[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 9/44 (2006.01)

[12] 发明专利说明书

专利号 ZL 200610058898.3

[45] 授权公告日 2009 年 9 月 16 日

[11] 授权公告号 CN 100541429C

[22] 申请日 2006.3.7
[21] 申请号 200610058898.3
[30] 优先权
[32] 2005.3.7 [33] DE [31] 102005010405.3
[73] 专利权人 西门子公司
地址 德国慕尼黑
[72] 发明人 德特利夫·贝克尔
卡尔海因兹·多恩 克里斯琴·沙夫
弗拉迪斯拉夫·尤基斯
汉斯-马丁·范斯托克豪森
[56] 参考文献
CN1308281A 2001.8.15
US6664978B1 2003.12.16
US2003/0023953A1 2003.1.30

Oracle Application Development Framework Overview. Shay Shmeltzer. ORACLE. 2004
审查员 钟文芳
[74] 专利代理机构 北京市柳沈律师事务所
代理人 邵亚丽 李晓舒

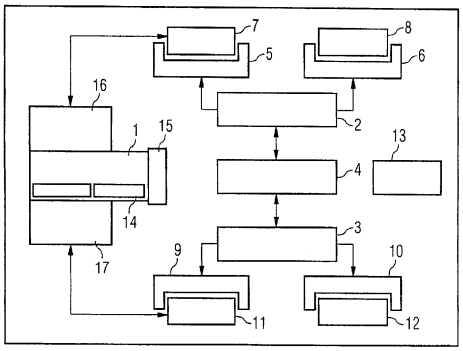
权利要求书 3 页 说明书 19 页 附图 3 页

[54] 发明名称

采用用户引导进行自动应用开发的系统和方法

[57] 摘要

本发明涉及一种用于利用用户引导进行应用开发的系统，包括至少一个具有多种表现形式(7，8)的可视组件(5－8)，分别用于在特定时刻察看数据和由用户进行输入，至少一个用于存储和调用该数据的模型组件(9－12)，用于将用户输入和/或对数据的查询传递到模型组件(9－12)并将被查询的数据传递到表现形式(7，8)的控制组件(1)，其中控制组件还用于在不同的表现形式(7，8)之间进行切换，还具有用于提供不同表现形式之间的切换的可配置顺序的过程运行组件(4)。



1. 一种用于利用用户引导进行应用开发的系统，包括

- 至少一个具有多种表现形式（7，8）的可视组件（5-8），分别用于由用户在特定时刻察看数据和由用户进行输入，

- 至少一个用于存储、处理和/或调用数据的模型组件（9-12），

- 用于将用户输入和/或对数据的查询传递到模型组件（9-12）并将被查询/被处理的数据传递到表现形式（7，8）的控制组件（1），其中该控制组件（1）还用于在不同的表现形式（7，8）之间进行切换，以及

其中还具有用于提供不同表现形式之间的切换的至少一种可配置顺序的过程运行组件（4），

其特征在于，

- 具有状态管理器（13），用于在系统挂起时存储该系统其它部分的状态并在该系统恢复时重新建立这些状态。

2. 根据权利要求 1 所述的系统，其特征在于，所述模型组件（9-12）具有多个业务形式（11，12），这些业务形式分别与至少一个表现形式（7，8）逻辑关联，其中不同表现形式（7，8）之间的切换与不同业务形式（11，12）之间的切换相耦合。

3. 根据权利要求 2 所述的系统，其特征在于，所述表现形式（7，8）以及业务形式（11，12）都分层地设置，从而使当一个切换顺序在一个层次内部进行时该切换顺序已在比该层次更低层的一个层次中运行过。

4.根据权利要求 1 所述的系统，其特征在于，所述表现形式（7，8）以及当存在业务形式（11，12）时的业务形式（11，12）都分层地设置，从而使当一个切换顺序在一个层次内部进行时该切换顺序已在比该层次更低层的一个层次中运行过。

5. 根据权利要求 3 所述的系统，其特征在于，所述过程运行组件（4）基于用户在表现形式（7，8）中输入的预定事件指示控制组件（1），利用确定的顺序在两个表现形式（7，8）之间切换以及在两个业务形式（11，12）之间切换。

6. 根据权利要求 4 所述的系统，其特征在于，所述过程运行组件（4）基于用户在表现形式（7，8）中输入的预定事件指示控制组件（1），利用

确定的顺序在两个表现形式（7，8）之间切换以及在两个业务形式（11，12）之间切换。

7. 根据权利要求 5 或 6 所述的系统，其特征在于，所述表现形式（7，8）的切换顺序通过用于过程运行组件（4）的至少一个配置文件来确定。

8. 根据权利要求 7 所述的系统，其特征在于，所述切换通过至少一个可预先配置的事件触发。

9. 根据权利要求 1 所述的系统，其特征在于，还包括可视管理器（2），用于为至少一个可视组件（5-8）产生通用的前端程序组件（5，6）。

10. 根据权利要求 1 所述的系统，其特征在于，还包括模型管理器（3），用于为至少一个模型组件（9-12）产生通用的后端程序组件（9，10）。

11. 根据权利要求 1 所述的系统，其特征在于，所述控制组件（1）包括与所述模型组件（9-12）和可视组件（5-8）相互传递事件的装置。

12. 根据权利要求 1 所述的系统，其特征在于，所述控制组件（1）包含或处理用于产生至少一个可视组件（5-8）的表现形式（7，8）的程序代码（16）。

13. 根据权利要求 2 所述的系统，其特征在于，所述控制组件（1）包含或处理用于产生模型组件（9-12）的业务形式（11，12）的程序代码（17）。

14. 根据权利要求 12 或 13 所述的系统，其特征在于，所述程序代码基于脚本。

15. 根据权利要求 1 所述的系统，其特征在于，所述系统包括至少一个用于在操作系统层面上本地使用的可视组件（5，7），和至少一个用于通过网络协议在网络中使用的可视组件（6，8）。

