(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 101031882 B
(45)授权公告日 2010.09.08

(21)申请号 200580026780.6

(22)申请日 2005.06.08

(30)优先权数据
60/577,971 2004.06.08 US

(85)PCT申请进入国家阶段日
2007.02.07

(86)PCT申请的申请数据
PCT/US2005/020367 2005.06.08

(87)PCT申请的公布数据
WO2005/121959 EN 2005.12.22

(73)专利权人 达尔特设备互操作有限公司
地址 美国加利福尼亚州

(72)发明人 丹尼尔·伊洛斯基
布鲁斯·伯恩斯坦
理查德·米拉贝拉
沃尔夫冈·皮埃博 雷蒙德·西德尼
理查德·泰伯瑞 迈克尔·温奥科

(74)专利代理机构 北京安信方达知识产权代理有限公司 11262
代理人 霍育栋 郑霞

(51)Int.Cl.
G06F 9/46(2006.01)

(56)对比文件
CN 1237734 A,1999.12.08, 说明书第1页第4行-第2页第13行, 第8页第2-31行, 第13页第7-21行, 第14页第20-29行、附图5,6.
US 2003/0115266 A1,2003.06.19, 说明书第0006-0022段、附图2.
全文.
全文.
US 2001/0042123 A1,2001.11.15, 说明书第0011-0048段、附图1.
全文.
KVALVAAG E L ET AL.Facility provision using mobile agents.DATABASE AND EXPERT SYSTEMS APPLICATIONS, 2001. PROCEEDINGS. 12TH.2001,696-700.
INTERNATIONAL WORKSHOP ON 3-7 SEP 2001,ISBN: 0-7695-1230-5.

审查员 毛习文

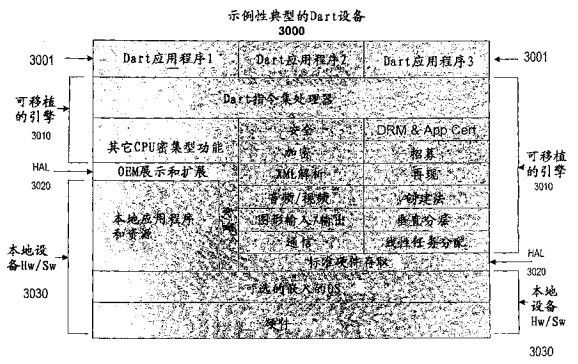
权利要求书 9 页 说明书 86 页 附图 32 页

(54)发明名称

用于通用设备互操作性平台的设备组招募和内容再现的体系结构、装置和方法

(57)摘要

提供了用于在具有相似或不同特征的设备之间提供通信并促进它们之间的无缝互操作性的系统、设备、方法,以及计算机程序和计算机程序产品。提供了用于在相似或不同的永久或间断连接的电子设备上共享内容、应用程序、资源和控制的计算机程序软件和方法以及系统和设备。提供了用于在框架内提供通信和/或互操作的设备、系统、应用程序,等等。提供了招募互操作性方法、再现程序适配和互操作性分段方法、互操作性源、互操作性框架、互操作性工具、互操作性格式、互操作性运行时间、线性任务分配、垂直分层、应用程序驱动的电源管理、互操作性应用程序驱动的错误恢复、互操作性指令集、创造法、互操作性引擎、互操作性设备授权、互操作性安全模型、社会同步互操作性方法、社会安全互操作性模型、互操作性设备、互操作性平台、虚指针,以及Dart特定形式或这些的执行。



CN 101031882 B

1. 一种用于招募设备组的方法，所述方法包括以下步骤：

(a) 通过至少一个通信链接，从初始设备将可操作地寻找具有需要的资源或能力的设备的检查程序发送到不同于所述初始设备的至少一个可到达设备，所述检查程序包括以可执行的形式编码的检查程序指令，所述可执行的形式是所述初始设备和所述检查程序将到达的所述可到达设备所共用的；

