人类可读句法（syntax）来创建并修改文本格式模型的 OMG（对象管理组）标准。HUTN 被认为比 XMI “更简单”且显然更具有可读性，原因是 HUTN 是面向“用户”的，与诸如 Java、C + + 之类的本领域中已知的语言类似。

- “实例”：一次出现、一个例子、一种类型或一个类的一个单独成员。通常用同义词“对象”和“实例”来指代类的成员。

- “编程语言”：用于描述将由计算机执行的动作的形式体系。

在编译或解译之后，这些描述由计算机执行。使用一种或多种编程语言来描述电子数据处理应用程序。术语“语言”是指用于描述某些软件或所有软件以便直接地或间接地促成其实际构造的所有表示格式。

- “元语言”：一种语言，对于其主题采用另一种语言并使该另一种语言正式化。

- “元建模”：使用另一种建模语言（元语言）对一种建模语言的说明，元模型是表示语言句法的模型。术语“建模”是指语句、问题或系统的抽象表示，一方面为便于理解以及传达该理解而执行，另一方面为解决该问题或者系统的具体实现而执行。

- “模型驱动体系结构”（MDA）：一组 OMG 规范，其涉及采用一系列模型的开发过程，该开发过程整合了项目的多个维度并从功能模型延伸到源代码模型。这些模型基于 UML 标准。从一个模型到另一个模型的递进（progress）包括对应用程序的说明的逐步丰富，以及使用由 UML 元模型认可的变换。

- “面向对象的建模”：一种特定类型的建模，其中建模单元是类、对象、属性、方法、关联(association)等，与“功能建模”相对，在功能建模中，建模单元是处理功能和数据流。也可参见“UML”。

- “MOF”：元对象工具，由 OMG 提出并被用于规定 UML 和其他建模语言的句法的元建模语言。

- “OMG”：对象管理组，一个组织，其目标包括定义标准以确保使用面向对象的语言编程的应用程序的相互兼容（参见 http://www.omg.org）。

- “对象”：在建模或面向对象的编程中，包括一组数据（类的内在定义数据）和用于操纵这些数据的过程的类实例。

- “面向对象”：是指例如其组件部分是类或对象的模型、语言、应用程序或应用程序单元。例如，面向对象的语言是一种编程语言，其中基本组件是类，类的实例，即对象动态地位于使用类的电子数据处理程序中。

- “封装”：首先是分组在一起以便再使用或改善模型的组织的一组建模单元（类或其他封装）。一旦已经将模型以编程语言进行了编码，则封装就指定包含类或其他封装的计算机的目录（文件）。

- “面向对象的编程”：一种特定类型的编程，其使用面向对象的语言（例如 Java、C + +、C#或 Eiffel）来操纵类和对象，与非面向对象的编程或“标准”编程相对，非面向对象的编程或“标准”编程只操纵简单的数据结构和单独的处理“功能”（例如使用 C、COBOL、FORTRAN 的编程）。

- “Web 服务”：可在因特网上经由标准接口访问的应用程序，该应用程序可以使用通信协议，例如基于 XML，独立于所使用的操作系统和编程语言动态地与其他应用程序进行交互。

- “SGML”：用于描述电子数据处理文档的内容与其结构之间的关系的标准语言。

- “定型”（Stereotype）：一种类型的建模单元，其扩展了元模型的语义。定型必须基于存在于元模型中的某些类型或类。定型能够扩展语义，而不是预先存在的类和类型的结构。UML 中预先定义了一些定型，其他的可以由用户定义。定型连同“标记值”（tagged value）和“注释”一起构成用于扩展 UML 的三种机制之一。

- “UML”：统一建模语言，一种使用对象的建模表示法（而不是建模语言），使得能够在目标系统的开发期间确定和呈现目标系统的组件，以及在适当的时候生成系统文档。UML 当前是 OMG 标准。UML 是通过 Jim Rumbaugh、Grady Booch 和 Ivar Jacobson 共同工作得到的，并且以各种方式演进。

- “XMI”：XML 模型互换，用于规定元模型与“方案”之间的对应关系的规则。XMI 是万维网联盟（W3C）的将模型领域与 XML 领域相联系的 OMG 标准。XMI 用于以 XML 文件的形式表示 UML 模型。

- “XML”：SGML 的一种演进，其特别地使得 HTML 文档的设计者能够定义其自己的标志，目的是对数据结构进行个性化。

对象建模是本领域中已知的并且包括独立于任意编程语言、通过类和对象来创建真实世界的单元的表示。例如，确定对象的类，分离它们的内部数据和使用它们的功能。有不同的形式体系。UML是这些形式体系中的一个（并且实际上更像一类表示法）。UML表示法是在20世纪90年代成为OMG标准的。

面向对象语言均构成实现类概念的特定方式。特别地，对象形式体系用于以“高级抽象”来定义问题，而不需要进入给定语言的细节。UML形式体系提供了一种用于以图形形式容易地表示问题的工具，使得可以更容易访问其解决方案中的各参与者。从这一方面讲，希望尽可能严格地定义对象形式体系并且应当优选地对其进行唯一定义，以便使模糊度最小化。

为解决给定问题，第一步通常是得到对象模型的抽象概念。然后，使用面向对象语言（诸如C#或Java）来实现该模型。

本领域中还已知Web服务（WS），其可以从任意因特网终端访问，并且越来越多地用于生成简单且可再使用的应用程序。

从其复杂性的角度来说，可以将这些WS分为两类：基本WS和复合WS。

基本WS提供与出现在数学库中的服务相当的基本服务，并且包含低级的数据转换，包括少量算法。例如，通常将转换服务视为基本WS。

另一方面，通过多个基本服务的协作，复合WS能够提供高级服务并且具有多个访问和数据转换级别。复合服务有时称为“配合服务”（或“交互处理”），意指所需的所涉及基本服务的配合。预留服务和安全支付服务是配合复合WS的例子。