16. 根据权利要求 1 所述的系统，其特征在于，所述系统还包括用于在切换表现形式时临时存储数据的装置。

17. 根据权利要求 16 所述的系统，其特征在于，所述数据的临时存储可以通过过程运行组件（4）进行，并且在控制组件（4）的存储区中进行。

18. 根据权利要求 12 所述的系统，其特征在于，用于产生和配置系统组件的程序代码是以脚本语言编写的代码。

19. 一种采用自动用户引导进行应用开发的方法，包括以下步骤：

- 提供具有多种表现形式（7，8）的可视组件（5-8），分别用于由用户在特定时刻察看数据和由用户进行输入，

- 提供至少一个用于存储、处理和/或调用数据的模型组件（9-12），

- 通过控制组件（1）将用户输入和/或对数据的查询传递到模型组件（9-12）并将被查询/被处理的数据传递到表现形式（5-8），

- 在确定了预先给定的用户输入之后，通过控制组件（1）从一个表现形式（7，8）切换到另一预定的表现形式，

- 利用过程运行组件（4）确定不同表现形式（7，8）之间的切换顺序，

- 通过状态管理器组件（13）在系统挂起时存储该系统其它部分的状态以及在系统恢复时重新建立这些状态。

20. 根据权利要求19所述的方法，其特征在于，所述模型组件（9-12）具有多个业务形式（11，12），这些业务形式分别与至少一个表现形式（7，8）逻辑关联，并且还具有步骤：

- 将不同表现形式（7，8）之间的切换与不同业务形式（11，12）之间的切换相耦合。

采用用户引导进行自动应用开发的系统和方法

技术领域

本发明涉及一种采用用户引导进行自动应用开发的系统和方法。

背景技术

当今的软件体系结构一般都提供了根据特定的规则构造前端逻辑和后端逻辑来实现软件的均匀性并构建软件。由此可以实现产生软件应用程序的应用软件的开发。在此前端逻辑理解为已准备好的程序逻辑，而后端逻辑以多个有序的处理步骤为目标，如将信息存储在数据库以及重新获取该信息。常见的应用程序具有整体特性，这在该应用长时间使用时会导致非常难以将针对该应用的变化的或者说新的要求移植到该应用中。这首先是因为当今的软件体系结构好得没有为应用的内部问题提供支持。

目前不存在子应用的概念。此外当今的软件体系结构没有为应用内部的数据流提供支持。使用者更多地是在处理查询、新信息的输入、从数据库获得信息等等时、尤其是在复杂的贯穿多个单独的逻辑阶段的过程中依靠自己。

此外，当今软件体系结构的特征是前端和后端中的编程模型原则上是不同的，这甚至会形成专门的软件开发团队，即前端开发者和后端开发者。但这种专门化会对整个开发过程、尤其是对成本产生负面影响。

此外当今的软件体系结构没有为应用内部的数据流提供支持，这会部分导致应用内不一目了然和不受控制的数据交换并且容易出错。当使用者在工作时没有受到特定的软件系统的各个工作步骤的布置支持，而是被告知要自己产生流程计划时，也可能会出现错误。

此外当前的根据现有技术的措施也带来安全性风险，因为在（非期望的）系统中断时无法保证开发状态。因此 Oracle 的白皮书“Oracle Application Development Framework Overview”，2004 年 4 月存档，同样展示了在建立软件体系结构时的框架。但该框架的缺点是不包含尤其是使得可以存储系统存在的状态、从而在出现干扰时可以保证无差错地重新运行的安全逻辑。

发明内容

本发明要解决的技术问题是提供一种软件系统和相应的方法，可以改善目前的过程并在开发时向应用程序开发工程师提供结构化的引导，同时提高系统的安全性。

该技术问题是通过一种用于在应用开发时自动产生用户引导的系统来解决的。

经典的系统由用于显示数据并以诸如表格、清单等不同形式显示该数据的可视组件组成。此外经典系统还包括作为后端的模型组件，其用于存储数据并具有关于数据存储类型的知识。上述系统中的第三部分是控制组件或控制器，其逻辑地设置在可视组件和模型组件之间。控制组件处理用户交互行为（键盘、鼠标事件等），将该用户交互行为转换为对模型组件的数据要求，并将其发送到模型组件。如果模型组件返回数据，则该数据由控制组件传送到可视组件并在该可视组件上以用户期望的方式进行显示。这个过程的优点在于，数据存储的类型和数据显示的类型都能被改变，而无需重建整个软件系统。控制组件的作用相当于“可视组件”和“模型组件”之间的中介。因此可视组件不涉及模型组件内的变化，并且模型组件不受可视组件中变化的影响。

对应地，本发明的用于在应用开发时自动进行用户引导的系统具有以下组件：

- 至少一个具有多种表现形式的可视组件（前端），分别用于在特定时刻察看数据和由用户进行输入，

- 至少一个用于存储和调用数据的模型组件（后端），

- 用于将用户输入和/或对数据的查询传递到模型组件并将被查询的数据传递到可视组件的控制组件，其中控制组件还用于在不同的表现形式之间进行切换，

- 用于确定不同表现形式之间的切换顺序的过程运行组件，

- 状态管理器，用于在系统挂起时存储系统的其它部分的状态以及在系统恢复时重新建立这些状态。

在优选实施方式中，所述过程运行组件用于提供可配置的运行过程。在本发明中，可配置的意思是针对每个开发过程专门匹配该运行过程，并且可由使用者进行设置。

所述过程运行组件用于提供可视组件之间可配置的切换顺序。此外还用于在应用开发时提供可配置的处理步骤顺序。

新型的体系结构引入了一种称为过程运行组件或“过程”的新量纲。该量纲针对上述与应用内的数据流在该应用及其开发所基于的软件体系结构没有为排列和导向数据流提供支持时可能为无序相关联的问题。利用过程运行组件可以达到如下目标：进行工作流自动化，从而使软件用户在其决定时受到数据处理步骤的可配置顺序的支持。这带来了能产生更少错误的优点。

与本发明的可视组件具有多种表现形式雷同，模型组件也构造为具有数据存储的多种业务形式，这些业务形式分别与至少一个表现形式逻辑关联，其中不同表现形式之间的切换与不同业务形式之间的切换相耦合。

在此业务形式理解为一种特定的编程技术方式，来自数据库或类似数据储备源的数据以该方式被存储、调用和处理。通过这种方式可以在针对每个表现形式或一组表现形式具有匹配的业务形式的情况下优化数据流。

在另一个优选实施方式中，表现形式以及必要时（如果存在）业务形式都分层地设置，从而切换顺序在一个层次内部进行时该切换顺序已在下一个更低层的层次中运行过。实际中每个表现形式和每个业务形式例如还可以由相继被激活、调用和处理的任务组成。

