[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G01R 31/319 (2006.01)

[12] 发明专利申请公布说明书

[21] 申请号 200580015953.4

[43] 公开日 2007 年 6 月 13 日

[11] 公开号 CN 1981202A

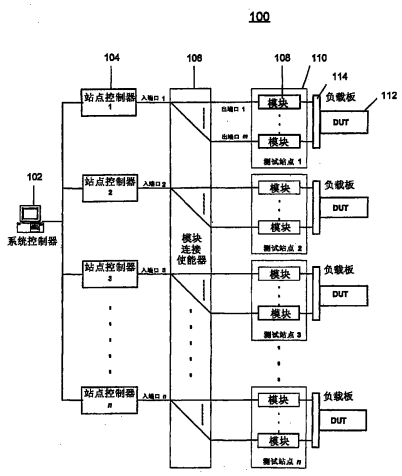
[22] 申请日 2005.5.23
[21] 申请号 200580015953.4
[30] 优先权
[32] 2004. 5.22 [33] US [31] 60/573,577
[32] 2004. 8.13 [33] US [31] 10/917,821
[86] 国际申请 PCT/JP2005/009810 2005.5.23
[87] 国际公布 WO2005/114238 英 2005.12.1
[85] 进入国家阶段日期 2006.11.17
[71] 申请人 株式会社爱德万测试
地址 日本东京都
[72] 发明人 马克·埃尔斯顿 安康·普拉马尼克
[74] 专利代理机构 北京集佳知识产权代理有限公司
代理人 杨生平 杨红梅
权利要求书 4 页 说明书 23 页 附图 11 页

[54] 发明名称

模块化测试系统中的数据日志支持

[57] 摘要

公开了一种用于将测试信息从源传送到目的地的方法。 该方法包括提供模块化测试系统，其中该模块化测试系统包括用于控制至少一个站点控制器的系统控制器，所述至少一个站点控制器用于控制至少一个测试模块。 所述方法进一步包括：提供用于支持用户定义的数据日志格式的扩展的数据日志框架；提供用于支持用户发起的数据日志事件的支持类；接收请求将输入测试信息从所述源传送到所述目的地的数据日志事件；基于所述目的地、数据日志框架和支持类来配置输出测试信息；以及将所述输出测试信息传递到所述目的地。



1. 一种用于将测试信息从源传送到目的地的方法，包括：

提供模块化测试系统，其中该模块化测试系统包括用于控制至少一个站点控制器的系统控制器，所述至少一个站点控制器用于控制至少一个测试模块；

提供用于支持用户定义的数据日志格式的扩展的数据日志框架；

提供用于支持用户发起的数据日志事件的支持类；

接收请求将输入测试信息从所述源传送到所述目的地的数据日志事件；

基于所述目的地、数据日志框架和支持类来配置输出测试信息；以及

将所述输出测试信息传递到所述目的地。

2. 根据权利要求 1 所述的方法，其中所述数据日志框架包括：

用于支持来自不同设备提供商的各个源的输入测试信息的源接口；

用于表示数据日志源和目的地的类的类型接口；

用于提供格式化能力的格式接口；

用于将格式化的测试信息传递到不同目的地的流接口；以及

用于根据对应的源、流、格式和目的地来管理数据日志事件的数据日志管理器。

3. 根据权利要求 1 所述的方法，其中所述支持类包括：

用于存储头部、类型和事件信息的数据日志值类；

用于将用户事件的名称和类型相组合的数据日志格式映射类；

用于捆绑格式映射组的数据日志格式组类；

用于支持用户定义的格式类的用户格式类；以及

用于支持预定义格式类集的一般格式类。

4. 根据权利要求 1 所述的方法，其中提供数据日志框架的步骤进一步包括：

创建一个或多个数据日志流；

创建一个或多个用户格式；

创建一个或多个数据日志格式映射；

创建一个或多个数据日志格式组；以及

将所述一个或多个数据日志流、用户格式、数据日志格式映射和数据日志格式组链接到对应的数据日志管理器。

5. 根据权利要求 1 所述的方法，其中所述数据日志框架独立于输入测试信息的源和内容。

6. 根据权利要求 1 所述的方法，其中所述数据日志框架独立于输出测试信息的目的地和内容。

7. 根据权利要求 1 所述的方法，其中所述输出测试信息的格式能够由用户配置。

8. 根据权利要求 1 所述的方法，其中所述配置步骤包括：

创建数据日志值对象，该数据日志值对象包括类型和事件信息；

基于对应的目的地、数据日志框架和支持类来确定数据日志格式对象；以及

使用所述数据日志格式对象来格式化所述输出测试信息。

9. 一种模块化测试系统，包括：

系统控制器；

耦合到所述系统控制器的至少一个站点控制器；

至少一个测试模块及其对应的被测试器件（DUT）；

数据日志框架，配置成支持用户定义的数据日志格式的扩展；

配置成支持用户发起的数据日志事件的一个或多个支持类；

用于接收请求将输入测试信息从所述源传送到所述目的地的数据日志事件的装置；

用于基于所述目的地、数据日志框架和支持类来配置输出测试信息的装置；以及

用于将所述输出测试信息传递到所述目的地的装置。

10. 根据权利要求 9 所述的系统，其中所述数据日志框架包括：

用于支持来自不同设备提供商的各个源的输入测试信息的源接口；

用于表示数据日志源和目的地的类的类型接口；

用于提供格式化能力的格式接口；

用于将格式化的测试信息传递到不同目的地的流接口；以及

用于根据对应的源、流、格式和目的地来管理数据日志事件的数据日志管理器。

11. 根据权利要求 9 所述的系统，其中所述支持类包括：

用于存储头部、类型和事件信息的数据日志值类；

用于将用户事件的名称和类型相组合的数据日志格式映射类；

用于捆绑格式映射组的数据日志格式组类；

用于支持用户定义的格式类的用户格式类；以及

用于支持预定义格式类集的一般格式类。

12. 根据权利要求 9 所述的系统，其中提供数据日志框架进一步包括：

用于创建一个或多个数据日志流的装置；

用于创建一个或多个用户格式的装置；

用于创建一个或多个数据日志格式映射的装置；

用于创建一个或多个数据日志格式组的装置；以及

用于将所述一个或多个数据日志流、用户格式、数据日志格式映射和数据日志格式组链接到对应的数据日志管理器的装置。

13. 根据权利要求 9 所述的系统，其中所述数据日志框架独立于输入测试信息的源和内容。

14. 根据权利要求 9 所述的系统，其中所述数据日志框架独立于输出测试信息的目的地和内容。

15. 根据权利要求 9 所述的系统，其中所述输出测试信息的格式能够由用户配置。

16. 根据权利要求 9 所述的系统，其中所述用于配置输出测试信息的装置包括：

用于创建数据日志值对象的装置，该数据日志值对象包括类型和事件信息；

用于基于对应的目的地、数据日志框架和支持类来确定数据日志格式对象的装置；以及

用于使用所述数据日志格式对象来格式化所述输出测试信息的装置。

模块化测试系统中的数据日志支持

相关申请的交叉参考

本申请要求 Advantest Corporation 于 2004 年 5 月 22 日提交的临时申请 no. 60/573,577“Software Development in an Open Architecture Test System”的权益，其全部内容通过引用结合于此。

技术领域

本发明涉及自动化测试设备（ATE）领域。更具体而言，本发明涉及一种用于支持开放体系测试系统中的数据日志的方法和系统。

背景技术

芯片上系统（SOC）器件的日益复杂以及同时对芯片测试成本减小的需求已经迫使集成电路（IC）制造商和测试器厂商重新考虑应当如何进行 IC 测试。工业研究表明，如果不重新进行工程设计，测试器的预计成本在不远的将来将持续显著增加。

测试设备高成本的主要原因是常规测试器体系的专门化特性。每个测试器制造商都具有许多测试器平台，不仅在诸如 Advantest、Teradyne 和 Agilent 的公司之间不兼容，而且在一个公司内的平台之间也不兼容，如 Advantest 制造的 T3300、T5500 和 T6600 系列测试器。由于这些不兼容性，每个测试器都需要其自己的专门化硬件和软件组件，并且这些专门化的硬件和软件组件不能用在其它测试器上。另外，需要相当的精力来将测试程序从一个测试器移植到另一个测试器以及开发第三方解决方案。即使当为平台开发了第三方解决方案时，它也不能移植或再用于不同的平台。从一个平台到另一个平台的转换通常是复杂而易于出错的，从而导致附加的精力、时间和增加的测试成本。

当运行一个测试或一系列测试时，数据日志被用于向用户提供状

态信息。数据日志中所报告的信息可包括被测试器件（DUT）的通过/失败状态、任何相关的测量参数以及测试本身的总体运行状态。该信息典型地离线使用以便评估测试运行的完成和正在测试的器件的性能。数据日志能力的支持允许用户将其来自指定源的测试信息以所需格式输出到指定目的地。

专门化测试器体系的问题之一是所有硬件和软件对于给定测试器都保持在固定配置。为测试硬件器件或 IC，开发了专用的测试程序，其使用测试器能力中的一些或全部来限定测试数据、信号、波形以及电流和电压电平，以及收集 DUT 响应和确定 DUT 通过/失败。

由于测试系统需要实施各种各样的功能性和操作以便测试各种各样的测试模块及其对应的 DUT，需要一种可配置成支持各种各样的测试模块的开放体系测试系统。具体而言，为了支持所述各种各样的测试模块，需要一种开放体系测试系统内的数据日志框架，其可配置成与测试系统的不同源和目的地的不同格式一起作用。

发明内容

在本发明的一个实施例中，一种用于将测试信息从源传送到目的地的方法包括：提供模块化测试系统，其中该模块化测试系统包括用于控制至少一个站点控制器的系统控制器，所述至少一个站点控制器用于控制至少一个测试模块。所述方法进一步包括：提供用于支持用户定义的数据日志格式的扩展的数据日志框架；提供用于支持用户发起的数据日志事件的支持类；接收请求将输入测试信息从所述源传送到所述目的地的数据日志事件；基于所述目的地、数据日志框架和支持类来配置输出测试信息；以及将所述输出测试信息传递到所述目的地。

在本发明的另一个实施例中，一种模块化测试系统包括：系统控制器；耦合到系统控制器的至少一个站点控制器；至少一个测试模块及其对应的测试器件（DUT）；数据日志框架，配置成支持用户定义的数

据日志格式的扩展；以及配置成支持用户发起的数据日志事件的一个或多个支持类。该模块化测试系统进一步包括用于接收请求将输入测试信息从所述源传送到所述目的地的数据日志事件的装置；用于基于所述目的地、数据日志框架和支持类来配置输出测试信息的装置；以及用于将所述输出测试信息传递到所述目的地的装置。