尽管在诸如Apache/Axis或.NET平台之类的本领域中已知的标准环境中可以相对容易地建立和部署基本服务，但有益的是具有能够实现由对现有服务的配合集合得到的复合WS的产生和部署的适当的环境。

然而，由于将继承服务进行了集合和配合，继承服务的交互，以

及应当执行继承服务的方式以达到其目标并在确保与终端服务的功能一致性的情况下提供该终端服务，这种环境会引发问题。

从这一方面讲，目前存在多种集合技术。尽管在业界仍然没有关于公共语言的一致意见，但有两种被认为是互补的语言：

- WSBPEL（Web服务业务流程执行语言）或BPEL，其描述了WS之间的交互，包括这些交互的应用程序逻辑和顺序；以及

- WS-CDL（Web服务编排描述语言），其描述了在WS之间交换的消息，包括关于这些交换的顺序和约束。

然而，使用BPEL并不是描述应用程序逻辑的唯一方式，还可以使用Java或C#。

因此，需要一种产生新的Web服务的方法，该方法确保服务的功能结合并且不会施加与最终的实现语言相联系的约束。

发明内容

为此，本发明提出一种产生新的Web服务的方法，包括步骤：将文本描述分解为基于对象建模形式体系的新的Web服务的建模单元；根据建模单元创建内部对象模型；以及将内部对象模型的建模单元转换为外部元模型的建模单元。

本发明方法的优选实施例包括以下特征中的一个或多个特征：

- 在分解步骤中，文本描述包括场景和/或使用场合；

- 分解步骤包括对数据的语法和/或语义的分析；

-分解步骤还包括在使用场合中识别：对新的Web服务的一般描述；以及可能将由该服务使用的现有Web服务；

- 分解步骤还包括：从使用场合识别新服务中所涉及的参与者；以及从新的Web服务的一般描述中提取新服务的输入－输出；

- 分解步骤还包括重新组织场景的短语；

- 分解步骤还包括根据重新组织的场景短语来产生逻辑短语；

- 外部元模型是XMI模型，并且变换步骤是经由XMI生成器创建XMI模型的步骤；

- 该方法在创建 XMI 模型的步骤之后包括：根据所创建的 XMI 模型创建 UML 模型的步骤；

- 该方法在转换步骤期间或之后包括：生成代码的步骤；

- 创建内部对象模型的步骤包括：稳定内部对象模型的数据结构的步骤，并且数据结构包括多个子句，将这些子句中的每个子句与形式体系的一个或多个建模单元相关联，并且这些子句中的每个子句是完全声明性的；

- 每个所述子句与一个单独的建模单元有关；

- 每个所述子句包括与其所关联的建模单元有关的代码以及与该建模单元有关的特征；

- 一些所述子句包括与其所关联的建模单元的容器有关的特征；

- 对象建模的形式体系是 UML 形式体系；

- 形式体系的建模单元中的一个建模单元是封装、使用场合、类、属性、关联或继承。

本发明还涉及一种适合于实现本发明方法的计算机程序产品。

本发明还涉及一种通过本发明方法得到的电子数据处理文件。

附图说明

通过阅读以下对仅通过示例给出的本发明实施例的详细描述并参考示例和附图，本发明的其他特征和优点将变得明显，其中：

图 1 是本发明方法的一个实施例的特定步骤的示图。

具体实施方式

本发明提出了一种产生新的 Web 服务的方法，一种适合于实现该方法的计算机程序产品，以及一种通过该方法得到的电子数据处理文件。

本发明的方法包括第一步骤，即将文本描述（通常是场景和使用场合）分解为对象建模形式体系的建模单元，该对象建模形式体系例如 UML 形式体系。场景和使用场合就现有服务的操作之间的交互

而描述了未来的“交互处理”的详细工作方式。使用场合类似于一般描述，而场景与对特定场合的详细的、功能性的描述有关，这是本领域中已知的。如有必要，该方法可以使用多个场景文本。

可以在该第一步骤期间或之前以半正式文本的形式表达场景和使用场合。该步骤优选地采用语法和/或语义分析。如有必要，则在该第一步骤期间，识别对新的 Web 服务以及可能将由该新服务使用的现有 Web 服务的生成描述。

该方法还包括一个步骤，即从建模单元创建内部对象模型（MOI）。该 MOI 用于进行存储（stacking），并且优选地是屏蔽的，即对于用户是透明的。

之后是将内部对象模型的建模单元转换为外部元模型的建模单元的步骤。例如，该操作可以伴随产生采用任意对象语言（Java、C++、C#、Ada 等）的代码的步骤或创建 XMI 模型的步骤。

采用 MOI 确保了未来服务即事务处理的功能一致性，并保证了从一个模型到另一模型的转换运行良好。为此，服务构造机制优选地基于 MDA 方法。注意，所选的方法是不可知的（因为建模单元在对象意义上的使用）。此外，包含在数据中的描述（使用场合/场景）是功能性的。因此，该方法不会引起与最终的实现语言相联系的任何约束。

如果需要容易地既产生例如基于 UML 的对象模型又产生采用任意对象语言的源程序，则采用 MOI 也可证明是有利的。然后，MOI 用作使用多种语言的对话者。从这一方面讲，特别有利的是 MOI 基于特别适合于表示 UML 模型或特别适合于容易地生成源程序的特定数据结构。

一旦产生，这些源程序就可以被部署。这样就得到一种环境，在该环境中可以容易地部署最终复合的服务，以使得该服务对终端用户可用。

下文中将通过本说明书第一部分中的示例来对本发明方法的优选实施例进行说明。在本说明书的该第一部分中，重点在描述分解

数据的步骤和创建MOI的步骤上。然后，在本说明书的第二部分中，更详细地描述了特别有利于MOI的数据结构。该数据结构基于在下文中称为“FIL”表示法（FIL意指“格式互换语言”或“格式内部建模语言”）的表示法。本说明书的第三部分针对源代码的产生并针对服务的部署。