(b) 接收并在每个所述可到达设备上执行所接收到的所述检查程序，以确定所述可到达设备是否存在所述初始设备需要的至少一个资源或能力；

(c) 至少当所述可到达设备有权使用被识别为所述初始设备需要的资源或能力时，向所述初始设备发送应答；

(d) 直接或间接地通过通信链接，接收来自每个所述可到达设备的所述应答；

(e) 通过在所述初始设备上执行的程序，分析所接收的来自所有发出响应的可到达设备的应答，以确定识别所述初始设备和所述发出响应的可到达设备的能力和资源的组合的利用计划，以最佳地实现软件应用程序的目的；以及

(f) 根据所述被确定的利用计划，通过在所述初始设备上执行的应用程序，将可执行代码、数据、内容和 / 或 Dart 中的至少一个分配到被识别为具有需要的资源或能力的每个所述可到达设备中的至少一个上。

2. 如权利要求 1 所述的方法，其进一步包括通过所述至少一个可到达设备接收所述分配的可执行代码、数据、内容和 / 或 Dart 中至少之一。

3. 如权利要求 1 所述的方法，其中所述方法进一步包括：

与所述至少一个可到达设备进行互操作以实现所述初始设备的应用程序的目的的至少一部分，其中所述可执行代码、数据、内容和 Dart 中至少之一被分配到所述至少一个可到达设备。

4. 如权利要求 1 所述的方法，其中所述方法进一步包括：

每个可到达设备作为第二初始设备，在所述初始设备和可到达设备上传播可执行代码、数据、内容和 / 或 Dart，进一步又按照需要递推地传播到以前到达的和编组的设备可以到达的其它设备上，以按照需要或要求将所述应用程序扩展到其它可到达设备上，直到达到实现所述初始设备的应用程序的目的所需要的设备和资源或能力的所有的或预定标准为止，以有效地形成更大的完整的设备组；以及

根据所述初始设备应用程序的要求和需要，从所述初始设备和可到达设备组并在其之间分配可执行代码、数据、内容和 / 或 Dart，以执行所述要求的或需要的操作和资源存取，以通过执行代码和交换实现和 / 或协调所述操作所需要的任何可执行代码、数据、内容和 / 或 Dart，来实现所述初始设备的应用程序的所述目的，其中所述操作在所述设备组上执行以实现初始应用的所述目的。

5. 如权利要求 4 所述的方法，其中所述每个可到达设备作为第二初始设备而在所述初始设备和可到达设备上传播可执行代码、数据、内容和 / 或 Dart 的步骤进一步包括传播存在于所述可到达设备上的应用程序代码、数据、内容和 / 或 Dart。

6. 如权利要求 1 所述的方法，其中所述初始设备直接从最初到达的设备接收所述应答，所述最初到达的设备是所述初始设备向其发送所述招募消息的设备，或者所述初始设备间接地以串行或递推的方式通过一个或多个中间可到达设备从另一个可到达设备接收

所述应答。

7. 如权利要求 1 所述的方法，其中当所述可到达设备不能使用被确定为所述初始设备需要的资源或能力时，也向所述初始设备发送所述应答。

8. 如权利要求 1 所述的方法，其中到所述初始设备的所述应答是简单的参数，所述参数标识所述可到达设备本身拥有或可以使用被确定为所述初始设备需要的所述资源或能力，或标识所述可到达设备本身不拥有或不能使用被确定为所述初始设备需要的所述资源或能力。

9. 如权利要求 1 所述的方法，其中到所述初始设备的所述应答是“真”或“假”，所述“真”标识所述可到达设备本身拥有或可以使用被确定为所述初始设备需要的所述资源或能力，所述“假”标识所述可到达设备本身不拥有或不能使用被确定为所述初始设备需要的所述资源或能力。

10. 如权利要求 1 所述的方法，其中到所述初始设备的所述应答是逻辑“1”或逻辑“0”，所述逻辑“1”标识所述可到达设备本身拥有或可以使用被确定为所述初始设备需要的所述资源或能力，所述逻辑“0”标识所述可到达设备本身不拥有或不能使用被确定为所述初始设备需要的所述资源或能力。