附图说明

根据结合附图做出的对本发明实施例的详细描述，本发明的以上特点和优点以及其附加的特点和优点得到更清楚的理解。

图 1 示出根据本发明一个实施例的开放体系测试系统。

图 2 示出根据本发明一个实施例的数据日志框架的实施。

图 3 示出根据本发明一个实施例的数据日志源的实施。

图 4 示出根据本发明一个实施例的用于注册源类型的方法。

图 5 示出根据本发明一个实施例的用于格式化和流化数据的方法。

图 6 示出根据本发明一个实施例的动态使能数据日志的实施。

图 7 示出根据本发明一个实施例的动态禁止数据日志的实施。

图 8 示出根据本发明一个实施例的动态修改数据日志格式的实施。

图 9 示出根据本发明一个实施例的动态分配数据日志输出流的实施。

图 10 示出根据本发明一个实施例的动态禁止数据日志输出流的实施。

图 11 示出根据本发明一个实施例的动态适配新源的实施。

具体实施方式

本发明提供了用于模块化测试系统中的数据日志支持的方法和系统。以下描述是为了使本领域的任何技术人员能够制作使用本发明。对特定技术和应用的描述只是为了举例。对在此所述的例子的各种修改对

本领域的技术人员将是显而易见的，并且在此定义的一般原理可应用于其它例子和应用而不背离本发明的实质和范围。因此，本发明并不是想要局限于所描述和示出的例子，而是应当给予与在此所公开的原理和特征一致的最宽范围。

图 1 示出根据本发明一个实施例的开放体系测试系统。系统控制器（SysC）102 耦合到多个站点控制器（SiteCs）104。系统控制器亦可耦合到网络以便访问关联文件。每个站点控制器通过模块连接使能器 106 耦合以便控制位于测试站点 110 的一个或多个测试模块 108。模块连接使能器 106 允许所连接的硬件模块 108 的重新配置并且也用作用于数据传递的总线（用于加载模式数据、收集响应数据、提供控制等）。另外，一个站点处的模块可通过模块连接使能器 106 访问另一个站点处的模块。模块连接使能器 106 允许不同的测试站点具有相同或不同的模块配置。换句话说，每个测试站点可采用不同数量和类型的模块。可能的硬件实施包括专用连接、开关连接、总线连接、环形连接和星形连接。模块连接使能器 106 可以由例如开关矩阵来实现。每个测试站点 110 与一个 DUT 112 关联，该 DUT 112 通过负载板 114 连接到对应站点的模块。在另一个实施例中，单个站点控制器可连接到多个 DUT 站点。

系统控制器 102 用作总体系统管理器。它协调站点控制器活动、管理系统级并行测试策略，并且另外提供处理机/探测器控制以及系统级数据日志和错误处理支持。系统控制器 102 是在验证和调试测试环境时测试工程师交互的主要点。它提供到站点控制器 104 的关口，并且管理多 DUT 环境中的站点控制器活动的同步。它进一步运行用户应用和工具，如数据日志图形用户界面（DatalogGUI）。依赖于操作设置，系统控制器 102 可在与站点控制器 104 的操作分离的 CPU 上配置。可替选地，公用的 CPU 可由系统控制器 102 和站点控制器 104 来共享。类似地，每个站点控制器 104 可在其自己的专用 CPU（中央处理单元）上配置，或配置为同一 CPU 内的单独进程或线程。

站点控制器 104 负责运行测试 DUT 的测试计划。测试计划通过使用框架类以及封装测试方法的标准或用户提供的测试类来建立特定的测试。另外，测试计划使用标准接口来配置硬件，并且定义测试流程。

本发明的系统体系可在概念上设想为图 1 所示的分布式系统，同时应理解各个系统组件亦可被当做集成单片系统的逻辑组件，而不必当做分布式系统的物理组件。通过在硬件和软件级使用标准的接口，可使即插即用或可更换模块变得容易。测试器操作系统（TOS）允许用户使用测试计划编程语言来编写测试计划程序，以及以特定于特定测试器件（DUT）的方式操作测试系统。它亦允许用户将测试计划程序中公用的测试系统操作序列打包为库。这些库有时被称为测试类和测试模板。

数据日志框架

数据日志框架连同测试计划和系统框架类运行在各个站点控制器上。它提供一组接口、类和方法来支持数据日志系统的开发、应用和管理。该数据日志框架

管理数据日志单元，如源、流和格式器；

分派（dispatch）数据日志事件；

提供允许创建新数据日志事件的应用编程接口（API）；

提供 API 来设置和控制数据日志；

提供通用格式器来处理共同的格式化需要；并且

提供对模块化第三方或顾客数据日志服务的集成的支持，这使得数据日志框架开放。

标准数据日志接口

图 2 示出根据本发明一个实施例的数据日志框架的实施。数据日志框架接口和支持类之间的关系集通过图 2 的统一建模语言（UML）类图而示出。数据日志框架 200 包括数据日志流接口 202、数据日志格式

接口 204、数据日志类型接口 206 和数据日志源接口 208。数据日志框架 200 进一步包括数据日志管理器类 210、数据日志格式映射类 212、数据日志格式组类 214、数据日志值类 216、一般格式类 218 和用户格式类 220。

到测试器操作系统（TOS）的标准数据日志接口被定义为纯抽象 C++类。数据日志流接口（IDatalogStream）202 表示输出文件或设备。数据日志系统将格式化输出发送到流。该系统提供两个内置流，用于将输出发送到本地盘文件的 FileStream 和用于将输出发送到系统的控制台应用的 ConsoleStream。

数据日志格式接口（IDatalogFormat）204 表示将与数据日志事件关联的数据格式化为输出所需要的必要指令。格式化能力的特定特性隐藏在 IDatalogFormat 的每个实施内。例如，内置的 GeneralFormat 允许一般格式串与类宏（macro-like）能力一起使用，所述类宏能力用于从 DatalogValues 对象中提取带名字的值，并且将它们插入到格式化串中。

数据日志类型接口（IDatalogType）206 表示数据日志源的类或“类型”，如测试的特定类型。DatalogManager 210 部分地基于施加源的类型将事件映射成格式和流。在新的源类型产生时，它们向 DatalogManager 210注册自己并且接收标识符(ID)值。管理这些类型允许DatalogManager 210 使能/禁止数据日志源的全部类以及数据日志源的特定实例。IDatalogType 对象 206 可产生数据日志事件的多个类。例如，在一组模式内循环直到发生某个条件为止的用户测试可产生 TestStart、TestEnd、IterationStart、IterationEnd 和 ConditionTest 数据日志事件。这些事件类中的每个可具有与其关联的不同数据字段（见以下的 DatalogValues 类）。

数据日志源接口（IDatalogSource）208 表示数据日志事件的源。尽管 IDatalogType 206 表征了数据日志源对象的整个类，IDatalogSource 208 表示单独的源实例。该接口允许使能/禁止各个源的数据日志。

数据日志支持类

数据日志框架 200 的中心对象是 DatalogManager 对象 210。该对象负责维护所有数据日志输出流以及格式化数据日志事件及其关联数据并将其前转到适当的流。DatalogManager 210 驻留在站点控制器的测试计划服务器(TPS)中。它是数据日志执行引擎。具体而言，DatalogManager 210 的主要功能包括：

管理全局数据日志流集（即实现 DatalogStream 接口的对象）。

管理管理数据日志格式集（即实现 DatalogFormat 接口的对象），每个格式产生格式化串，作为数据日志事件集的结果。

将数据日志事件分配到所有关联的数据日志流。流的所有目的地可通过 DatalogManager 来控制。

保持和控制具有过滤数据日志输出的效果的数据日志屏蔽条件。注意屏蔽的应用可发生在源处以便改善系统性能。

启动/清理数据日志系统。

使能/禁止整个数据日志系统以及各个数据日志类型或源。

数据日志格式映射类 212 是带名字的<EventType, EventName, Format>组合。也就是说，它将 EventType（表示特定的 IDatalogType 对象）、 EventName（表示 EventType 对象所产生的数据日志事件之一）和 Format（表示 IDatalogFormat 对象）分组在一起。

数据日志格式组类 214 是 DatalogFormatMap 对象的集合。用户可将 DatalogFormatMap 对象捆绑成单个组并且将该组分配给一个或几个流。这提供了对数据日志事件最终格式化以及这些格式化串到适当的流的路由的用户级控制。

数据日志值类 216 包含头部信息、类似数据日志类型标识符和事件以及由事件源设置的数据日志字段的列表。DatalogValues 对象充当从名称到值的串到串映射，从而允许格式对象通过名称来提取所需值并且将其插入到格式化流中。

一般格式类 218 是 IDatalogFormat 的实施。它允许一般格式串与类宏能力一起使用，来从 DatalogValues 对象中提取指定值并且将其插入到格式化流中。

用户格式类 220 是用于用户定义的格式类的占位符。如果所提供的 GeneralFormat 类不符合用户的特定需要，人们可以向系统添加新的格式类并且对其进行利用。

数据日志源

在一个实施例中，通过实现 IDatalogSource 接口 208，基于站点控制器的对象可充当数据日志事件的源。例如，测试计划和测试对象是事件源的实例。这样的对象的每个实例都实现了 IDatalogSource 接口。另外，这样的对象的每种类型，例如 FunctionalTest，具有与其关联的 IDatalogType 接口 206 的实施。这是利用每种类型的数据日志源的类范围内的（class-wide）静态对象来完成的。图 3 示出根据本发明一个实施例的数据日志源的实施。该数据日志源包括功能测试()类 302、数据日志类型实施()类 304、数据日志类型接口 306 和数据日志源接口 308。

如图 3 所示，FunctionalTest 类 302 具有类型 DatalogTypeImpl 304 的类范围内的静态成员。当该成员变量创建时，FunctionalTest 的实例对其进行检查。如果该变量尚未创建和赋值，则构造器创建 DatalogTypeImpl 的实例，对其进行初始化，将其赋给静态成员变量，并且将其向 DatalogManager 注册。FunctionalTest 类的后续实例访问 DatalogTypeImpl 数据成员，并且将自己（即作为 IDatalogSource 的 FunctionalTest 的实例）添加到后面的 FunctionalTest 实例提供的 DatalogTypeImpl 的数据日志源实例列表。

在另一个实施例中，例如 ADifferentFunctionalTest 的 FunctionalTest 类的子类可尝试建立与 FunctionalTest 的数据日志类型不同的数据日志类型。通过不使用类范围内的与 FunctionalTest 关联的

DatalogTypeImpl 而使用其自己的类范围内的 DatalogTypeImpl 实例，一个 ADifferentFunctionalTest 的实施可实现这一点。注意每个 FunctionalTest 实例都实施 IDatalogSource 接口。DatalogTypeImpl 使用该接口来使能/禁止各个源。

源类型注册

当测试计划加载时，数据日志事件源可向 DatalogManager 对象注册。从该注册返回的值是标识符，该标识符然后用于生成事件。

图 4 示出根据本发明一个实施例的用于注册源类型的方法。用于注册源类型的合作对象组包括测试计划对象 402、FunctionalTest1 对象 404、FunctionalTest2 对象 406 和 DatalogManager 对象 408。注意当测试计划创建相同类型（FunctionalTest1 和 FunctionalTest2）的两个测试实例时，第一测试实例将新的数据日志类型向 DatalogManager 注册。