I. 借助于通过使用场合辅助程序而实现的本发明方法的示例进行描述

接下来将联系简单的事务处理示例来描述上述一般概念。

假定Web服务库包括多个基本服务：

- 消息传输服务，用于向给定接收方发送SMS消息、传真或电子邮件；

- 目录服务，即用于获得特定于给定用户的数据（例如其电子邮件地址或其联系人列表）的一般服务；以及

- 用户选项服务，其管理与每个用户相关联的选项。

I.0 对新服务的描述

参考图1,采用上述基本组件，现在假定需要建立新的服务，第一参与者（分析者）通过以下使用场合50不严密地描述到：

使用场合：

新服务的名称是：广播消息。采用该新服务，用户希望根据其消息传送选项向多个接收者广播消息。该新服务使用某些现有服务，但在这一级别上，我（分析者）忽略了这些现有服务的准确名称……。

场景：（对新服务的描述）

用户给出他的名字，并且该新服务得到该用户的相应的联系人列表。然后，对于该列表上的每个联系人，该新服务得到对应于该联系人的消息传送类型，并且根据该类型，该新服务向该联系人发送SMS、传真或电子邮件。

适合于实现根据本发明的方法的使用场合辅助程序实现了将以上数据分解（100）为基于例如UML之类的对象建模形式体系的新的Web服务的建模单元。

这种分解（100）优选地包括多个转换，在下文中将对其进行描述。

根据以上不太严密的描述，使用场合辅助程序执行一系列逐步的转换，直至其得到（120、140）诸如XMI之类的外部元模型（70）的单元或对应于新服务的可编译的、可执行的且准备好进行部署的源代码（80）。

I.1 对应于新服务的文本到半正式模型的第一转换

可能将由辅助程序实现的第一转换包括对文本的重新布置，以便赋予该文本可以由计算机“处理”的形式（在本领域中将其称为“机器可用的”形式）。这种转换可以包括两个子步骤：重组（reformulation）的第一子步骤以及在半正式模型中进行填充的第二子步骤。

对文本进行重组的第一子步骤包括对文本的语法分析。这种分析由例如能够实现短语重构的辅助程序的适当的语法分析模块来实现。

从这一方面讲，注意到在本领域中已知有用于实现文本的语法分析的工具，例如Grammatica，其是用于采用法语或英语的文本的语法和拼写检查器。除建议之外，Grammatica还提供解释。Grammatica分析单词的功能（文本和语法联系，同音字和同形异义字，同义字和在法语中的定义）。参见站点www.zdnet.fr。另一个示例是来自Lexiquest的工具，其执行文本分析和文本挖掘：www.lexiquest.com。

辅助程序的语法分析模块通常能够：

- 清楚地分辨数据的“使用场合”部分与“场景”部分；
- 在“使用场合”部分中识别以下子部分：

- 新服务的名称；
- 对该新服务的一般描述；以及
- 对可能将用于设计该新服务的现有服务的识别；以及

- 在“场景”部分中：
  - 将文本分解为单独的短语；
  - 识别这些短语，以便为其赋予短语的第一编程指令轮。

在上述操作之后，得到一个结果，例如：

使用场合

新服务的名称：广播消息。

一般描述：[采用该新服务，]用户希望根据其消息选项向多个接收者广播消息。

所使用的现有服务：新服务使用某些现有服务，但[在这一级别上，我（分析者）忽略了这些现有服务的准确名称……。]=在该步骤中，这些现有服务还是未知的。

场景

详细描述：

- 用户给出他的名字；
- 该新服务得到该用户的联系人列表；
- 对于该列表上的每个联系人：
  - 该新服务得到对应于该联系人的消息类型；
  - 根据该消息类型：
    - 该新服务发送 SMS；或
    - 该新服务发送传真；或
    - 该新服务向该联系人发送电子邮件。

第二子步骤必须有对文本中的有关信息的详细提取，以便在半正式模型中进行填充，然后将该半正式模型用于生成属于新服务的源代码。

采用适当的语法提取器，实现以下操作。

- 在“使用场合”部分中：
    - 识别新服务的名称；
    - 识别参与者：初始参与者和最后的参与者（如果合适的话，则有多个）；
    - 提取对该新服务的一般描述（如果可能的话）；*
    - 提取通常以“用户将……给服务”、“服务向用户返回……”等表示的服务的主要输入－输出。
- 在“场景”部分中：
    - 根据单独的短语，生成将要变为指令的“逻辑短语”，即独立短语或短语的一部分；
    - 识别必要的编程单元（例如“if … then … else ”、“switch … case ”、“ for each …”等）。

将这些操作的结果存储于半正式模型的相应字段中，例如，如下所示。

用于消息业务处理的使用场合

参与者

参与者是：用户（消息广播者）和接收者。

使用场合

影响参与者的使用场合：用户（消息广播者）

使用场合名称：广播消息

服务名称：　　消息业务处理

描述：用户希望根据其消息选项向多个接收者广播消息。

输入：用户将以下数据给服务：

- 他的发送者名称；
- 消息。

输出：服务向用户返回：

- 无特定数据。

异常: 现在没有异常。

所使用的现有服务: 新服务使用某些现有服务，但在该步骤中，这些现有服务还是未知的。

场景

用于新服务的场景: 广播消息

* 该新服务根据该用户的名字得到发送者的联系人列表
* 对于该列表的每个联系人:
* 得到对应于该联系人的消息类型
* 根据该消息类型:
* 如果是 SMS:
* 向该联系人发送 SMS
* 如果是传真:
* 向该联系人发送传真
* 如果是电子邮件:
* 向该联系人发送电子邮件