11. 如权利要求 1 所述的方法，其中到所述初始设备的所述应答包括 DartEvent、Dart 的一部分、数据、内容、可执行程序、一个 Dart、多个 Dart，以及上述任何组合中之一。

12. 如权利要求 1 所述的方法，其中到所述初始设备的所述应答包括通过通信协议识别对于所述初始设备的特定的可到达设备的资源和 / 或能力的返回数据或内容。

13. 如权利要求 1 所述的方法，其中所述检查程序包括检查所述初始设备需要的至少一个特别确定的资源或能力，以实现至少部分地在所述初始设备上执行的所述应用程序任务的所述目的的指令。

14. 如权利要求 1 所述的方法，其中所述方法至少部分地被编码为计算机程序产品的软件应用程序执行，其中所述软件应用程序产品在设备组上招募并有效地运行应用程序。

15. 如权利要求 1 所述的方法，其中所述方法有效地连接并接着允许控制进行招募的和被招募的设备的操作，如同这些设备是具有所有所述进行招募的和被招募的设备的组合资源的一个设备。

16. 如权利要求 1 所述的方法，其中所述应答包括下述中任何之一 ：没有任何应答、数据或内容应答、任何数字化编码的信息、一个或多个程序、所述可到达设备将是有用的显示、返回的事件、通过文件净载荷包括任何数量的数据或数据集的应答事件、应答程序、Dart、包括文本名称和设备描述从而人可以从所述初始设备的菜单上选择使用哪个设备的应答事件、存在于所述初始设备上或者能够在所述初始设备上生成的一组可执行代码、数据以及内容的至少一个示例的特定包的标识符、最适合于在所述可到达设备上运行的再现程序或再现程序集，或者上述的任何组合。

17. 如权利要求 1 所述的方法，其中所述至少一个资源或能力是从包括下述的组中选择的 ：(i) 可用的资源或特别需要的资源 ；(ii) 可用的能力或特别需要的能力 ；(iii) 所述可到达设备的一个或多个相互关联的资源和能力组，或所述可到达设备为实现所述应用程序的目的可用的那些资源和 / 或能力 ；以及 (iv) 这些的任何组合。

18. 如权利要求 1 所述的方法，其中所述资源或能力包括从包括下述的资源或能力组

中选择的至少一个确定的能力：

确定的数据处理软件、确定的信息处理软件、确定的计算软件、确定的图片处理软件、确定的通信软件、确定的通信硬件、确定的介质、确定的数据集、确定的内容、确定的程序或多个程序、确定的配置信息、确定的图形加速硬件或软件、确定的存储介质而不管其是暂时的或不是暂时的、确定的打印能力、确定的传真能力、确定的扫描能力、确定的用户接口设备，而无论所述用户接口设备为输入或为输出或者既是输入又是输出、对其它设备的所述资源的存取，由于所述资源，所述设备能够进行通信并且所述其它设备能够在不断的链条中进行通信、以及这些的两个或多个的任何组合。

19. 如权利要求 1 所述的方法，其中以共同可执行形式的所述检查程序包括由 Dart 兼容的指令集或任何其它互操作性指令集形成的至少一个检查程序，其中所述 Dart 兼容的指令集被包括在 Dart 指令兼容的引擎中。

20. 如权利要求 1 所述的方法，其中所述至少一个通信链接包括任何数量的通信链接、信道，和 / 或协议，所述协议包括任何数量或组的相同或不同的通信协议，而不管这些通信协议是有线的或无线的，是永久可用的或是间断可用的。

21. 如权利要求 1 所述的方法，其中相同或不同的通信链接、信道，以及协议被确定的硬件抽象层工具支持，其中所述硬件抽象层工具是运行在任何两个或多个通信设备上的播放器的部分。

22. 如权利要求 21 所述的方法，其中所述确定的硬件抽象层工具包括作为 Dart 指令兼容的引擎组件的 Dart 硬件抽象层工具。

23. 如权利要求 1 所述的方法，其中所述至少一个通信链接和通信协议被用于发送运行程序类型的事件，所述事件的事件标识符引用识别在所述可到达设备上运行的程序的文件。