用户可为不同的事件类型设置数据日志格式。这可在测试计划初始化期间进行以便提供缺省格式，并且/或者从远程应用交互地进行。遵循类似的过程以便允许用户设置数据日志流。DatalogManager 维护从{数据日志类型，数据日志事件}组到对应的{数据日志格式，数据日志流}的映射，其规定了特定类型的事件可采取来通过测试系统的路由。对于基于文件的流，站点控制器上的测试类可将流关闭，从而允许数据日志文件传递到系统控制器。然后该流可重新打开或替换为另一个流。

对数据进行格式化和流化

在测试期间，对象可产生记录到数据流的数据日志事件。图 5 示出根据本发明一个实施例的用于格式化和流化数据的方法。用于执行该任务的合作对象和接口组包括测试计划对象 502、DatalogValues 对象 504、DatalogManager 对象 506、数据日志格式接口（IDatalogFormat）508 和数据日志流接口（IDatalogStream）510。测试计划 502 创建

DatalogValues 对象实例 504，通过调用 setValue()方法来设置适当的值，并且将该对象传递到 DatalogManager 506 的 doDatalog()方法。然后 doDatalog()方法寻找用于类型/事件组合的适当数据日志格式对象 508，并且调用关于该数据日志格式对象 508 的 apply()方法以便在返回中获得串对象。然后使用 writeMessage()方法将该串对象传递到关联的数据日志流对象 510 以便输出。

数据日志初始化

在一个不同的实施例中，使用以下步骤来初始化数据日志系统：

创建数据日志流并将其添加到 DatalogManager。

创建数据日志格式并将其添加到 DatalogManager。

创建数据日志格式映射并将其添加到 DatalogManager。

创建数据日志格式组并将其添加到 DatalogManager。

数据日志应用编程接口（API）提供了执行上述任务的功能。另外，测试系统提供了测试类 DatalogSetupTest，其读取一个或多个配置文件并执行上述步骤。以下的例子说明了使用 DatalogSetupTest 配置文件来执行初始化步骤。数据日志流、数据日志格式映射和数据日志格式组内的 DLL 参数规定了用于实现 IDatalogStream 202 或 IDatalogFormat 204 的库。

```
Version 0.1.0;
# Enable the datalog system
Enabled;

# Step 1: Stream Creation

# Creates a stream of type FileStream, which logs the message to
# a file.
Stream FunctionalTestStream
{
        DLL "FileStream";
        FileName "FuncDatalogDut<DutID>.log";
```

```
        Overwrite "1";
}

# Step 2: Format Creation

# Adds a format with a line that looks like:
#   Signal Info: $SignalInfo
# where $SignalInfo is replaced by the string provided in the datalog
# source.
Format SignalInfoFormat
{
        DLL "GeneralFormat";
        Format "Signal Info: $SignalInfo";
} # Adds a format with a line that looks like:
#   Test Result: $Result
# where $Result is replaced by the string provided in the datalog
# source.
Format TestResultFormat
{
        DLL "GeneralFormat";
        Format "Test Result: $Result";
}

# Step 3: FormatMap Creation

# Maps format SignalInfoFormat to event DumpSignal from sources
# of type com.Advantest.oai.TestClasses.FunctionalTest.
FormatMap SignalFormatMap
{
        Type com.advantest.oai.TestClasses.FunctionalTest;
        Event DumpSignal;
        Format SignalInfoFormat;
}

# Maps format TestResultFormat to event TestResult from sources
# of type com.Advantest.oai.TestClasses.FunctionalTest.
FormatMap ResultFormatMap
{
        Type com.advantest.oai.TestClasses.FunctionalTest;
        Event TestResult;
        Format TestResultFormat;
}

# Step 4: FormatGroup Creation
# Creates a group which directs messages from format map
# SignalFormatMap and ResultFormatMap to destination FunctionalTestStream
# and ConsoleStream.
FormatGroup FunctionalTestGroup
{
```

```
FormatMap SignalFormatMap;
FormatMap ResultFormatMap;

Stream FunctionalTestStream;
Stream ConsoleStream;
}
```

数据日志设置文件的语义

在又一个实施例中，数据日志设置文件包括四个不同类型的块：数据日志格式块、数据日志流块、数据日志格式映射块和数据日志格式组块。这些块分别用于定义数据日志格式、数据日志流、数据日志格式映射和数据日志格式组。这四个块中的任何一个可在同一设置文件中使用多次，并且数据日志系统亦可利用多个单独的数据日志设置文件来设置。这样，数据日志系统设置是模块化的且灵活的。例如，可以有用于设置 Cal/Diags 数据日志的第一数据日志设置文件、用于设置功能测试的数据日志的第二数据日志设置文件以及用于设置参数测试的数据日志的第三数据日志设置文件。

数据日志格式块

在一个实施例中，数据日志格式块用于定义数据日志格式。该格式定义可包括格式名称、格式 DLL 名称、消息格式串和任选的用户定义的格式参数。如果格式 DLL 名称是 GeneralFormat，则使用预定义的 GeneralFormat；否则使用用户定义的数据日志格式。在后者的情况下，用户在系统测试目录中提供数据日志格式 DLL。调用以下数据日志 API 来定义数据日志格式块：

```
void DatalogManager::addFormat(const OFCString &formatName,
                               const OFCString &typeName,
                               const OFCString &format,
                               const DatalogProperties_t properties);
```

并且

```
Format SignalInfoFormat
{
        DLL "GeneralFormat";
        Format "$SignalInfo";
}
```

定义格式 SignalInfoFormat，其利用了预定义的 GeneralFormat。其格式串是$SignalInfo，其中$SignalInfo 是可在运行时间期间由传入的（passed-in）数据日志记录（DatalogValues）中的数据日志变量 SignalInfo 替换的令牌。数据日志格式块可用于将现有数据日志格式从数据日志系统中去除。在此情况下调用以下数据日志 API。

```
void DatalogManager::removeFormat(const OFCString &formatName);
```

并且

```
Format AnExistingFormat Disabled;
```

将数据日志格式 *AnExstingFormat* 从数据日志系统中去除。

数据日志流块

在另一个实施例中，数据日志流块用于定义数据日志流。数据日志流的定义包括流名称、流 DLL 名称和任选的用户定义的流参数。如果流 DLL 名称是 FileStream 或 ConsoleStream，则使用内置数据日志流 FileStream 或 ConsoleStream；否则使用用户定义的数据日志流。在后者的情况下，用户在系统测试目录中提供数据日志流 DLL。调用以下数据日志 API 来定义数据日志流块：

```
void DatalogManager::addStream(const OFCString &streamName,
                               const OFCString &typeName,
                               const DatalogProperties_t properties);
```

并且

```
Stream FunctionalTestStream
{
    DLL "FileStream";
    FileName "datalog<DutID>.log";
    Overwrite "0";
}
```

定义流 FunctionalTestStream，其利用了具有任选参数 FileName 和 Overwrite 的内置流 FileStream。注意 FileStream 支持具有自动变量<DutID>和<TimeStamp>的文件名。<DutID>在运行时间期间由当前 DUT ID 替换，而<TimeStamp>由具有诸如 Thu_Nov_13_06_10_28_2003 的格式的当前日期和时间来替换。

数据日志流块可用于将现有数据日志流从数据日志系统中去除。在此情况下调用以下数据日志 API：

```
void DatalogManager::removeStream(const OFCString &streamName);
```

并且

```
Stream AnExistingStream Disabled;
```

将数据日志流 *AnExsitingStream* 从数据日志系统中去除。

数据日志类型块

在一个不同的实施例中，数据日志类型块用于针对数据日志类型来定义特定于用户的特性。调用数据日志 API 来定义数据日志类型块：

```
IDatalogType *DatalogManager::getType(const OFCString &typeName)
const;

void IdatalogType::addProperty(const OFCString &propName,
                               const OFCString &propValue);
```

并且针对数据日志类型 "*com.Advantest.oai.TestClasses. FunctionalTest.*" ，

```
Type com.Advantest.oai.TestClasses.FunctionalTest
{
    Mode "Detailed";
}
```

定义了具有值“*Detailed*”的特性 *Mode*。

数据日志格式映射块

在又一个实施例中，数据日志格式映射块用于定义数据日志格式映射。格式映射的定义包括格式映射名称、数据日志类型名称及其待映射的事件名称以及映射的格式名称。调用以下数据日志 API 来定义数据日志格式映射块：

```
void DatalogManager::createFormatMap(const OFCString &mapName,
                                     const OFCString &typeName,
                                     const OFCString &eventName,
                                     const OFCString &formatName);
```

并且

```
FormatMap SignalFormatMap
{
    Type com.Advantest.oai.TestClasses.FunctionalTest;
    Event DumpSignal;
    Format SignalInfoFormat;
}
```

定义了格式映射 SignalFormatMap，其将数据日志类型 *com.Advantest.oai.TestClasses.FunctionalTest* 及其事件 *DumpSignal* 映射为格式 *SignalInfoFormat*。

数据日志格式映射块可用于将现有数据日志格式映射从数据日志系统中去除。在此情况下，调用以下数据日志 API 来执行任务：

```
void DatalogManager::removeFormatMap(const OFCString &mapName);
```

数据日志格式组块

在一个实施例中，数据日志格式块用于定义数据日志格式组。格式组的定义包括格式组名称、格式组包含的格式映射名称的列表以及从格式组产生的消息所输出到的流的名称列表。调用以下数据日志 API 来定义数据日志格式组块：

```
void DatalogManager::createFormatGroup(const OFCString &groupName,
                    const OFCStringVec_t &mapNames);
void DatalogManager::setStream(const OFCString &groupName,
                    const OFCString &streamName);
```

第一调用创建包含格式矢量的格式组，并且第二调用将数据日志流添加到所创建的格式组中。例如，

```
FormatGroup FunctionalTestGroup
        {
            FormatMap SignalFormatMap;
            FormatMap ResultFormatMap;
            Stream FunctionalTestStream;
            Stream ConsoleStream;
        }
```

定义格式组 FunctionalTestGroup，其包括格式映射 SignalFormatMap 和 ResultFormatMap。从所述格式组产生的消息被输出到数据日志流 FunctionalTestStram 和 ConsoleStream。

数据日志格式映射块可用于将现有数据日志格式组从数据日志系统中去除。在此情况下，调用以下数据日志 API 来实施功能：

```
void DatalogManager::removeFormatGroup(const OFCString
&groupName);
```