这种半正式描述特别地具有两个优点: 用户可以读取该描述并且在该阶段由计算机来处理该描述是实际可行的。在该阶段可以添加三类单元以使得计算机能够使用该描述:

- 将使用的现有服务的名称或标识符;
- 可能将由这些服务使用的方法的名称或识别标志(对象意义上的);
- 这些方法所使用的参数的名称和类型。

在该实施例和本例中，在下文中描述的第二转换期间添加这些单元。然而，可以考虑多种实施例，其中可以默认地或直接由用户本身指定这些单元(用户本身直接指定这些单元的情形将构成非自动的操作模式)。

I.2 正式模型的确定

第一转换采用了语法分析。接下来将描述的内容更多地基于语义分析。为了能够描述XMI类型的服务，在服务库中使用例如本领域中已知的WSDL（关于该主题，参见www.w3.org并且特别是www.w3.org/TR/wsdl）描述了现有的（继承）服务。

基于给定Web服务的WSDL描述，使用场合辅助程序的专用模块优选地执行以下操作：

- 其查看场景的伪指令（即半正式指令）；

- 其从这些指令中提取“服务条件”，即关于必须适合于现有服务的服务的要求；

- 其将这些条件与服务方法的识别标志和参数（假定它们是用WSDL描述的）相比较；

- 如果在原始要求与属于继承Web服务的方法的识别标志之间找到令人满意的匹配，则其用服务方法的“真实”识别标志来代替条件。

如有必要，则该转换模块可以附加地完成该半正式模型的“所使用的现有服务”信息。

使用场合

……

所使用的现有服务：新服务使用以下现有服务：

* 目录服务；

* 消息传送服务

* 用户选项服务。

场景

用于新服务的场景：广播消息

* 利用参数“发送者名称”，该新服务通过调用目录服务的GetEmailAddress（得到电子邮件地址）操作来得到发送者的电子邮件地址。

* 利用参数“发送者名称”，该新服务通过调用目录服务的GetMyContactList（得到我的联系人列表）操作来得到发送者的联系人列表。

* 对于发送者的联系人列表的每个单元，新服务工作如下：

* 利用参数“联系人名称”，该新服务通过调用用户选项服务的GetMessagePreference（得到消息选项）操作来得到消息类型，

* 根据该消息类型：

* 如果是 SMS:

* 利用参数“联系人移动电话号码”和“消息”，调用消息服务的 SendSMS（发送 SMS）操作

* 如果是传真：

* 利用参数“联系人传真号码”和“消息”，调用消息服务的 SendFax（发送传真）操作

* 如果是电子邮件：

* 利用参数“联系人电子邮件地址”和“消息”，调用消息服务的 SendMail（发送邮件）操作

注意， SendMail、 SendSMS 等操作是本发明意义上的建模单元。

在这些新的附加之后，如有必要，辅助程序可以测试通过将由第二转换模块所选择的识别标志和参数与继承目录服务的原始 WSDL 描述相比较而得到的描述的一致性。例如，使用辅助程序的特定命令来找到并存储场景的原始文本所需的服务描述，以便之后将它们与由第一转换的自动模块所选择的 WSDL 描述进行比较。

在该一致性测试之后，开发者可以例如校正由辅助程序实现的选择并替代性地施加他自己对服务的选择。

I.3 场景的内部对象模型的产生

由辅助程序所使用的根据本发明的方法包括中心步 110，即从新的 Web 服务的建模单元创建内部对象模型 60。

例如，该步 可以使用“内置文本到正式语言”的转换模式以及“WSDL 到正式语言”的转换模式。因此，这些模式的规则将会把使用场合和场景的半正式文本以及继承 WSDL 服务的描述转换为正式的 MOI。

例如，MOI 60 可以采用在本说明书的第二部分中详细说明的特定数据结构（FIL）。如上所述，MOI 特别地用于容易地生成代码，同时确保涉及不同的转换时的功能一致性并从而保证最终服务的一致性。

在“应用”“文本到 MOI”转换（即本实施例中的“文本到 FIL”的转换）之前，优选地将基本模式用于由操作于从第二转换得到的场景文本上的关键字处理，以便将可以由用户读取的伪指令转换为可由计算机执行的指令。

为此，可以首先去掉诸如“该”、“服务”、“调用”、“操作”、“参数”等不重要的词语。然后，关键字处理模式可以例如替换预定的关键字（诸如“得到”、“利用”、“从”、“的”等）并将每个伪指令重新组织为最终的编程指令形式。

考虑例如来自上述场景中的以下短语：

利用参数“发送者名称”，该新服务通过调用目录服务的 GetEmailAddress 操作来得到发送者的电子邮件地址。

该短语在“开发者”的领域中具有已知的匹配。例如：

- 通过调用由开发者编写的操作“B”来得到数据“A”：A = B();

- 将服务“S”的操作“B”写为：S.B()。

因此，在开发者的领域中，上述短语就转换为：

```
sender_email_address = DirectoryService.GetEmailAddress (sender_name);
```

在由关键字处理模块执行的操作之后，得到：

```
Scenario for the new service: Broadcast Message
sender_contact_list = DirectoryService.GetMyContactList (sender_name);
for each element of sender_contact_list:
messaging_type = UserPreferenceService.GetMessagePreference (contact.name);
switch (messaging_type):
case SMS: messagingService.SendSMS (contact.Mobile_phone_number, message);
case Fax: MessagingService.SendFax (contact.Fax_number, message);
case Email: MessagingService.SendMail (contact.Email_address, message);
```