通过这种方式可以对用户引导进行结构化的控制，直至向下达到开发者被迫要输入形式表格的一个特定字段或命令行的特定格式化输入的层面。优选的，过程运行组件基于预先给定的事件在用户输入时在一个视图中指示控制组件利用所确定的顺序在两个表现形式以及必要时两个业务形式之间进行切换。在此分别对有效的用户输入进行分析，在对用户输入的特定要求进行编辑之后，过程运行组件可以整理特定条件，以便为用户输入更换表现形式。可以理解，还可以切换其它“触发器”，即切换到从显示表现形式以来的某个特定时刻的流程或清楚阐明该表现形式的明白的用户动作，使得该流程已完全处理了当前表现形式（例如按下返回键）。

优选的，通过针对过程运行组件的配置文件来确定表现形式的顺序，该配置文件被读出并确定哪个表现形式应当被切换为下一个。

两个表现形式之间的切换通过以下方式进行：当用户完成子数据流的最后一个任务之后，跳转到主数据流的第一个步骤。在浏览图表中配置主数据流的步骤。也就是说通过浏览图表确定可视组件被切换为哪一种顺序。

可视组件又可以分为多个子组件，以实现结构化的编程。这样就可以将可视组件分为通用的“前端”组件和实际的表现形式，其中前一个的任务是提供作为基础的功能来显示数据，后一个要定义表现这些数据的具体结构。

对应的，根据本发明的系统还包括可视管理器来为至少一个可视组件产生通用的前端程序组件。此外还可以包括模型管理器来为至少一个模型组件产生通用后端程序组件。通过相应的管理器产生前段程序组件和后端程序组件同样也可以由过程运行组件控制或至少由它来启动，当采用自己的管理器来产生通用后端和前端组件时，可以通过控制组件来产生实际的表现形式和业务形式。相应的，控制组件可以包含或处理用于产生可视组件的表现形式的程序代码。

控制组件还可以包含或处理用于产生可视组件的模型的程序代码。

该程序代码例如可以是配置文件，尤其是基于 XML 的配置文件。

此外在所述系统中还具有状态管理器，用于在系统中断（或挂起）时存储该系统其它部分的状态并在该系统恢复运行时重新建立这些状态。通过这种方式可以中断本发明的系统或者说转换到停止状态并重新恢复到同一状态下，而不必完全重新启动该处理过程。

本发明的系统既可以在计算机上本地实施，也可以在网络中实施，其中至少可以有通过网络相互连接的模型组件和可视组件运行在不同的计算机上。相应的，优选系统包括至少一个用于在操作系统层面、即在各个计算机上本地使用的可视组件，和至少一个用于通过网络协议在网络中使用的可视组件。由于在本地显示器上例如通过 X-Windows、PDF、Postscript 或 GDI 提供的与通过网络例如借助 HTML、XML 或专门协议提供的显示相比完全不同的处理方式，还需要专门的可视组件（要注意，X-Windows 和 Display-Postscript 变形既可以本地使用也可以通过网络使用，因此在此不是一定需要不同的可视组件）。原则上还提供了不同的可能性，即不是为本发明系统的单个组件而是为这些组件之间的交互及其运行来实施必要的程序代码。

本发明的系统还可以包括用于在切换表现形式时临时存储数据的装置，以便能将以一种表现形式获得的数据继续用于另一种表现形式。数据的临时存储可以通过过程运行组件、并且在控制组件的例如通过“端口”进入的存储区中进行。

在优选实施方式中，用于产生和配置所述系统的组件的程序代码是以脚本语言编写的代码。通过这种方式即使在建立所述系统之后也可以随时通过简单

的修改脚本代码来更改系统，因此系统的维护也很简单。

上述本发明方法的实施方式也可以构成为计算机程序产品，其中该计算机用于执行上述本发明的方法，并通过处理器执行其程序代码。

可替换的解决技术问题的方案在于一种存储介质，其用于存储上述用计算机实施的方法，并可由计算机读取。

此外还可以按照一种可销售整体来实施上述方法的各个组件，并按照另一种可销售整体来实施其余组件。因此本发明技术问题的其它解决方案在于一种产品，包括:

分布式系统的至少一个可视组件、模型组件、控制组件和过程运行组件，该系统包括根据至少一个上述方法方面来执行由该产品完成的方法的步骤的装置，其中至少另一个产品用于执行该方法的其余步骤，并通过该至少两个产品的协作来执行该方法的所有步骤。

附图说明

在下面的详细附图描述中借助附图非限制性地给出了实施例及其特征和其它优点。其中:

图 1 示出本发明的系统的体系结构;

图 2 示出形成本发明系统的各个组件;

图 3 示出各个表现形式的分层流程计划的例子。

具体实施方式

当向已知的软件模式 MVC（模型•可视•控制器•）添加过程运行组件时，可以在用户与软件系统的交互中提供该软件系统应当向应用开发者或用户作为序列提供的可执行步骤的可配置模式。通过以特定顺序专门化这些步骤，可视组件恰好以该顺序通过控制组件要求模型组件的数据，从而软件利用这些步骤构成的引导来支持开发者做出决定。

图 1 示出本发明的用于引导用户的系统的体系结构的概貌。可视组件通过控制组件与模型组件通信。在此本发明的过程是负责该交互的设置或者说序列，并确定该交互应当以何种顺序进行。该过程由所谓的浏览图、即配置文件执行。浏览图具有关于在处理步骤进行之后下一步进行什么步骤的信息。在分层设置中，除了浏览图之外还有子浏览图。例如用于特定的“任务”序列。下面详细

解释本发明系统的组件结构。在开发体系结构时就非常重视应用组成部分的对称性。可视组件由所谓的通用前端(GenFE)表示,模型组件由通用后端(GenBE)表示。两种组件都要容纳容器的通用实施。GenFE 是应用的可视前端部分的通用容器,GenBE 是应用的后端部分的通用容器。这些前段和后端组件之间的通信通过控制组件控制。

从图 1 可以看出,GenFE 和 GenBE 分别包含所谓的形式,也就是用于表现数据和输入该数据或进行格式化处理的显示。在前端中具有由 GenFE 容纳的表现形式,在后端中具有由 GenBE 容纳的业务形式。