数据日志系统应用的例子

在以下例子中，诸如 DatalogManagerProxy、DatalogHandlerProxy 和 DatalogFilterProxy 的代理对象由运行系统控制器的应用用来远程控制一个或多个站点控制器上的数据日志框架的当前状态和操作。这些对象充当站点控制器上的真实对象的远程代理并且向这些对象提供透明的通信通道，该透明的通信通道允许系统控制器上的应用应对本地对象，而不是通信协议的复杂性。

图 6 示出根据本发明一个实施例的动态使能数据记录的实施。用于实现该任务的合作对象组包括数据日志图形用户界面（DatalogGUI）对象 602、DataloggerProxy 对象 604（也称为 DatalogManagerProxy）、测

试对象 606 和数据记录器对象 608（也称为 DatalogManager）。序列图说明用于动态使能数据日志系统的步骤。如图 6 所示，序列图中的每个步骤描述如下：

1. 在步骤 1，DatalogGUI 602 或其它测试器 GUI（例如测试控制面板）包括按钮或菜单项使能数据日志。当用户点击所述按钮或菜单项时，DatalogGUI 602 从测试计划服务器代理（TPSProxy）搜索 DataloggerProxy 对象 604。

2. DatalogGUI 602 对返回的 DataloggerProxy 604 调用 setEnable()方法，从而传入参数值“true”，这意味着使能数据日志系统。

3. 利用该代理模型，DataloggerProxy 604 的 setEnable()方法调用数据记录器 608 的 setEnableTest()方法以使能数据日志系统。

4. 测试对象 606 调用 enteringText()方法以将数据日志事件发出到数据记录器 608。

5. 数据记录器 608 检查其是否通过调用 isEnabled()方法而使能。

6. 如果数据记录器 608 被使能，它调用 log()方法来开始数据记录。

图 7 示出根据本发明一个实施例的动态禁止数据记录的实施。用于实现该任务的合作对象组包括 DatalogGUI 对象 702、DataloggerProxy 对象 704 和数据记录器对象 708。序列图说明用于动态禁止数据日志系统的步骤。如图 7 所示，序列图中的每个步骤描述如下：

1. 在步骤 1，DatalogGUI 702 或其它测试器 GUI（例如测试控制面板）包括按钮或菜单项禁止数据日志。当用户点击所述按钮或菜单项时，DatalogGUI 从 TPSProxy 搜索单态 DataloggerProxy 对象 704。

2. DatalogGUI 702 对返回的 DataloggerProxy 704 调用 setEnable()方法，从而传入参数值“false”，这意味着禁止数据日志系统。

3. 利用该代理模型，DataloggerProxy 704 的 setEnable()方法调用数据记录器 708 的 setEnableTest()方法以禁止数据日志系统。

4. 测试对象 706 调用 enteringText()方法以将数据日志事件发出到数

据记录器 708。

5. 数据记录器检查其是否使能。由于其被禁止，不会发生数据记录。

图 8 示出根据本发明一个实施例的动态修改数据日志格式的实施。用于实现该任务的合作对象组包括 DatalogGUI 对象 802、DatalogFormatterProxy 对象 804、DatalogHandler 对象 806 和 DatalogFormatter 对象 808。序列图说明用于动态改变数据日志格式的步骤。DatalogHandler 对象 806 允许用户以单个命名来将数据日志流和格式规格组合在一起，DatalogManager 可使用所述单个命名来传递数据日志事件。如图 8 所示，序列图中的每个步骤描述如下：

1. 在步骤 1，DatalogGUI 802 显示与所选数据日志处理机关联的可用数据日志格式器的列表。用户可通过打开下拉菜单来选择任何数据日志格式器，并且选择菜单项 Edit。DatalogGUI 的实施将所述列表中的每个数据日志格式器与对应的数据日志格式器代理参考（通过使用用户数据参数）相关联。接下来获得 DatalogFormatterProxy 对象 804。

2. DatalogGUI 802 使用 DatalogFormatterProxy 对象 804 来调用 getFormat()方法以获得格式串和相关参量。

3. 利用该代理模型，DatalogFormatterProxy 804 的 getFormat()方法调用 DatalogFormatter 808 的对应 getFormat()方法。

4. DatalogGUI 802 显示具有所选格式器的数据日志事件的格式和参量。用户可根据一组预定数据日志格式要求来改变格式和参量。

5. 当用户将修改的格式器应用到数据日志系统中时，DatalogGUI 802 调用 DatalogFormatterProxy 804 的 setFormat()方法并且传入修改的格式串和参量。

6. 利用该代理模型，DatalogFormatterProxy 804 的 DatalogFormatterProxy.setFormat()方法调用 DatalogFormatter 808 的对应 setFormat()方法。

7. 接下来，当 DatalogHandler 806 对修改的数据日志格式器调用

getOutput()方法以获得具有所述数据日志事件的格式化的消息时，应用修改的格式器。

图 9 示出根据本发明一个实施例的动态分配数据日志输出流的实施。用于实现该任务的合作对象组包括 DatalogGUI 对象 902、DatalogManagerProxy 对象 904、IDatalogHandlerProxy 对象 906、IDatalogHandler 对象 908 和 DatalogManager 对象 910。序列图说明用于分配数据日志输出流以便与数据日志处理机动态关联的步骤。如图 9 所示，序列图中的每个步骤描述如下：

1. 在步骤 1，当用户选择数据日志处理机来编辑其关联的数据日志输出流时，DatalogGUI 902 显示一面板，包括两个列表：第一列表显示数据日志系统中的数据日志流名称，第二列表显示与数据日志处理机关联的数据日志流名称。另外，DatalogGUI 亦包括 Add、Remove、Apply、OK 和 Cancel 按钮。

2. DatalogGUI 902 调用 DatalogManagerProxy 904 的 getStreamNames()方法。

3. 利用该代理模型，DatalogManagerProxy 904 的 getStreamNames()方法调用 DatalogManager 910 的对应 getStreamNames()方法。

4. DatalogGUI 902 在第一列表中显示所检索的数据日志流名称。

5. DatalogGUI 902 然后调用 IDatalogHandlerProxy 906 的 getStreamNames()方法。

6. 利用该代理模型，IDatalogHandlerProxy 906 的对应 getStreamNames()方法调用 IDatalogHandler 908 的对应 getStreamNames()方法以获得当前与所选数据日志处理机关联的所有数据日志流名称。

7. DatalogGUI 902 在第二列表中显示与所选数据日志处理机关联的所检索的数据日志流名称。

8. 然后用户使用按钮 Add 和 Remove 来编辑关联数据日志流列表。在用户完成编辑并且点击 Apply 或 Ok 按钮以便将修改的流应用于数据

日志系统之后，DatalogGUI 902 对 IDatalogHandlerProxy 906 调用 setStream()方法并向其传递修改的数据日志流名称。

9. 利用该代理模型，IDatalogHandlerProxy 906 的 setStream()方法调用 IDatalogHandler 908 的对应 setStream()方法。

10. 然后，IDatalogHandler 908 从 DatalogManager 910 获得每个数据日志流名称的关联的数据日志流对象参考。

11. IDatalogHandler 908 然后将所述流添加到流矢量中。之后，IDatalogHandler 908 以流矢量发出新选择的数据日志流，用于其数据日志输出。

图 10 示出根据本发明一个实施例的动态禁止数据日志输出流的实施。用于实现该任务的合作对象组包括 DatalogGUI 对象 1002、IDatalogStreamProxy 对象 1004、IDatalogHandler 对象 1006 和 IDatalogStream 对象 1008。序列图说明用于动态禁止数据日志输出流的步骤。如图 10 所示，序列图中的每个步骤描述如下：

1. 在步骤 1，DatalogGUI 1002 显示可用数据日志流的列表。用户可通过打开下拉菜单来选择任何数据日志流，并且选择菜单项 Disable。DatalogGUI 1002 的实施将所述列表中的每个数据日志流与对应的 IDatalogStreamProxy 参考 1004（通过使用用户数据参数）相关联。然后获得 IDatalogStreamProxy 1004。

2. DatalogGUI 1002 对所获得的 IDatalogStreamProxy 对象 1004 调用 disable()。

3. IDatalogStreamProxy 1004 的 disable()方法通过代理模型来调用 IDatalogStream 1008 的对应 disable()方法。

4. IDatalogHandler 1006 调用 write()方法来将格式化的消息写出到所述流。

5. IDatalogStream 1008 检查其是否使能。如果使能则可修改输出流。

图 11 示出根据本发明一个实施例的动态适配新源的实施。用于实

现该任务的合作对象组包括DatalogGUI对象1102、DatalogManagerProxy对象1104、测试对象1106、DataLogger对象1108、DatalogManager对象1110、DatalogHandlerProxy对象1112、DatalogHandler对象1114、DatalogFilterProxy对象1116、DatalogFilter对象1118和DatalogFormatter对象1120。序列图说明用于动态适配新源的步骤。DatalogFilter对象1118允许用户根据类型和源标识符来选择性地过滤数据日志事件。该过滤时用户打开和关闭特定数据日志事件。如图11所示，序列图中的每个步骤描述如下：

1. 在步骤1，DatalogGUI 1102寻找DatalogManagerProxy对象1104，并且调用其addHandler()方法来注册该新源的新DatalogHandler实例。注意以这种途径，DatalogHandler1114将事件与不同的数据日志格式器相关联。利用该模型，新的DatalogHandler实例得以创建以便对所述新源（测试类）起作用。返回新创建的DatalogHandlerProxy实例。

2. 利用该代理模型，DatalogManagerProxy 1104的addHandler()方法调用DatalogManager 1110的对应addHandler()方法。

3. 接下来，DatalogManager1110的addHandler()方法创建DatalogHandler的新实例，然后它将新创建的DatalogHandler实例注册到DatalogManager对象1110。

4. 然后，DatalogGUI 1102通过在步骤1返回的DatalogManagerProxy 1104将数据日志过滤器代理与新创建的数据日志处理机相关联。

5. 利用该代理模型，DatalogHandlerProxy 1112的getFilter()方法调用DatalogHandler 1114的对应getFilter()方法。

6. 利用DatalogFilterProxy 1116，DatalogGUI 1102调用其enableTest()方法。

7. 利用该代理模型，DatalogFilterProxy 1116的enableTest()方法调用DatalogFilter 1118的对应enableTest()方法。结果，新创建的数据日志

处理机实例对新测试源的所选事件进行处理。

8. DatalogGUI 1102 通过其代理为新创建的数据日志处理机的每个所选数据日志事件设置格式。对于此新测试源，新参量的格式可以规定为 Test.FooArgument。在该新测试源中，可实现如下示出的接口 IProperty。

```
class IProperty
    {
    public:
                OFCString &getProperty(const OFCString &name) const;
                void setProperty(const OFCString &name, const OFCString
*value);
                OFCStringVec_t getPropertyNames() const;
    };
```