现在仍然应用“文本到 MOI”的转换模式，其代表第三级转换（实际上，在该级别上，更多的是一种翻译），以 FIL 格式产生使用场合和场景的完整半正式文本。以这种方式（提取），得到：

```
* 正式内部对象模型语言
* 对于新服务广播消息
* --------------------------
* 封装声明
p: Messaging Business Process
p: MessagingService
p: UserPreferenceService
p: DirectoryService

* 类声明
c: MessagingService, MessagingService
c: DirectoryService, DirectoryService
...
* 联合声明
r: MessagingBusinessProcess (1..1) uses (1..1) DirectoryService
(...)
r: DirectoryService (1..1) uses (1..n) ContactInfo
```

```
* 属性声明
a: Name:string, ContactInfo
(...)
a: theDirectoryService:DirectoryService, MessagingBusinessProcess
a: type_of_messaging:String, MessagingBusinessProcess


* 操作声明
o: GetUserProfile(userName:string) :UserProfile; UserPreferenceService
o: GetMyContactList (username:string) :ArrayOfContactInfo, DirectoryService


* 操作声明 + 操作详细描述
o: BroadcastMessage (String sender_name, String message); MessagingBusinessProcess
display_message ("The user wants to broadcast a message to several Receivers")
sender_contact_list = theDirectoryService.GetMyContactList (sender_name)
for (i<sender_contact_list.Length)
type_of_messaging =
theUserPreferenceService.GetMessagePreference (sender_contact_list [i] .Name)
switch (type_of_messaging)
case Sms: theMessagingService.SendSMS (sender_contact_list [i] .Mobile.message)
case Fax: theMessagingService.SendFax (sender_contact_list [i] .Fax.message)
case Email: default: the MessagingService.SendMail (sender_contact_list[i] .EMail,message)
end switch
end for
* FIL 结束
```

以“*”开头的行是注释。关键字母后接区分新服务（在可应用时涉及现有服务（在本例中是“DirectoryService”））的建模单元的冒号“:”（例如关系

“MessagingBusinessProcess (1..) uses (1..1) DirectoryService”）。

FIL 表示法反映了新服务的内部模型，新服务的内部模型以正式的方式表达并存储于例如使用场合辅助程序的专有数据库中。本说明书的下一部分将对该表示法进行说明。

II. 基于 FIL 表示法的内部对象模型的数据结构

II.0 上下文、定义

通过对 FIL 格式的数据结构的初步详细描述，有必要指出，当前已标准化的 UML 表示法不具有用于表示对象模型的文本表示法。该标准中只提出了针对各种 UML 图表（类、交互、状态－转换等的图表）的图形表示但在 UML 标准中没有提出该图形表示的对应文本。XMI 并非旨在“可读”，即对于人类用户不是可理解的。此外，XMI 不是自包含的（self-contained）。HUTN 句法的设计目标是可被用户理解。HUTN 仍然是面向过程的语言，即行的顺序对该语言的解译来说是非常重要的。

在这种背景下，特别希望具有一种数据结构，该数据结构能够由计算机处理，能够用于表示 UML 模型，并且能够容易地生成采用任意对象语言的源程序。同样希望该数据结构是简单的且便于操纵，并且一旦具体实现，就可以被用户理解。

术语“数据结构”首先是指在使用数据的程序的内部表示中所用的该数据的实际结构。然而，术语“数据结构”同样是指文件中所存在的信息的结构，正如在编辑或打印文件时所显示的那样。该文件可以是例如以其第一种意义来理解的上述“数据结构”的具体实现文件。

II.1 FIL 数据结构的一般特征

该数据结构是包括多个建模单元的对象建模形式体系的表示。表述“对象建模形式体系的表示”意指该数据结构可以特别地用于提供基于该形式体系的图形表示。可以将该形式体系的图形表示例如限制为 UML 表示法，在此情况下，建模单元是 UML 通常使用的单元，即封装、类、属性、操作、关联、基数、继承等。

所提出的数据结构包括多个子句，例如逻辑子句，该逻辑子句可以由程序来稳定或使用。这些子句中的每个子句与形式体系的一个或多个建模单元相关联，并且这些子句中的每个子句是完全声明性

的。

该数据结构是简单的。一旦将该数据结构具体实现在例如文件中，则每个子句就使用正式表示法变为该文件的一行。现在，假定每个子句是完全声明性的，则这些子句的顺序不会影响这些子句的解译，并且因此文件的行的顺序不会影响这些行的解译。这简化了用户对文件的理解。

每个子句优选地只对应于一个建模单元，这既简化了对数据结构的算法处理又简化了用户对由该数据结构构造的文件的理解。

此外，每个子句优选地是自包含的，以使得能够对其进行解译而不需要为该目的而访问某个其他的子句。

数据结构用高级的正式表示法来表示。这使得能够使用简单的且集成的数据处理方法。例如，用于转写（transcribe）并表示 UML 模型的方法，该 UML 模型通常是以与非正式文本相关联的图表和图形方案的形式描述的并且该 UML 模型的结构未标准化。还通过适当的电子数据处理将该数据结构用于容易地产生基于 UML 的对象模型和采用任意对象语言（Java、C++、C#、Ada 等）的源程序。稍后将在本说明书中对此进行详细描述。

因此，该数据结构有利地替代了当前提出的备选方案，该备选方案基于 XMI/XML 或基于 HUTN，没有提供同时是简单、便捷且能够容易地生成代码的工具。

为便于理解，下文将参考 FIL 数据结构的具体实现文件来描述 FIL 数据结构。然而，情况仍然是，该文件的行可以对应于能够由电子数据处理程序解译的逻辑子句。

II.2 FIL 数据结构的正式特征

该文件的每个 FIL 行优选地包括与其所关联的建模单元有关的代码，在适当的时候包括一个单独的字符来规定所描述的单元的类型，例如，“a”表示属性（attibute），“c”表示类（class），“h”表示继承（heritage），“r”表示“关系”(relationship)，等等。每