24. 如权利要求 23 所述的方法，其中所述事件包括 DartEvents，所述运行程序类型事件包括 Dart RUN_PROCEDURE 类型事件。

25. 如权利要求 1 所述的方法，其中引用识别将在所述可到达设备上运行的所述程序的文件的事件标识符包括指向包括将在所述可到达设备上运行的所述程序的图像的文件的所述事件的文件标识符。

26. 如权利要求 25 所述的方法，其中所述文件包括符合 DartFormat 的 Dart 兼容的文件，所述程序的所述图像包括 Dart Procedure 的二进制数据图像。

27. 如权利要求 1 所述的方法，其中所述检查程序包括 DartProcedures 或完整的 Darts。

28. 如权利要求 1 所述的方法，其中将所述检查程序作为与事件相关联的文件发送，可到达设备将所述检查程序作为与所述事件相关联的所述文件接收，使得所述检查程序开始在所述可到达设备上执行。

29. 如权利要求 1 所述的方法，其中所述检查程序包括 DartProcedure。

30. 如权利要求 1 所述的方法，其中通过利用指令集轮廓指令确定包括所述可到达设备的基本资源和能力的资源和能力。

31. 如权利要求 30 所述的方法，其中所述指令集轮廓指令包括 Dart 顺应的指令集的 Dart 顺应轮廓指令。

32. 如权利要求 1 所述的方法，其中在每个可到达设备中执行的所述检查程序根据再现程序确定规则确定包括应用程序的初始 Dart 的最佳再现程序，以被发送到每个特定的可到达设备并将所述确定的最佳再现程序的标识符作为返回的数据的一部分发送回来。

33. 如权利要求 32 所述的方法，其中所述再现程序确定规则被包括在至少一个程序中，所述至少一个程序适于执行任何复杂度的任何需要的计算并通过轮廓指令存取任何需要的轮廓信息，以确定所述可到达设备的资源、能力，和 / 或状态。

34. 如权利要求 32 所述的方法，其中所述检查程序执行过程参考定义再现程序的顺序的规则，从多个再现程序中确定最佳再现程序，并检查每个可到达设备以确定是否以预定义的再现程序优先顺序满足所述多个再现程序的每一个的所有要求，直到发现排序的所述多个再现程序中的第一再现程序满足所有所述再现程序的要求为止。

35. 如权利要求 1 所述的方法，其中所述检查程序通过所述通信链接，使用知道的通信协议，将 Darts、程序、数据或内容返回到所述初始设备。

36. 如权利要求 1 所述的方法，其中所述应答包括返回的程序、数据或内容和下述中任何一个或组合中的至少之一 :完整的 Darts、DartParts、DartProcedures、或 DartEvents，可以将其中的一个或任何组合与相关联的事件文件一起或不与相关联的事件文件一起返回。

37. 如权利要求 1 所述的方法，其中所述应答包括返回的程序、数据或内容和 / 或 Dart 中至少之一。

38. 如权利要求 37 所述的方法，其中所述应答进一步包括表示发生错误的应答代码、确定或者发生特定的错误或者发生非特定的错误的错误代码。

39. 如权利要求 38 所述的方法，其中所述错误代码包括在改正或传输所述特定的错误和 / 或所述特定的错误的性质中有用的信息。

40. 如权利要求 1 所述的方法，其中所述可执行代码是下述中至少之一 :Dart、DartProcedure，或可以在所述初始设备或所述初始设备能够使用的设备上执行的任何形式的任何程序，用于启动转移或执行程序和利用所述执行的结果。

41. 如权利要求 1 所述的方法，其中所述可执行代码包括 Dart、DartProcedure，或可以在所述可到达设备上执行或另外地向所述可到达设备传递信息的另一种程序格式，而不管所述程序格式是否利用将在所述可到达设备上执行的 DartInstructionSet。

42. 如权利要求 1 所述的方法，其中所述被招募的设备组可以按照需要，在有效期内或在为执行所述应用程序定义的时间段内，动态地扩展所述组以包括其它可到达设备，或减少所述设备组以按照需要排除可到达设备。