其中 FooArgument 是所述新测试源的特性之一。

9. DatalogHandlerProxy 1112 通过对应的 DatalogHandler 1114 来调用 setFormat()方法。

10. DatalogHandler 1114 为所选事件创建 DatalogFormatter 并将其添加到事件路由图中。接下来，所述新创建的数据日志处理机处理来自新源的数据日志事件。然后，可以开始具有新源测试的测试计划。

11. 一旦 TPS 达到新源测试的执行的结尾，它就向数据日志系统发送 exitingTest()数据日志请求。

12. Datalogger 1108 接受该请求，并将其传递到数据日志事件以便记录。

13. Datalogger 1108 将该新数据日志事件前转到 DatalogManager 1110 以便分派。

14. DatalogManager 1110 将所述事件逐一发布到所注册的数据日志处理机。一旦数据日志处理机能够处理该事件，可停止该发布过程。

15. 当所述事件被发布到每个数据日志处理机时，处理机检查该事件是否可记录。如果新创建的 DatalogHandler 实例能够用于所述新测试源，则该事件是可记录的。

16. DatalogHandler 1114 寻找与当前事件关联的 DatalogFormatter

1120，并且调用其 getOutput()方法以基于定义的格式来获得格式化的消息。所述 getOutput()方法通过调用

```
cvent.getIDatalogger->getTest()->getProperty("FooArgument");
```

来获取新的参数 Test.FooArgument 值，然后格式化的输出消息被前转到输出数据日志流。

所公开的数据日志框架实现了许多有益效果。首先，数据日志框架独立于数据日志的源、特性和内容。这允许无需修改数据日志框架来添加新的数据日志源。另外，数据日志输出的格式化独立于数据日志框架。尽管为系统提供了 GeneralFormat，但它被规定为到格式块的动态链接库（DLL）参数，并且它不是“硬编码”到框架中的。而且，数据日志事件的格式化可由终端用户来配置。例如，事件可以格式化为人可读的文本、用于电子表格或数据库的逗号隔开的值、专用文本或用于进一步处理的二进制格式，或者它们甚至可以全部忽略。数据日志框架独立于数据日志流的目的地。此外，格式和输出流可扩展。也就是说，用户可向系统添加新的 IdatalogFormat 和 IDatalogStream 实施而无需修改框架。测试类（和其它数据日志源）及其对应的数据日志事件也可扩展，因此框架不需要了解特定的源或事件类型。在用户添加具有新数据日志事件类型的新测试类时，不需要修改数据日志框架。

相关领域的技术人员将认识到，可以使用对所公开的实施例的许多可能的修改，而仍采用相同的基本机理和方法。为了说明的目的，以上描述参照了特定的实施例。然而，以上的说明性的讨论并不是穷尽的或者将本发明局限于所公开的精确形式。可根据以上教导进行许多修改和变化。所选择和描述的实施例用来说明本发明的原理及其实际应用，并且使本领域的技术人员能够以适合于所设想的特定用途的各种修改而最好地利用本发明和各种实施例。

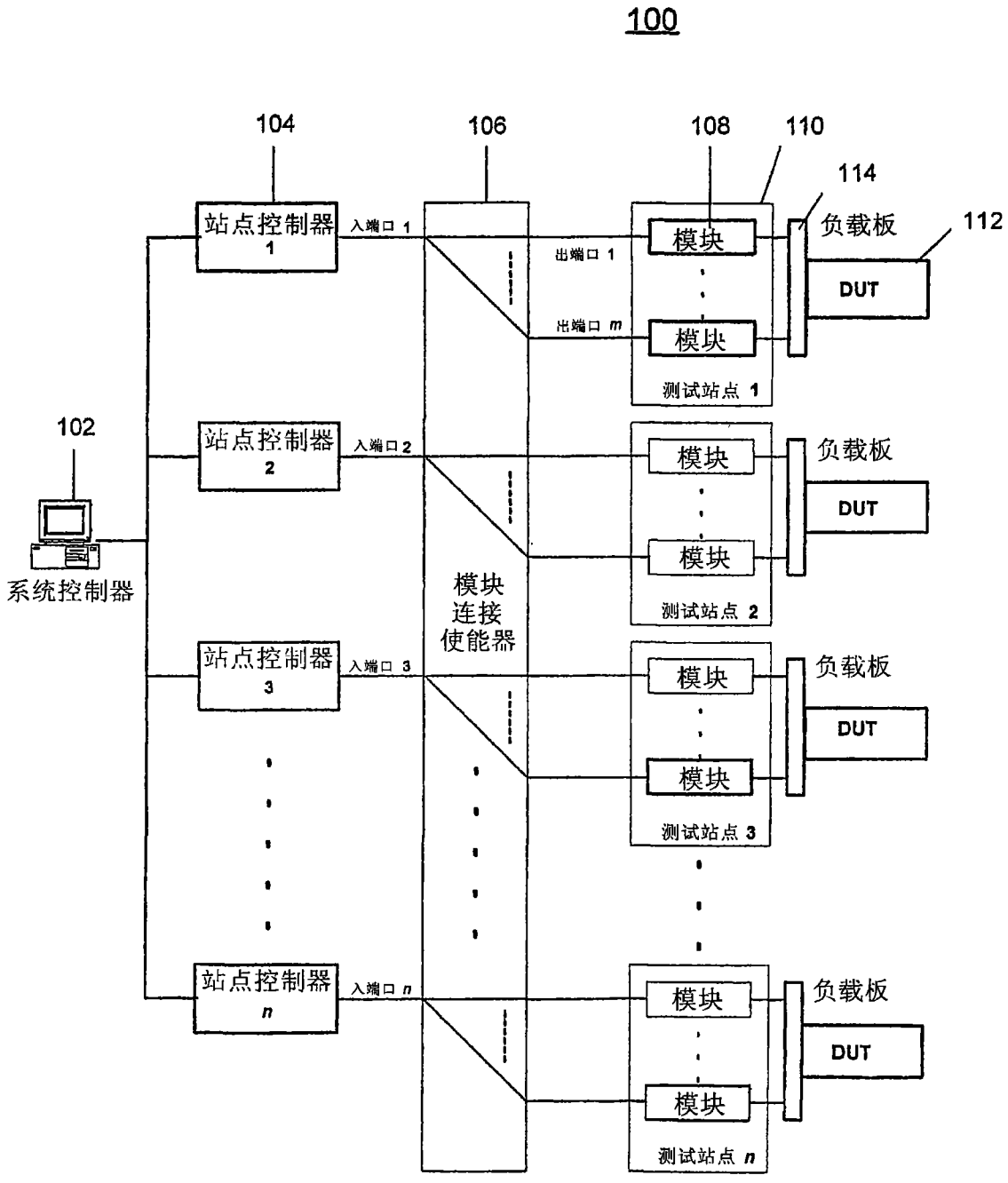

100
104
106
108
110
114
112
102
站点控制器 1
站点控制器 2
站点控制器 3
站点控制器 n
入端口 1
入端口 2
入端口 3
入端口 n
出端口 1
出端口 m
模块
负载板
DUT
测试站点 1
测试站点 2
测试站点 3
测试站点 n
模块连接使能器
系统控制器

图 1

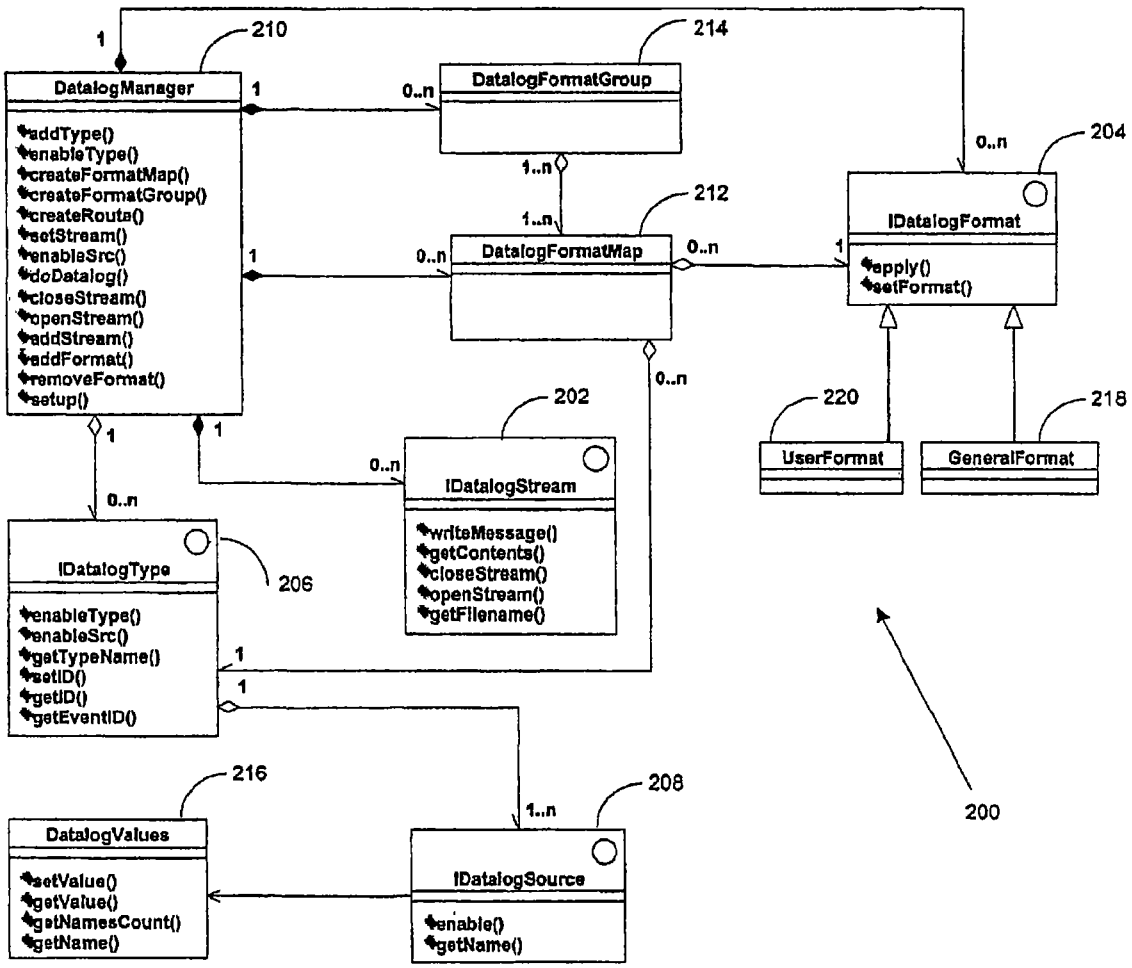

DatalogManager
addType()
enableType()
createFormatMap()
createFormatGroup()
createRoute()
setStream()
enableSrc()
doDatalog()
closeStream()
openStream()
addStream()
addFormat()
removeFormat()
setup()
210
DatalogFormatGroup
214
DatalogFormatMap
212
IDatalogFormat
apply()
setFormat()
204
IDatalogStream
writeMessage()
getContents()
closeStream()
openStream()
getFilename()
202
IDatalogType
enableType()
enableSrc()
getTypeName()
setID()
getID()
getEventID()
206
UserFormat
220
GeneralFormat
218
DatalogValues
setValue()
getValue()
getNamesCount()
getName()
216
IDatalogSource
enable()
getName()
208
200
1
0..n
1..n