行还包括主体，主体包括与该单元有关的特征。这就产生了适合于电子数据处理应用程序使用该行的结构。相应代码的存在还便于用户对该单元的识别。

通过示例考虑以下短语：“猫吃老鼠”。该短语在“猫”与“老鼠”之间建立了关系。可以使用 FIL 表示法将该关系编码为以下行：

r: cat (1..1) to eat (1..n) mice

其中“r”规定所讨论的建模单元是关系，并且(1..1)和(1..n)是指分别对应于（一只）猫和（一些）老鼠的基数。

在下文中将对其他示例进行详细描述。

某些 FIL 行的主体优选地包括两个部分，第一部分包括建模单元内在的特征（例如属性的类型值），以及第二部分包括与建模单元的容器有关的特征。这样就将一个单元声明为与其容器直接联系。然而，该声明主要与所述建模单元有关，并且容器可以是独立声明的对象。因此，相应的声明（即子句的声明或文件行的声明）保持独立。此后，这就得到使得应用程序能够重构多个单元之间的层级的文件结构。

在下文中所详述的示例与特定的实施例有关，在该实施例中，出于可读性的原因，按照类型以例如以下指明的顺序将 FIL 行分类：封装、类、属性、操作、关联、继承。这使得了解单元及其关系更加容易。然而，应当牢记，可以对这些行进行声明并以任意顺序将其存储于存储器中。在行的第一附加内容中，将规定这些单元的代码缩减为一个单独的小写字母。如上例中所示，冒号“:”紧接在后，其后又接着一个空格。此外，逗号（如果有的话）始终后接一个空格。可以应用其他句法规则，这一点在这些示例中将变得明显。

如果合适，则可以将数据结构产生和/或电子数据处理工具集成到辅助程序中，或至少辅助程序可以包括该工具的功能。

该工具可以稳定数据结构，即，将相应的值指定给该数据。该工具还可以访问该结构，即，能够从该结构中提取值并且/或者能够操纵该数据结构（替换其值）。特别地，该工具解译文件的 FIL 行。

如有必要，可以配备用于纠正或解译句法错误和疏忽的例行程序。可以以任意顺序将这些行存储于存储器中。

II.3 示例

示例 1

将属性声明如下:

a: 属性名称，类名称

其中“属性名称”和“类名称”逻辑地与属性名称和该属性所属的类的名称相联系，逗号用作分隔符。

示例 2

a: 属性名称:类型，类名称

在本例中，用属性的类型来声明属性。优选地，在“:类型”之前或之后没有空格。标准类型是本领域中通常的类型，例如“String”（串）、“boolean”（布尔）、“int”或“integer”（整数）、“Date”（日期）。可以使用其他类型，包括表示法本身中描述的类的名称。例如:

a: 属性名称:类 1，类 2

示例 3

a: 属性=值，类名称

在此，为属性分配初始值。优选地，在等号之前或之后没有空格。可以注意到，该工具可以有利地解译不完整的句法（如同在上例中那样，其中缺少了属性类型）。例如，在考虑多种标准的情况下，可以将该工具设计为进行解译并设计为自动地找到适当的类型。这些标准可以基于属性名称本身，例如，“编号（number）”、“数量”、“米”（meter）等，然后该工具默认地提出“整数”类型。这些标准还可以基于与属性相关联的值。例如，如果该值是“真”或“假”，则所提出的类型为“布尔”；如果该值是数字的，则根

据该值是否在小数点后包含数字，所提出的类型为“整数”或“实数”；如果该值在双引号之间（从而为“...”），则所提出的类型为“串”（表示“字符串”）；等等。因此，在很多情况下，由于该工具能够确定类型，因此指出类型可能是多余的。

示例 4

a: 属性:类型=值，类名称

在本例中，句法是完整的。

示例 5

c: 类名称[，封装名称]

在此声明了类，以及[，类所属的封装]。在此，方括号表示对封装名称的可选引用。例如，如果没有提到封装并且先前没有遇到封装，则可以系统地将类存储于例如“p_system”封装之类的给定封装中，随后将该给定封装视为最高级别的封装。该封装可以例如默认地存在并且仅当类没有目的封装时才使用该封装。如有必要，如果没有指明封装名称，则可以使用电子数据处理工具将类存储于在 FIL 文件中遇到的最后一个封装“p:”中。注意，在此包括顺序的概念只是为了弥补用户部分造成的句法错误。类似地，通过调整该工具可以省略利用“c:”对类进行的明确声明，从而基于属性、操作、关联和继承而自动地识别和声明类。然而，该工具对作为输出而生成的 FIL 文件使用代码“c:”。另一方面，代码“c:”可以有利地明确用于利用定型（参见下文）来声明类或强制地使类属于一个封装。

示例 6

c: <<定型>>类名称[，封装名称]

在此，利用定型和[，类所属封装]对类进行了声明。如有必要，还可以将定型与封装、属性、操作和关联相关联。

示例 7

h: 类名称 1 “是一个” 类名称 2

在这个新的示例中，使用关键字声明了继承。对继承进行声明所必须的所用关键字是“是一个”。应当注意，代码“h:”可以省略。如果不存在代码“h:”，则该工具可以自动地添加该代码，该工具将 FIL 行解译为与继承单元有关，原因是存在相应的关键字。因此，可以简单地将该行输入为以下示例中这样：“正方形是一个图形单元”。

示例 8

o: 操作名称，类名称

在此声明了操作。只需要一个逗号，后接一个空格。

示例 9

p: 封装名称[，包围封装]

在此声明了封装，该封装属于具有名称“包围封装”的包围封装。对包围封装的指明是可选的。

示例 10

r: 类名称 1 (1...1) 关联名称 (1...n) 类名称 2

本例示出了规定基数的一种方式。使用了两个数字，关联的每一端具有一个数字并且在括号中。在适当时，指令以“(1”、“(0”、“(m”或“(n”开始并以“1)”或“n)”结束。之前或之后有空格。至于关联或类的名称，其可以是任意名称，例如包括一个或多个字。同样，在名称之前和之后有空格。此外，如果关联名称是“由……构成”或“包含”，则可以在随后使用的 UML 工具中自动地生成集合（aggregation）。此外，正如代码“h:”，代码“r:”不是必需的，该工具适合于这样解译关联，即如果需要，则通过识别基数的编码（codification）来解译关联。例如，可以将行“系统(1...1)由(1...n)