两种形式都基于将软件层的组件(前端或后端组件)及其交互可能在配置文件中进行说明并可以由所谓的形式产生的思想。真正的交互是本地化的,并在一个或多个文件中优选以脚本语言进行编程,从而该软件层的用户可以灵活地适应由已配置的组件组成的形式内容以及组件交互,而不必重新建立整个软件层。图 2 示出在优选实施方式中两种形式的脚本代码驻留在控制组件中,这称为脚本代码支持(Script Code Behind)。这在显示程序组件必须以不同技术、例如本地或通过网络(例如用 WinForm 或 WebForm)来显示数据时有利。如果表现形式的脚本在控制组件中运行,则当前端组件的显示类型改变时不一定改变组件交互。该“形式”已在较早时刻定义过。

当今的软件体系结构的特征在于,前端和后端中的程序模型原则上是不同的。但前端开发和后端开发的不对称可以通过在这里介绍的本发明的体系结构中采用形式概念来显著减小。

图 2 以示意性的结构示出本发明系统的细化模型。本发明体系结构的 4 个核心元件在图 2 中清楚示出。4 个激活的主部件是控制器 1、可视管理器 2、模型管理器 3 以及本发明的过程,也就是过程运行程序组件 4。可视管理器 2 产生通用前端组件 5、6,它们又分别借助脚本代码产生描述和处理应用的前端逻辑的表现形式 7、8。在本发明的该实施方式中值得注意的特征是可视管理器 2 可配置,且在图 4 示出的例子中针对不同的技术 Webform 和 Winform 存在两个可视管理器 2。通过从外面进行配置来选择、产生和应用特定的可视管理器 2。WinForm 前端组件 5、7 可以在 Windows 下用于桌面,WebForm 前端 6、8 用于本发明系统的网络应用。在网络应用中,容器或者说整体容器在 Web 服务器中运行。通常在 Windows 下采用 Microsoft Internet Information Server (IIS)。

在本发明系统的后端中具有能产生通用后端组件 9、10 的模型管理器 3。

通用后端组件9、10又借助脚本代码17产生描述并处理应用的应用逻辑的业务形式11、12。

在图2中可以在右侧看见状态管理器13，其中可视管理器2和模型管理器3取出它们的状态。对状态管理器13的访问一直传播到形式7、8、11、12中，由此位于相应形式的组件可以存储其状态并又读出。利用该特征，本发明的整个系统可以在系统暂停之前通过状态管理器13存储系统的状态，并在系统重新恢复之后马上读取该状态。因此可以中断处理并重新开始，而无需完全重新启动处理过程。

图2的左边是控制前端和后端之间通信的控制组件1。前端和后端不必在操作系统的同一个过程中运行。同样可以启动两个分别只具有前端或后端的容器，驻留在容器1中的通信层负责使两个通信伙伴也通过网络找到对方。此外，控制器1还容纳了事件系统14，其负责使应用的前端和后端能产生事件。

在图2的中间，用数据流机可以看见本发明的系统体系结构的第4个基本组件，即由浏览图代表的过程运行程序组件4（过程）。浏览图包含关于用户应当以何种顺序执行特定的工作步骤的信息，以开发特定的应用或执行特定的工作。下面的程序清单示出浏览图的配置。其中有3个视图相互连接：View_2D, View_3D和View_Filming。因此如果使用者已经结束了与View_2D的交互，则连接到 View_3D。此外，如果使用者结束了与 View_3D 的交互，则连接到View_Filming等。

```
<navigationGraph name="Datenflussl" startView="View_2D" state="State"
statePersist="MemoryStatePersistence" iViewManager
="“UserControlViewManager” >

    <node view= ‘View2D’ screen= ‘’ subWorkFlow= ‘’>
      <navigateTo navigateValue=”previous” view=’ View_Filming’ />
      <navigateTo navigateValue=”next” view=’ View_3D’ />
    </node>

    <node    view=’View_3D’ screen=’screen1’ popup=’false’ subWorkFlow= ‘’>
      <navigateTo navigateValue=”previous” view=’ View_2D’ />
      <navigateTo navigateValue=”next” view=’ View_Filming’ />
```

</node>

<node view= 'View_Filming' screen='screen2' subworkFlow= 'SubWorkFlow.xml'>

<navigateTo navigateValue="previous" view=' View_3D' />

<navigateTo navigateValue="next" view= 'View_2D ' />

</node>

</navigationGraph>

所属的模型 11、12 在此由可视组件或者说表现形式 7、8 驱动，也就是在从一个可视组件切换到另一个可视组件时所属的模型 11、12 也一起自动被切换。数据流机可以在切换可视组件时还在该可视组件之间传输数据。为此图 2 中在控制器 1 上具有端口 15。可视组件 7、8 可以使数据流机将其完成的数据存储到一个端口中，下一个可视组件可以使数据流机从该端口中取出数据以进行处理。

分层运行控制和所属后台数据流的概念同样属于本发明系统的过程方面。外部的数据流通过先前示出的浏览图描述，并确定各个所谓的活动（可视组件+模型组件）怎样相互连接。"内部"的数据流或者说下一个低级的层次通过在下列程序清单中引用的数据流"Subdatenfluss1"描述，并确定活动中的数据流如何表现。

<?xml version="1.0" encoding="UTF-8" ?>

- <document xmlns:xi="http://www.w3.org/2003/XInclude">

- <SUBWORKFLOW>

- <TASKS>

- <TASK name="taskl">

- <!--

<xi :include href= ".\Task\taskl.xml" xpointer="xpointer(//Task)"/>

-->

<xi: include href= ".. \. \Task\taskl.xml" xpointer= "xpointer(//Model)" />

<xi: include href= "..\ .\Task\taskl.xml" xpointer= "xpointer(//View)"/>

</TASK>

- <TASK name= "task2">

-<!--

<xi: include href= “\XML-Files\Task\task2 .xml” xpointer= “xpointer(//Task)

”/>

-->

<xi: include href= “..\.\Task\task2.xml” xpointer= “xpointer(//Model)”/>

<xi: include href= “..\.\Task\task2.xml” xpointer= “xpointer(//View)”/>

</TASK>

- <TASK name= “task3”>

<xi: include href= “..\.\Task\task3.xml” xpointer= “xpointer(//Task)”/>

</TASK>

</ TASKS>

- <NAVIGATIONGRAPH name= “Subdatenflussl” startTask= “taskl”>

<xi: include href= “..\.\TaskNavigationGraph\TaskNavigationGraph.xml” xpointer= “xpointer(//NODE)” />