图 2

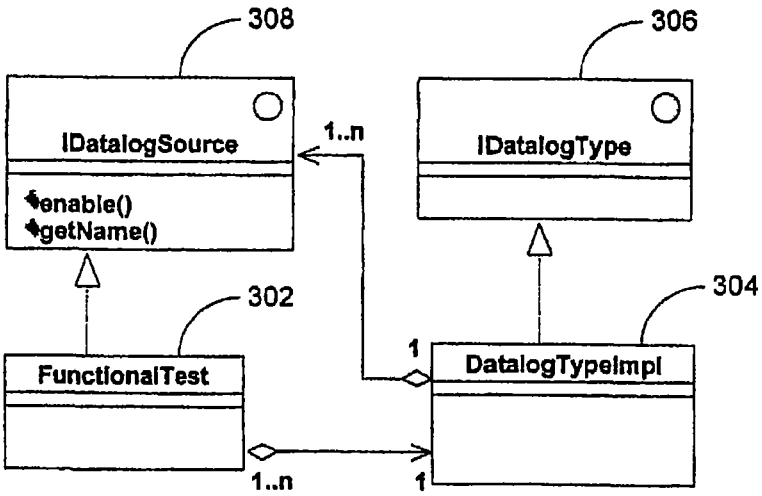

308
306
IDatalogSource
enable()
getName()
1..n
IDatalogType
302
304
1
FunctionalTest
DatalogTypeImpl
1..n
1

图 3

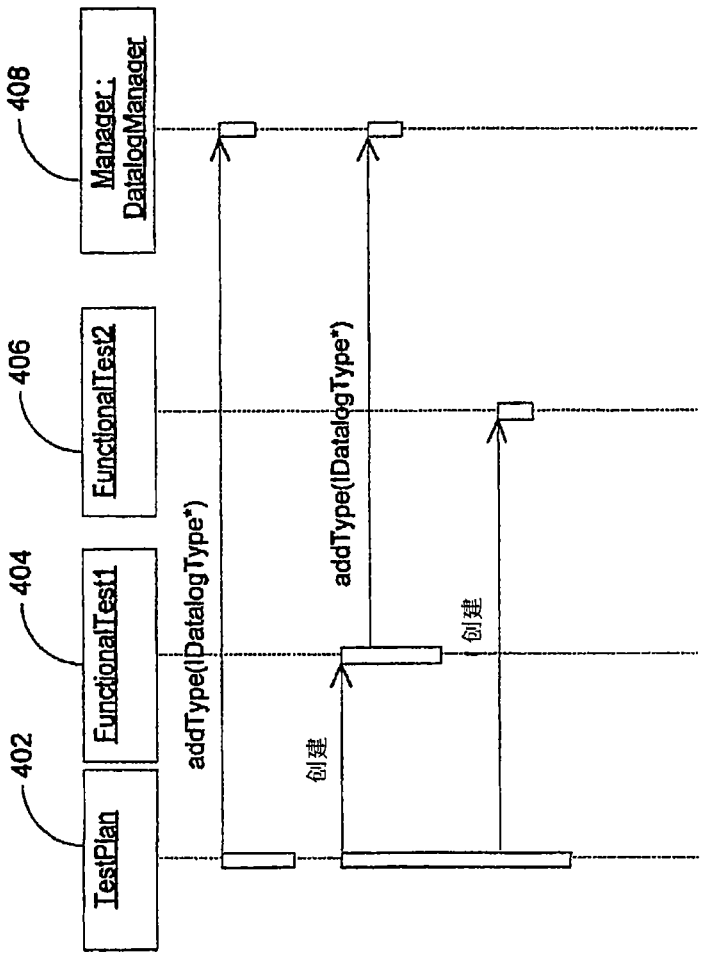

408
Manager : DatalogManager
406
FunctionalTest2
404
FunctionalTest1
402
TestPlan
addType(IDatalogType*)
addType(IDatalogType*)
创建
创建

图 4

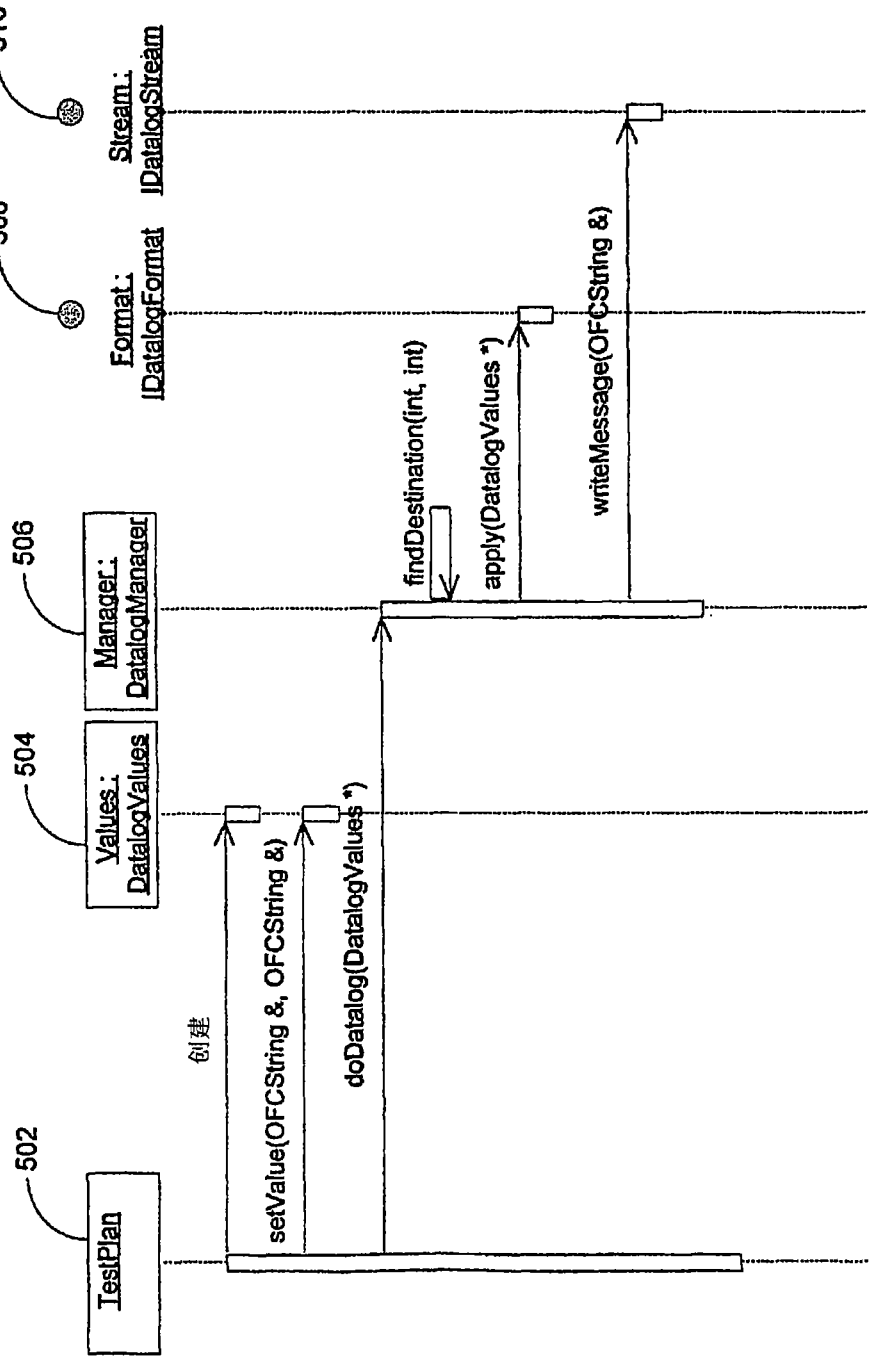

502
TestPlan
504
Values : DatalogValues
506
Manager : DatalogManager
508
Format : IDatalogFormat
510
Stream : IDatalogStream
创建
setValue(OFCString &, OFCString &)
doDatalog(DatalogValues *)
findDestination(int, int)
apply(DatalogValues *)
writeMessage(OFCString &)

图 5

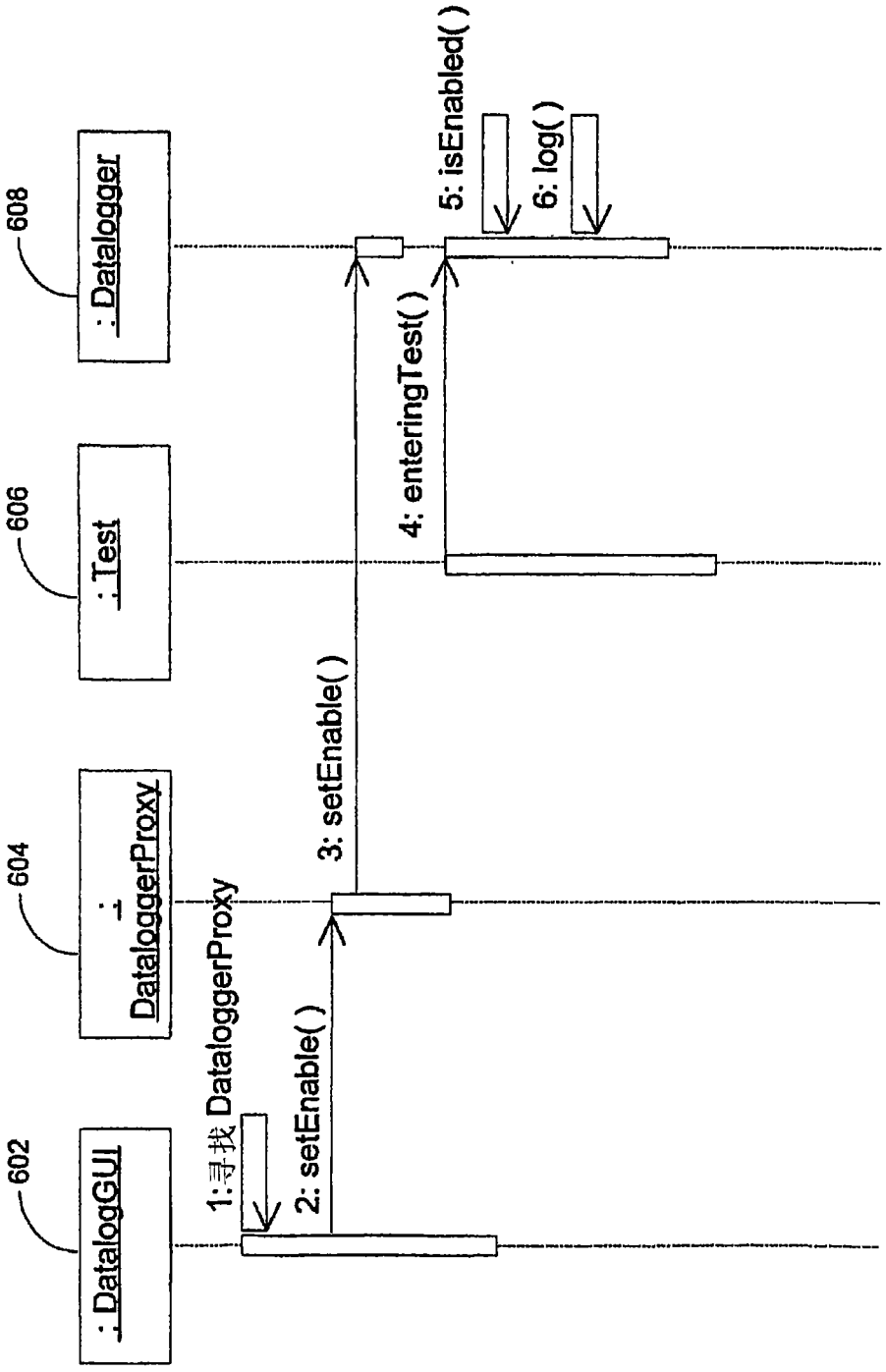

602
:DatalogGUI
604
:DataloggerProxy
606
:Test
608
:Datalogger
1:寻找 DataloggerProxy
2: setEnable( )
3: setEnable( )
4: enteringTest( )
5: isEnabled( )
6: log( )