子系统构成”直接解译为描述关联“由……构成”（集合）。

示例 11

u: 使用场合名称，封装名称

对使用场合的声明还可以用于声明可能对应于一个或多个使用场合的系统的功能。

考虑以下示例:

c: 猫，示例

c: 老鼠，示例

r: 猫(1...1)要吃(1...n)老鼠

在 UML 表示法中，将以上子句表示如下。类“猫”和“老鼠”通常通过标签来符号化。相应的基数可以在附近出现。上述关系（r: 猫(1...1)要吃(1...n)老鼠）通过联系标签的箭头来符号化。

II.4 FIL 格式数据结构的稳定（valorization）

FIL 表示法的简单性使得能够将数据结构（或文件）创建为在可应用时直接从对以自然语言编写的文本的分析中反映该表示法。因此，所创建的结构可以用作例如创建 UML 对象或生成源程序的基础。

再次参考图 1，其中创建内部对象模型 60 的步骤 110 反映在对 MOI 的数据结构的稳定中。

“稳定”意指对对应于数据的值进行存储（例如在计算机的随机存取存储器中或在大容量存储介质上）。因此，可以在计算机上处理数据结构 60。

特别地，可以将建模单元表示（即由计算机转换）为数据结构 60 的相应子句。注意，可以考虑不同的算法变型。例如，如果建模单元是单独的（在分解步骤 100 中），则将该建模单元转换为相应的子句（步骤 110）。作为替代，在步骤 100 中对每个单元进行识别

后，可以将这些单元转换为子句。因此，分解步骤 100 和创建步骤 110 可以是至少部分伴随的。

III. 转换为外部元模型的建模单元（创建新模型或代码生成）

III.1 创建新模型，例如 XMI 模型

再次参考图 1，该方法可以在转换步骤 120、140 期间或之后包括 XMI 生成器从与 MOI 60 相关联的数据结构创建 XMI 模型 70 的步骤 120。XMI 生成器可以根据与例如在下文中描述的码生成器所采用的原理类似的原理来操作。在访问数据结构的内容之后，该生成器将该结构存储于其自身的内部存储器中，并随后根据其特有的转换规则对其内容进行转换，针对这些转换规则有多个可以设置的参数，并且这些转换规则依赖于将在两个元模型之间建立的匹配：例如，一方面是 FIL 数据结构本身（输入结构）的元模型，以及另一方面是 XMI、Java、C#或 C++数据结构（期望的输出结构）的元模型。因此，该生成工具仅应用于内部地存储了将给定元模型的建模单元与另一元模型的建模单元相匹配的转换规则的数据结构。

从这一方面讲，存在用于构造这种生成器的工具，诸如来自 Sodifrance 的 Model-In-Action（参见 www.mia-softwore.com）：利用 UML 描述两个元模型，并且随后以这种方式非常迅速地得到使用高级语言的转换规则以及转换器－语言生成器。

在此，从先前稳定的数据结构 60 生成模型 70。然后，可以在模型产生步骤 130 期间从 XMI 模型 70 产生图形的 UML 模型 75。从 XMI 模型产生图形的 UML 模型本身在本领域中是已知的（例如，诸如来自 Softeam 的 Objecteering®或来自 i-Logix 的 Rhapsody®之类的工具可以用于这一点）。

III.2 代码生成

本发明的方法还可以包括步骤 140，即在转换步骤 120、140 期

间或之后生成代码 80。以与上面参考 XMI 生成器所描述的方式类似的方式，在代码生成步骤期间，例如存储数据结构，并随后根据转换规则对其内容进行转换，针对这些规则有多个可以设置的参数，并且这些转换规则依赖于所需的两个元模型之间的匹配。

在该步骤结尾，生成 Java 类或 C#类。这些类可以有利地在综合服务开发环境（SDE）85 中使用。服务开发环境是知道例如如何直接读取 Java 或 C#源类的对象应用程序开发环境。从这一方面讲，步骤 150 是引入这些类 80 的简单步骤，这些类 80 直接由环境 85 以目标格式正确地生成。

应当注意，存在用于从“专有”内部模型生成代码的现有技术工具（例如来自 IBM 的 Rational Rose®，Objecteering®，或 Rhapsody®）。所讨论的内部对象模型通常是每个制造商特有的，这些内部对象模型不会被发布、不可修改并且不可输出（exportable）。

为实现代码生成，现有的代码生成器采用级别比将产生的代码的级别更高的描述。该描述通常依赖于诸如 UML 之类的建模并用于以给定的编程语言生成其代码。

实际上，辅助程序使得开发者能够选择目标语言（例如 C#或 Java）。然后，辅助程序进行采用适当模式向目标语言的转换。例如，Addison-Wesley 专业出版社 1995 出版的 Erich Gamma、Richard Helm、Ralph Johnson、John Vlissides 所著的“Design Patterns”（设计模式）第一版中描述了对产生转换模式来说有用的转换原则。