</NAVIGATIONGRAPH>

</SUBWORKFLOW>

</document>

图 3 以不涉及上述具体例子的示意图的形式示出系统的整个数据流。在此数据流 1（DF1）同等地与子数据流 1、2、3（SDF1, SDF2, SDF3）连接，这些子数据流的输出数据又引向数据流 1 并在经过继续处理之后到达数据流 2（DF2），子数据流 4、5、6（SDF4, SDF5, SDF6）中的数据同样也到达数据流 2。

上述程序清单展示了每个任务都由一个模型组件和一个可视组件构成。相应的配置如所述地对业务形式和表现形式来说是完全标准的配置。用于连接各任务的浏览图显示在下面给出的程序清单中：

<?xml version= “1.0” encoding= “UTF-B” ?>

- <NAVIGATIONGRAPH>

- <NODE task=”taskl”>

<navigateTo navigateValue=”PREVIOUS” task=”task3” />

<navigateTo navigateValue= “NEXT” task=” task2” />

</NODE>

- <NODE task="task2">
<navigateTo navigateValue="PREVIOUS" task="taskl" />
<navigateTo navigateValue= "NEXT" task=" task3" />
</NODE>
- <NODE task="task3">
<navigateTo navigateValue="PREVIOUS" task="task2" />
<navigateTo navigateValue= "NEXT" task=" task1" />
</NODE>
</NAVIGATIONGRAPH>

本发明系统的不同现有组件必须被启动并相互连接。这也是通过在下面的程序清单中显示出来的脚本逻辑地进行的。

<?xml version="1.0" encoding="UTF-8" ?>
- <objectTypes>
<iViewManager name= "UserControlViewManager"
type="Siemens.ApplicationBlocks.UIProcess.UserControlViewManager,
Siemens.ApplicationBlocks.UIProcess, Version=1.0.1.0,Culture=neutral,
PublicKeyToken=null" />
<iModelManager name= "MyModelManager"
type= "BusinessProcess.MyModelManager, BusinessProcess, Version=1.0.1.0,
Culture=neutral , PublicKeyToken=null" />
<state name="State"
type="Siemens.ApplicationBlocks.UIProcess.State, Siemens.
ApplicationBlocks.UIProcess, Version=1.0.1.0,Culture=neutral,
PublicKeyToken=null"/>
<controller name= "ActivityWorkflowController"
type="UIPUtil.ActivityWorkflowController, UIPUti1, Version=1.
0.1.0 ,Culture=neutral , PublicKeyToken=null" />
<statePersistenceProvider name="MemoryStatePersistence"
type="Siemens.ApplicationBlocks.UIProcess.MemoryStatePersistence,
Siemens.ApplicationBlocks.UIProcess, Version=l.0.1.0,Culture=neutral,
PublicKeyToken=null" />

</objectTypes>

上面的程序清单展示了被引用的对象类型的配置，在后面进行的配置中对该对象类型进行了引用。在此定义了可视管理器 2、模型管理器 3、状态管理器 13 和控制器 1。

下面的程序清单展示了反映本发明的整个系统的所谓的活动的配置。在此定义了一个模型组件、一个可视组件和一个控制器。控制器通过 Activity Work Flow Controller 被引用。实际的定义已经在下面和上面的程序清单中配置对象类型时进行了。

<?xml version="1.0" encoding="UTF-8" ?>

-<Activity>

<Model name="MyModel_l" type="BusinessProcess.MyModel,

BusinessProcess,

Version=1.0.1.0, Culture=neutral, PublicKeyToken=null"

iModelManager= "MyModelManager"

config-File= "\AT\dotNet\Core\BusinessForm\test.u\config\BizFormConfig.xml"

CMDPATTERN= "cmdPtnl" EVENTPATTERN= "eventPtnl" />

<View name= "Activityl" type= " syngo . Common .Activity .Activity, Activity,

Version=l.0.1.0, Culture=neutral, PublicKeyToken="null" configFile= ".

/PresentationContent/UI_Activitylxml"

CMDPATTERN= "cmdPtnl" EVENTPATTERN= "eventPtnl" />

<Controller name= "ActivityWorkflowController" />

</Activity>

下面给出的程序清单说明了应用的具有数据和子数据流的完整配置。

<?xml version="l.0" encoding="UTF-8"?>

- <document xmlns:xi="http://www.w3.org/2003/XInclude">

- <workflow>

- <!--

- <objectTypes>

- ->

- <objectTypes xml:base="ObjectTypes/ObjectTypes.xml">

<iViewManager name= “UserControlViewManager” type=”Siemens.ApplicationBlocks.UIProcess.UserControlViewManager, Siemens.ApplicationBlocks.UIProcess, Version=1.0.1.0, Culture=neutral, PublicKeyToken=null” />
<iModelManager name= “MyModelManager” type=”BusinessProcess.MyModelManager, BusinessProcess, Version=l.0.1.0, Culture=neutral, PublicKeyToken=null” />
<state name=”State” type=”Siemens.ApplicationBlocks.UIProcess.State, Siemens.ApplicationBlocks.UIProcess, Version=l.0.l.0,Culture=neutral, PublicKeyToken=null” />
<controller name= “ActivityWorkflowController” type=”UIPUtil.ActivityWorkflowController, UIPUtil, Version=l.0.1.0, Culture=neutral, PublicKeyToken=null” />
<statePersistenceProvider name=”MemoryStatePersistence” type=”Siemens.ApplicationBlocks.UIProcess.MemoryStatePersistence, Siemens.ApplicationBlocks.UIProcess, Version=l.0.1.0, Culture=neutral, PublicKeyToken=null” />
</objectTypes>
- <!--
- </objectTypes>
- ->
- <Activities>
- <Activity name=”Activityl”>
<Model name=”MyModel_l” type=”BusinessProcess.MyModel, BusinessProcess, Version=l.0.1.0, Culture=neutral, PublicKeyToken=null” iModelManager=“MyModelManager”config-File= “\AT\dotNet\Core\BusinessForm\test.u\config\BiZFormConfig.xml” CMDPATTERN= “cmdPtnl” EVENTPATTERN= “eventPtnl” xml :base= “Activity/Activityl .xml” />
<View name=”Activityl” type=”syngo.Common.Activity.Activity, Activity,