图 6

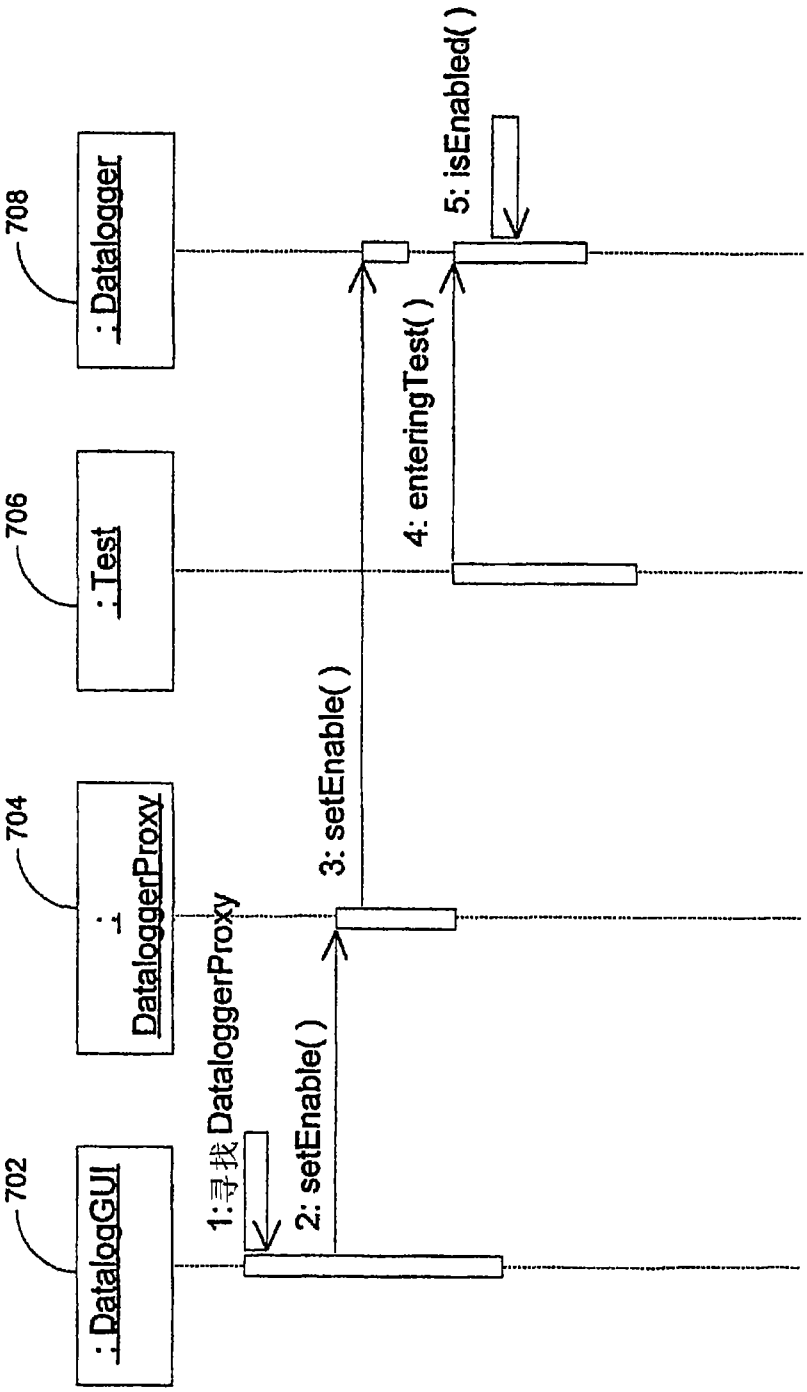

:DatalogGUI
:DataloggerProxy
:Test
:Datalogger
702
704
706
708
1:寻找 DataloggerProxy
2: setEnable( )
3: setEnable( )
4: enteringTest( )
5: isEnabled( )

图 7

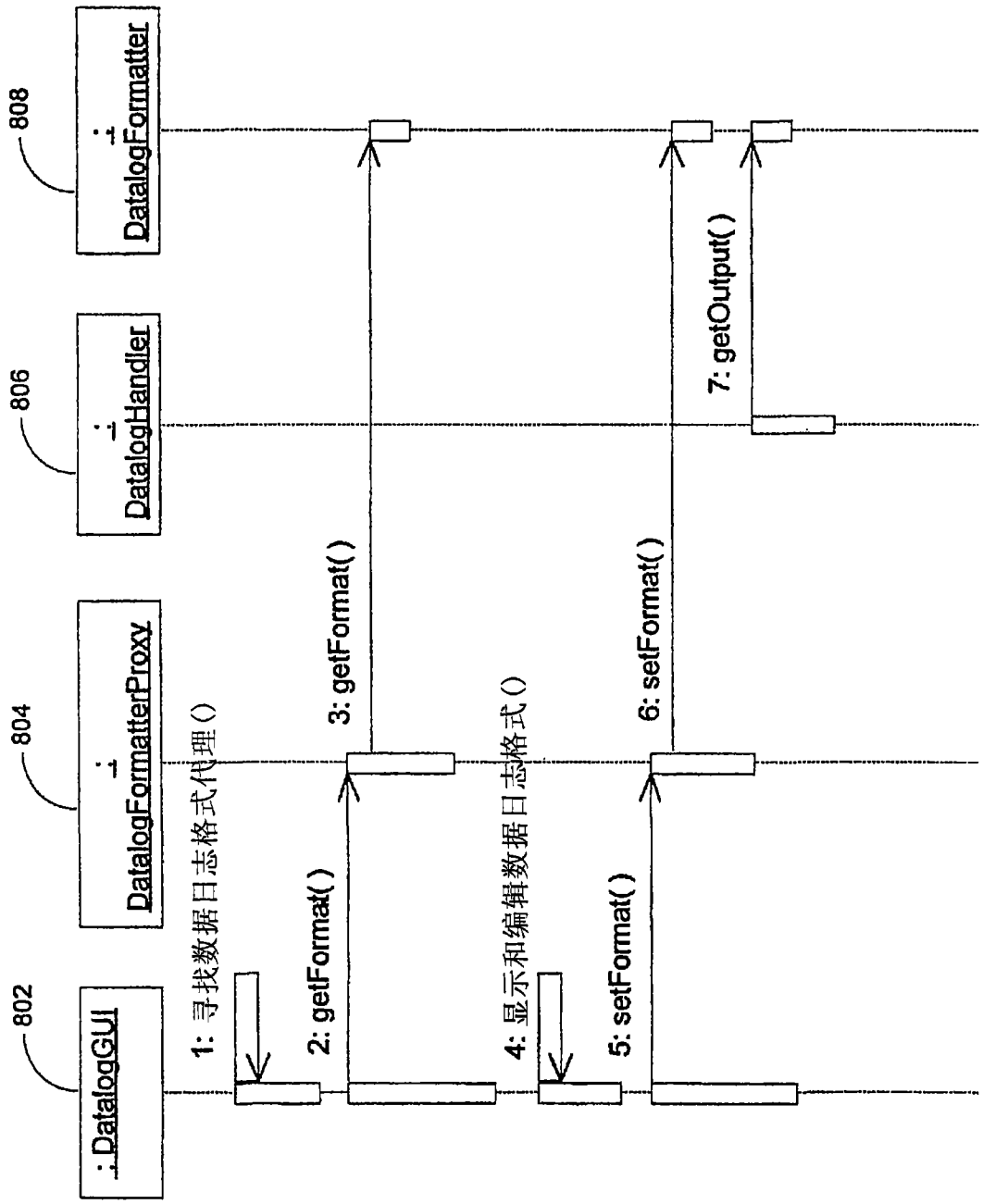

802
:DatalogGUI
804
:DatalogFormatterProxy
806
:DatalogHandler
808
:DatalogFormatter
1: 寻找数据日志格式代理()
2: getFormat()
3: getFormat()
4: 显示和编辑数据日志格式()
5: setFormat()
6: setFormat()
7: getOutput()