例如，在下文中给出由辅助程序从上述场景/使用场合生成的 C#源代码。

```
//------------------------------
// C# Class Name : MessagingBP
//
// Creation Date : 11-15-XXXX
// Generated by XXXXX
//------------------------------
 <%@ WebServce Language="c#"
Class="PoweredBy.BusinessProcess.Messaging.MessagingBP" %>

using System;
using System.Web.Services;
using PoweredBy.WebServiceProxy;
```

```
namespace PoweredBy.BusinessProcess.Messaging
{
        [WebService]
        public class MessagingBP:System.Web.Services.WebService
        {
                // Attributes
                private DirectoryService theDirectoryService;
                private MessagingService theMessagingService;
                private UserPreferenceService theUserPreferenceService;
                private String sender_email_address;
                private ContactInfo[] sender_contact_list;
                private String call_subject;
                private msgTypePref messaging_type;
                private int i;

                // Constructor
                public MessagingBP()
                {
                        theDirectoryService = new DirectoryService();
                        theMessagingService = new MessagingService();
                        theUserPreferenceService = new UserPreferenceService();
                }
                // Operations
                [WebMethod] [SoapRpcMethod]
                public void BroadcastMessage (String sender_name, String message)
                {
                        sender_contact_list =
theDirectoryService.GetMyContactList (sender_name);
                        for (i=0; i<=sender_contact_list.Length; i++)
                        {
                        messaging_type =
theUserPreferenceService.GetMessagePreference (sender_contact_list [i] .Name);
                        switch (messaging_type) {
                                case msgTypePref.Sms: {
        theMessagingService.SendSMS (sender_contact_list [i] .Mobile,message);
```

```
                                        break;}
                                (...)
                                } // end switch
                                } // end for
                            } // end of operation
                        } // end of class
                    } // end of namespace
```

一旦生成了例如“.asmx”文件形式的源代码，就可以立即通过连接到辅助程序的测试环境来对其进行测试。

III.2 部署

现在，开发工程师可以通过选择所需的部署类型（例如 Microsoft IIS、Apache/Axis 或 BPEL）来部署最终的服务。

然后，采用适当的模式，将新服务的最终源代码移入适当的目录中。然后，产生并部署相应的代理。然后就可以在因特网或内网上根据所选择的部署特征获得该新服务。

因此，本发明用于将由用户编写的不严密的原始文本转换为可编译的、可执行的、可测试的且准备好部署的软件。

应当牢记采用本发明方法的辅助程序的全部方法，本发明的方法集合了多个有益的基本方法并改善了 Web 服务的产生。根据本发明，可以更快地、以更低的成本并且以更好的质量产生服务。特别地：

- 根据“交互处理”方法，可以将新的 Web 服务视为对多个现有基本服务的配合集合；

- 根据多参与者方法，分析者、开发者等不同的参与者可以在不同的级别上使用该辅助程序；

- 根据多形式方法，辅助程序可以读取、转换和反转半正式的或模糊的文本；

- 根据其不可知（agnostic）方法，独立于最终的实现语言来表达目标事务处理的使用场合和场景；以及

- 根据“建模单元”和 MDA 方法，结合地且屏蔽地，构造了对

用户隐藏的一致的 MOI，用户只需要操纵高级描述。

然而，本发明不限于上述实施例，并且本发明可将其自身推广为多种其他变型，这些变型对本领域的技术人员来说是相当明显的。

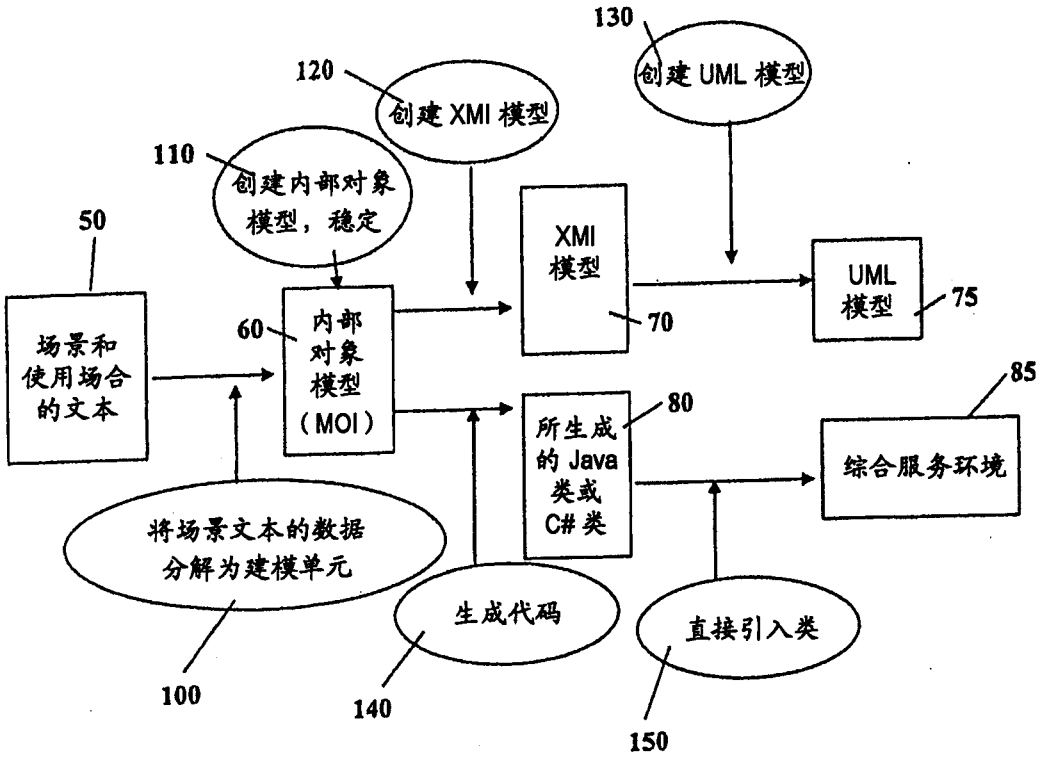

130
创建 UML 模型
120
创建 XMI 模型
110
创建内部对象模型，稳定
50
场景和使用场合的文本
60
内部对象模型（MOI）
XMI 模型
70
UML 模型
75
85
所生成的 Java 类或 C# 类
80
综合服务环境
将场景文本的数据分解为建模单元
100
生成代码
140
直接引入类
150

图 1