Version=l.0.1.0, Culture=neutral, PublicKeyToken=null" config-File= ".
/PresentationContent/UI_Activitylxml" CMDPATTERN="cmdPtnl"
EVENTPATTERN= "eventPtnl" xml: base= "Activity/Activityl . xml"/>
<Controller name="ActivityworkflowController"
xml: base="Activity/Activityl . xml" />
- <SUBWORKFLOW xml: base= "Subworkflow/Subworkflowl . xml">
- <TASKS>
- <TASK name="taskl">
-<!--
- <xi: include href=" . \Task\taskl .xml" xpointer="xpointer(//Task) "/>
- ->
<Model name="taskl" ID="groupl" PATTERNCMD="patl" PATTERNEVT=
"evtl" FILE=".\BusinessContent\stepl_Model.xml"
xml :base="Task/taskl .xml" />
<View name="taskl" ID= "groupl" PATTERNCMD= "patl" PATTERNEVT=
"evtl" FILE=" . \PresentationContent\subworkflow_stepl .xml"
xml : base=" Task/taskl . xml" />
</TASK>
- <TASK name="task2">
- <!--
- <xi: include href="\XML-Files\Task\task2 .xml"
xpointer= "xpointer (/ /Task)"/>
- ->
<Model name="task2" ID= "group2" PATTERNCMD= "pat2"
PATTERNEVT= "evt2" FILE= "\BusinessContent\step2_Model.xml"
xml :base= "Task/task2 .xml" />
<View name=" task2" ID= "group2" PATTERNCMD= "pat2" PATTERNEVT=
"evt2" FILE= ". \PresentationContent\subworkflow_step2 .xml"
xml :base="Task/task2 .xml" />
</TASK>
- <TASK name= "task3">

- <Task xml :base= “Task/task3 .xml”>
<Model name= “task3” ID= “group3” PATTERNCMD= “pat3”
PATTERNEVT=”evt3” FILE= “\BusinessContent\step3Model.xml” />
<View name=”task3” ID= “group3” PATTERNCMD=” pat3” PATTERNEVT=”
evt3” FILE= “ . \PresentationContent\subworkflow_step3 .xml” />
</Task>
</TASK>
</TASKS>
<NAVIGATIONGRAPH name= “navgraphl” startTask= “taskl”>
- <!--
<xi: include href= “. \TaskNavigationGraph\TaskNavigationGraph.xml”
xpointer= “xpointer(//NAVIGATIONGRAPH)”/>
- -->
- <NODE task=”taskl”
xml : base= “ TaskNavigationGraph/TaskNavigationGraph. xml”>
<navigateTo navigateValue=”PREVIOUS” task= “task3” />
<navigateTo navigateValuer=“NEXT” task=” task2” />
</NODE>
- <NODE task=”task2”
xml : base=” TaskNavigationGraph/TaskNavigationGraph . xml”>
<navigateTo navigateValue=”PREVIOUS” task=”taskl” />
<navigateTo navigateValue=”NEXT” task=”task3” />
</NODE>
- <NODE task=”task3”
xml : base=”TaskNavigationGraph/TaskNavigationGraph . xml”>
<navigateTo navigateValue= “PREVIOUS” task= “task2” />
<navigateTo navigateValue= “NEXT” task= “task1” />
</NODE>
</NAVIGATIONGRAPH>
</SUBWORKFLOW>
</Activity>

- <Activity name= “Activity2”>

- <Activity xml :base = “Activity/Activity2 .xml”>

<Model name=”MyModel_2” type=”BusinessProcess.MyModel, Busi-nessProcess, Version=l.0.1.0, Culture=neutral, PublicKeyToken=null” iModelManager=“MyModelManager” config-File = “\AT\dotNet\Core\BusinessForm\test.u\config\BizFormConfig.xml” CMDPATTERN= “cmdPtn2” EVENTPATTERN= “eventPtn2” />

<View name=”Activity2” type=”syngo.Common.Activity.Activity, Activity, Version=1.0.l.0, Culture=neutral, PublicKeyToken=null” config-File= “. \PresentationContent\UI_Activcity2.xml” CMDPATTERN= “cmdPtn2” EVENTPATTERN= “eventPtn2” />

<Controller name= “ActivityWorkflowController” />

</Activity>

- <SUBWORKFLOW xml: base= “Subworkflow/Subworkflowl . xml”>

- <TASKS>

- <TASK name= “taskl”>

- <!--

<xi: include href= “ . \Task\taskl.xml” xpointer= “xpointer(//Task)”/>

-->

<Model name=”taskl” ID=”groupl” PATTERNCMD=”patl” PATTERNEVT= “evtl” FILE= “.\BusinessContent\stepl_Model.xml” xml : base= “Task/taskl. xml” />

<View name = “taskl” ID= “groupl” PATTERNCMD=“patl” PATTERNEVT=”evtl” FILE= “.\PresentationContent\subworkflow_step1. xml” xml :base= “Task/taskl .xml” />

</TASK>

- <TASK name=”task2”>

- <! - -

<xi: include href= “\XML-Files\Task\task2 .xml”

xpointer= “xpointer (//Task)”/>

- - >

<Model name = “task2” ID=“group2” PATTERNCMD=“pat2”
PATTERNEVT=”evt2” FILE = “\BusinessContent\step2_Model.xml”
xml :base=”Task/task2 .xml” />
<View name= “task2” ID= “group2” PATTERNCIVD= “pat2” PATTERNEVT=
“evt2” FILE= “. \PresentationContent\subworkflow_step2 . xml”
xml :base= “Task/task2 .xml” />
</TASK>
- <TASK name=”task3”>
- <Task xml :base=”Task/task3 .xml”>
<Model name=” task3” ID= “group3” PATTERNCMD= “pat3” PATTERNEVT
“evt3” FILE=” .\BusinessContent\step3_Model. xml” />
<View name=”task3” ID=“group3” PATTERNCMD= “pat3” PATTERNEVT=
“evt3” FILE=”. \PresentationContent\subworkflow_step3 .xml” />
</Task>
</TASK>
</TASKs>
- <NAVIGATIONGRAPH name= “navgraphl” startTask= “ taskl”>
-<!--
<xi: include href = “.\TaskNavigationGraph\TaskNavigationGraph.xml”
xpointer = “xpointer (//NAVIGATIONGRAPH)”/>
- ->
- <NODE task=”taskl”
xml : base=” TaskNavigationGraph/TaskNavigationGraph. xml”>
<navigateTo navigateValuer”PREVIOUS” task=” task3” />
<navigateTo navigateValue=”NEXT” task = “task2” />
</NODE>
- <NODE task= “task2”
xml :base=“TaskNavigationGraph/TaskNavigationGraph. xml”>
<navigateTo navigateValue=”PREVIOUS” task=”taskl”/>
<navigateTo navigateValue=”NEXT” task=”task3” />
</NODE>