图 8

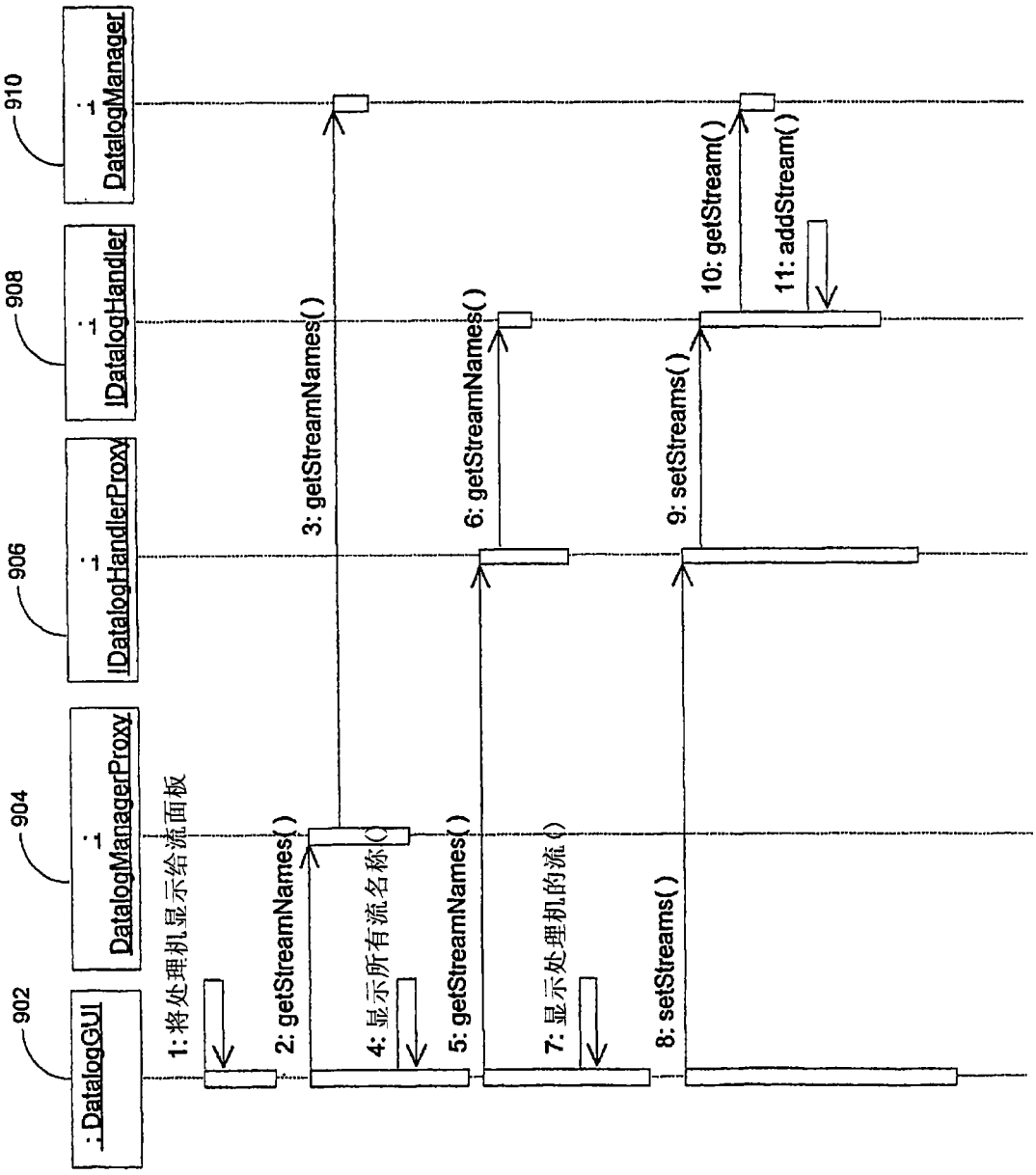

902
:DatalogGUI
904
:DatalogManagerProxy
906
:IDatalogHandlerProxy
908
:IDatalogHandler
910
:DatalogManager
1: 将处理机显示给流面板
2: getStreamNames( )
3: getStreamNames( )
4: 显示所有流名称(
5: getStreamNames( )
6: getStreamNames( )
7: 显示处理机的流 (
8: setStreams( )
9: setStreams( )
10: getStream( )
11: addStream( )

图 9

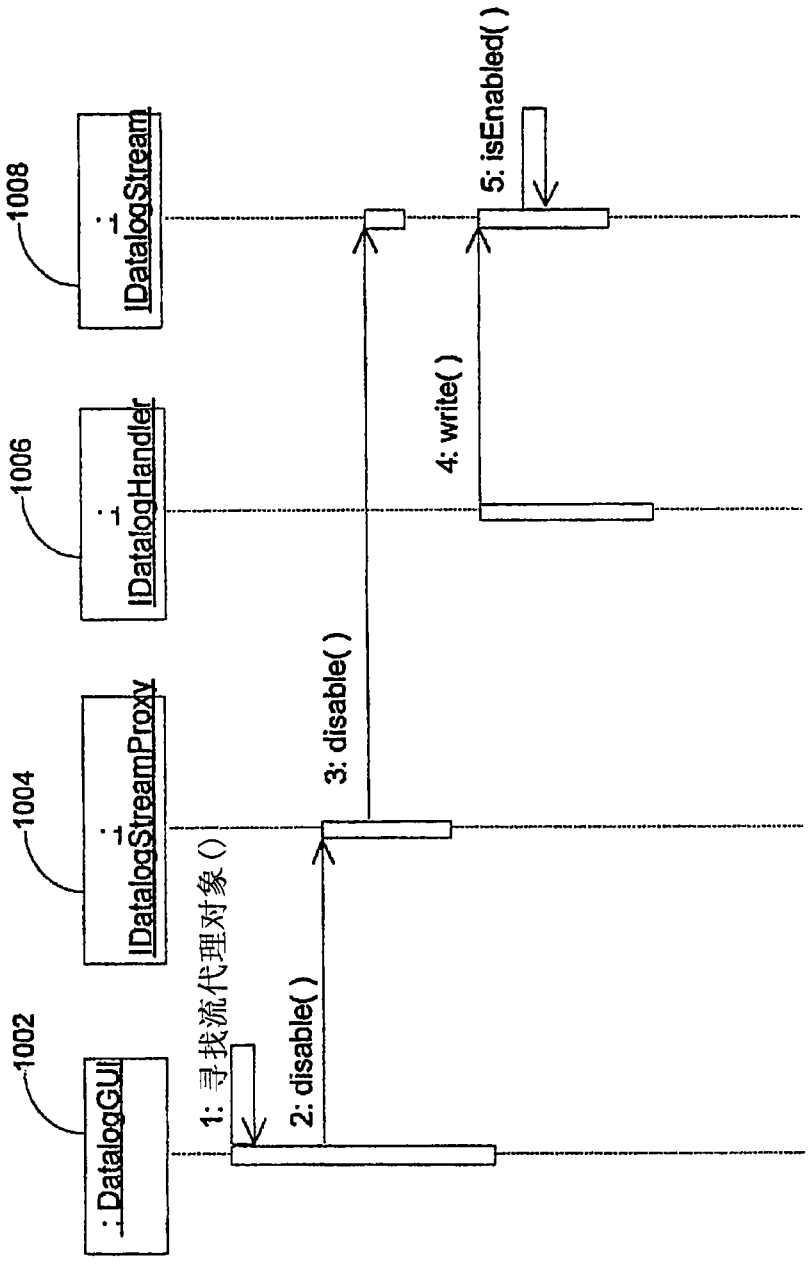

1002
:DatalogGUI
1004
:IDatalogStreamProxy
1006
:IDatalogHandler
1008
:IDatalogStream
1: 寻找流代理对象()
2: disable()
3: disable()
4: write()
5: isEnabled()

图10

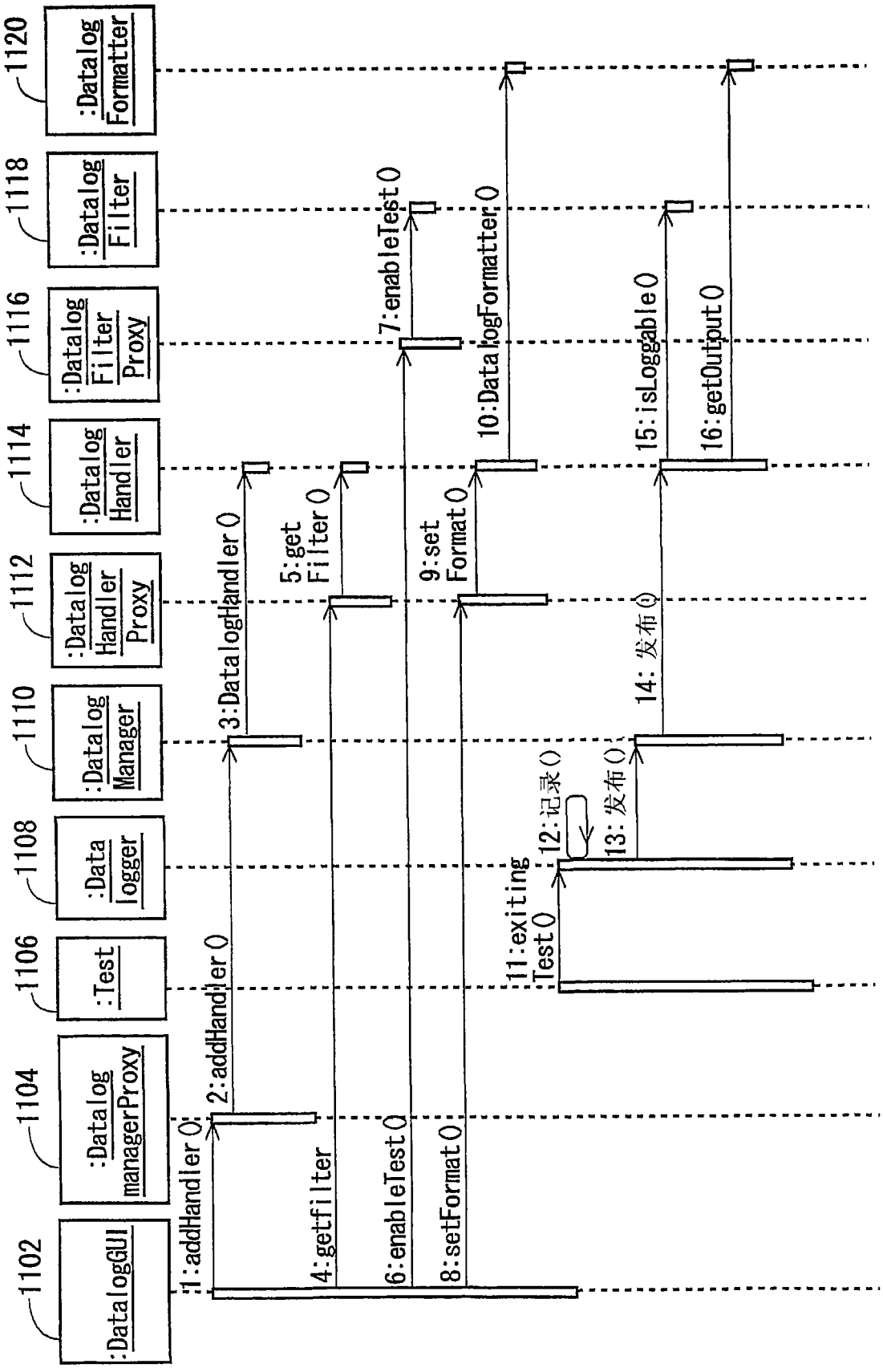

:DatalogGUI
1102
:Datalog managerProxy
1104
:Test
1106
:Data logger
1108
:Datalog Manager
1110
:Datalog Handler Proxy
1112
:Datalog Handler
1114
:Datalog Filter Proxy
1116
:Datalog Filter
1118
:Datalog Formatter
1120
1:addHandler()
2:addHandler()
3:DatalogHandler()
4:getFilter
5:get Filter()
6:enableTest()
7:enableTest()
8:setFormat()
9:set Format()
10:DatalogFormatter()
11:exiting Test()
12:记录()
13:发布()
14:发布()
15:isLoggable()
16:getOutput()

图11