- <NODE task="task3"
xml: base="Task.NavigationGraph/TaskNavigationGraph .xml">
<navigateTo navigateValue="PREVIOUS" task=" task2" />
<navigateTo navigateValue= "NEXT" task="taskl" />
</NODE>
</NAVIGATIONGRAPH>
</SUBWORKFLOW>
</Activity>
- <Activity name="Activity3">
- <Activity xml :base="Activity/Activity3 .xml">
<Model name="MyModel3" type="BusinessProcess.MyModel,
Busi-nessProcess, Version=l.0.1.0, Culture=neutral, PublicKeyToken=null"
iModelManager="MyModelManager" config-File =
"\AT\dotNet\Core\BusinessForm\test.u\config\BizFormConfig . xml"
CMDPATTERN= "cmdPtn3" EVENTPATTERN=" event Ptn3" />
<View name="Activity3" type="syngo.Common.Activity.Activity, Activity,
Version=l.0.l.0, Culture=neutral, PublicKeyToken=null" config-File=
"./PresentationContent/UI_Activity3.xml" CMDPATTERN="cmdPtn3"
EVENTPATTERN="eventPtn3" />
<Controller name="ActivityWorkflowController" />
</Activity>
- <!--
can be optional when activity has no subworkflow <includeSubworkflow href=
". /Subworkflow/Subworkflow3 .xml"
xpointer= "xpointer(//Subworkf low)"/>
- ->
</Activity>
</Activities>
- <navigationGraph name= "Workflow_l" startView="Activityl" state = "State"
statePersist = "MemoryStatePersistence" iViewManager =
"UserControlViewManager">

```
- <!--
<xi : include
href= ".\ActivityNavigationGraph\ActivityNavigationGraph.xml" xpointer=
"xpointer (//navigationGraph)"/>
- ->
- <node view="Activityl"
xml : base = "ActivityNavjgationGraph/ActivityNavigationGraph.xml">
<navigateTo navigateValue="previous" view= "Activity3" />
<navigateTo navigateValue= "next" view="Activity2" />
</node>
- <node view= "Activity2"
xml : base = "ActivityNavigationGraph/ActivityNavigationGraph. xml">
<navigateTo navigateValue="previous" view="Activityl" />
<navigateTo navigateValue="next" view= "Activity3"/>
</node>
- <node view="Activity3"
xml:base="ActivityNavigationGraph/ActivityNavigationGraph.xml">
<navigateTo navigateValue="previous" view="Activity2" />
<navigateTo navigateValue= "next" view= "Activityl" />
</node>
- ->
</navigationGraph>
</workflow>
</document>
```

通过采用有利且基于本发明系统的容器，可以描述其业务逻辑和表现逻辑是根据相同模式建立起来的应用。

此外还可以将一个应用分为独立的子应用，并且使可配置的数据流机来运行该子应用，以便灵活地支持用户作出决定。

优选的，本发明的转换不需要对现有技术作出原理上的修改，而是可以原则上事后作为组件、尤其是作为经过修改或附加的计算机程序产品添加进来。

最后要指出，本发明的描述和实施例原则上不能限制的理解成对本发明的

物理实现。对于有关的技术人员来说，很明显本发明可以部分或完全以软件实现并分布在多个物理产品、在此尤其是计算机程序产品上。

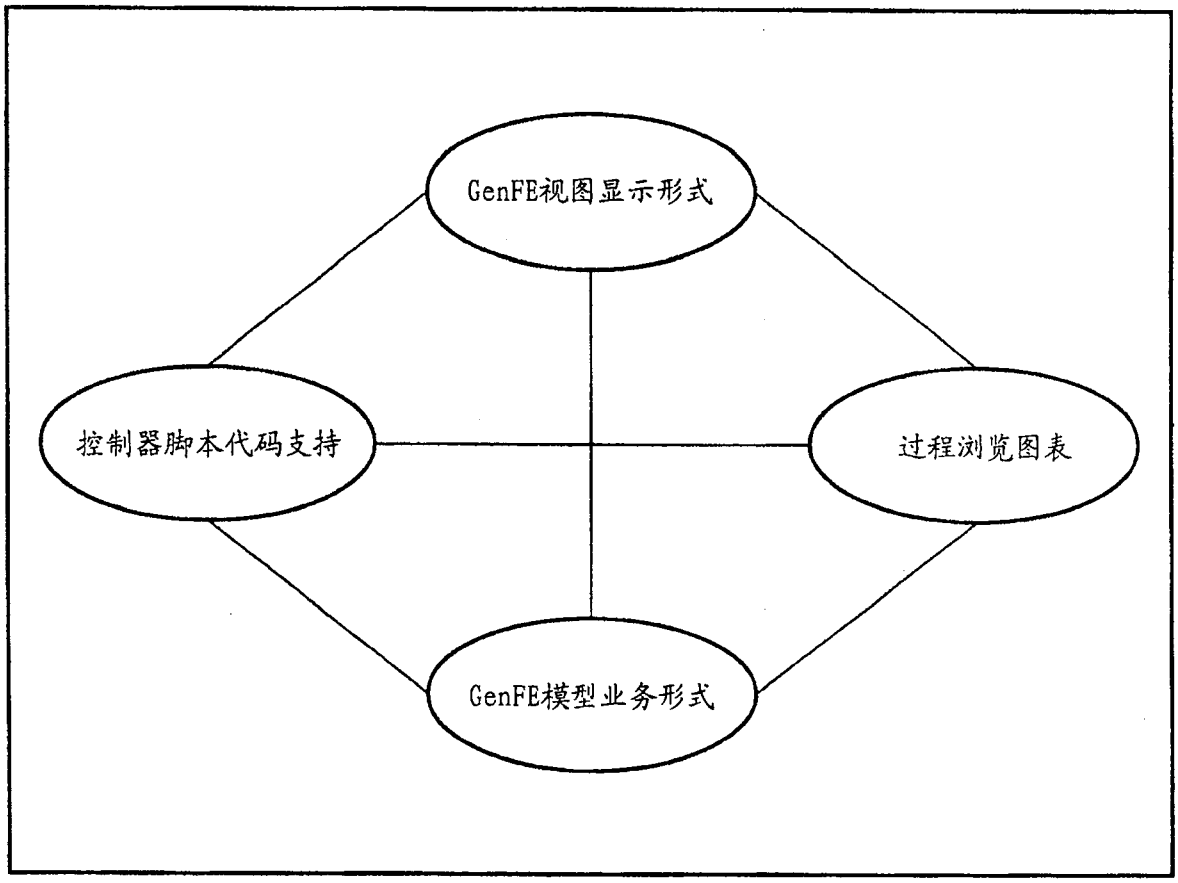


图 1

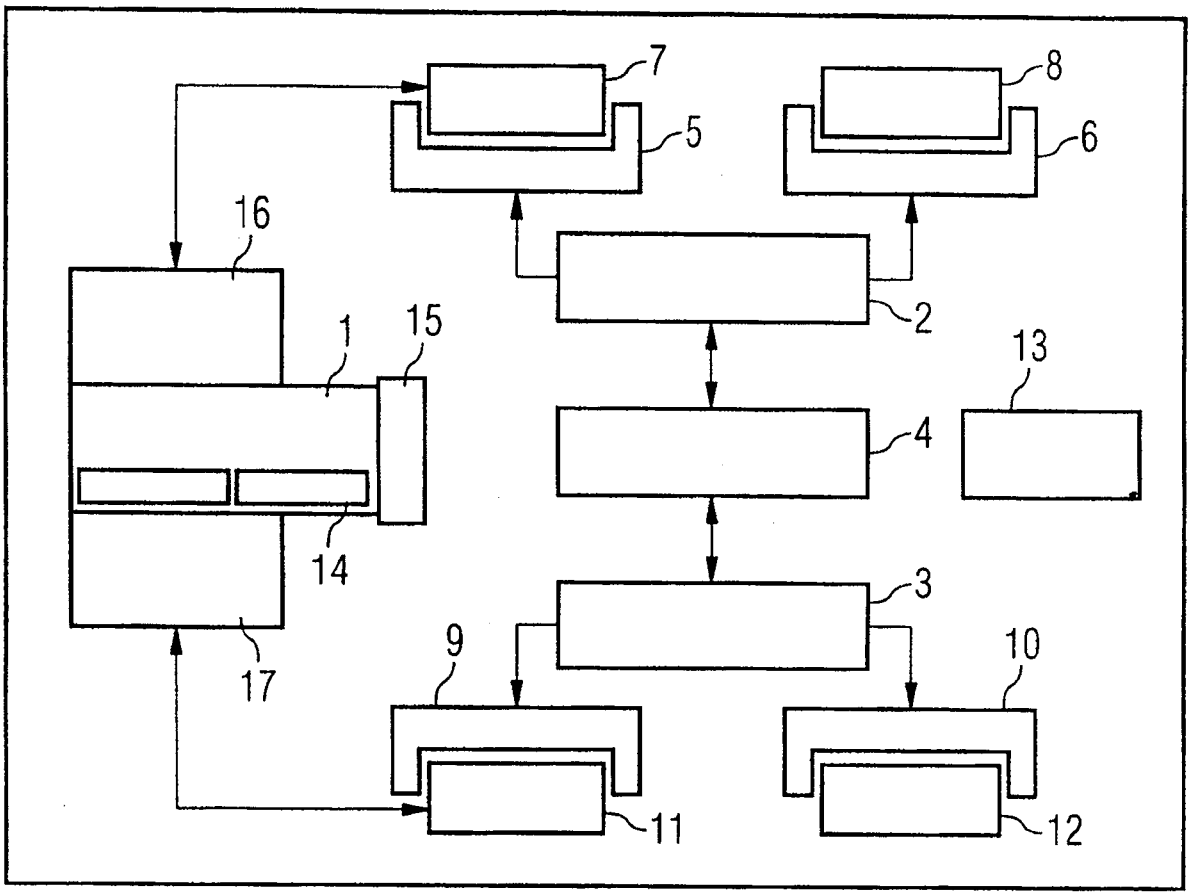


图 2

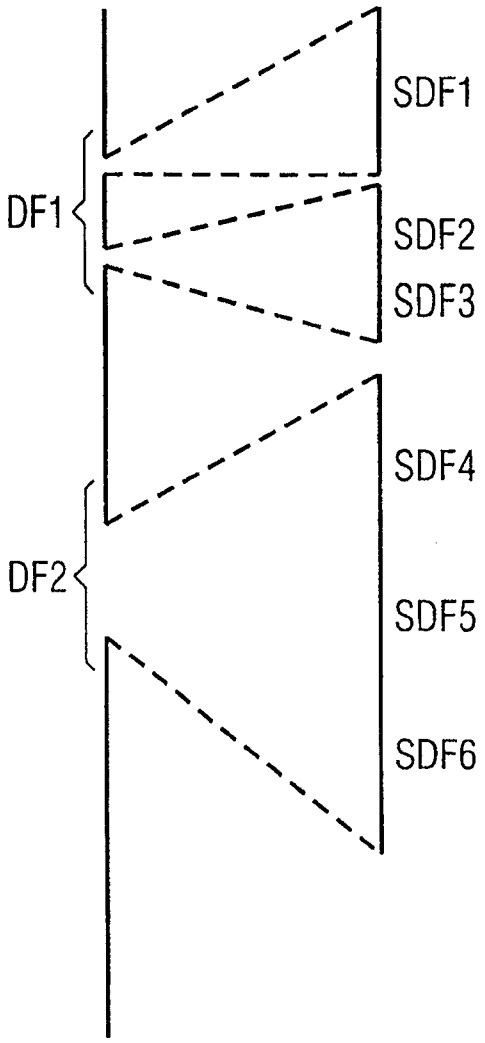


图 3