43. 如权利要求 1 所述的方法，其中通过发送 Darts、DartProcedures、数据、内容，或其它信息中至少之一、或这些的任何组合，实现所述分配，所述 Darts、DartProcedures、数据、内容，或其它信息被封装为 Dart 事件的一部分，而不管所述事件中的字段引用的信息是否随所述事件或作为所述事件的一部分被发送。

44. 如权利要求 1 所述的方法，其中根据初始应用的需要从所述设备组和在所述设备组中分配的所述代码、数据，以及内容，执行要求的操作和资源存取，以通过执行代码来实现所述初始应用的目的。

45. 如权利要求 1 所述的方法，其中根据所述初始应用的需要从所述设备组和在所述

设备组中分配的所述代码、数据，以及内容，执行要求的操作和资源存取，以通过执行代码和交换实现或协调所述操作所需要的任何其它或不同的代码、数据和内容，来实现所述初始应用的目的，其中所述操作将在所述设备组上执行以进一步实现所述初始应用的目的。

46. 如权利要求 1 所述的方法，其中所述应用程序被包括在单个二进制图像中，所述二进制图像包括作为招募程序的一部分被分配到所有所述可到达设备的代码。

47. 如权利要求 1 所述的方法，其中所述方法进一步包括同步、串行化，以及协调所述设备组的活动，所述同步、串行化，以及协调是通过单独地传递或交换事件或 DartEvents 而被全部地或部分地实现的。

48. 如权利要求 1 所述的方法，其中所述方法进一步包括同步、串行化，以及协调所述设备组的活动，所述同步、串行化，以及协调是通过传递或交换事件或 DartEvents 并且传递或交换与所述事件或 DartEvents 相关联的文件而被全部地或部分地实现的。

49. 如权利要求 47 或权利要求 48 所述的方法，其中所述事件引用至少一个文件，从而所述事件通过该引用有效地包括并合并具有任何复杂度的文件图像的文件或多个文件，这些文件图像是与所述事件结构内容一同进行传输的。

50. 如权利要求 1 所述的方法，进一步包括在进行互操作的和进行通信的初始设备和被招募的设备之间单独地传递或交换 DartEvents 或事件，通过 DartEvents 结构中引用的文件标识符，所述事件有效地包括任何复杂度的文件或其它数据或数据结构，并且当进行引用时文件图像总是与事件结构内容一同进行传输。

51. 如权利要求 1 所述的方法，进一步包括在进行互操作的和进行通信的初始设备和被招募的设备之间，与 DartEvents 或事件相关的文件一起，传递或交换 DartEvents 或事件，通过 DartEvents 结构中引用的文件标识符，所述事件有效地包括任何复杂度的文件或其它数据或数据结构，并且当进行引用时文件图像总是与事件结构内容一同进行传输。

52. 如权利要求 47 所述的方法，其中在任何数量的编组设备的事件队列中，DartFramework、DartRuntime，和 / 或 DartEngine 自动地备份和 / 或同步所述 DartEvents 或事件，从而被确定为自动的同步和被 Dart 放置在事件队列中的 DartEvents 或事件以自动串行化和同步的方式出现在任何数量的编组设备的所述事件队列中。

53. 如权利要求 47 所述的方法，其中在任何数量的编组设备的事件队列中，非 DartPlatform 专用的事件驱动的工具自动地备份和 / 或同步所述 DartEvents 或事件，从而被确定为自动的同步和被 Dart 放置在事件队列中的 DartEvents 或事件以自动串行化和同步的方式出现在任何数量的编组设备的所述事件队列中。

54. 如权利要求 52 或 53 所述的方法，其中所述自动串行化和同步是通过用于自动串行化和同步事件驱动的协作的应用程序、功能或在多个协作的设备上运行的再现程序的程序实现的，所述协作的设备包括初始设备，所述方法包括 ：

通过用于需要的每个设备间协作功能的应用程序，在所述初始设备上例示连接管理器对象示例 ；

通过所述应用程序按照程序将在所有协作的设备上同步的事件类型列表传输到所述连接管理器 ；

在每个设备上建立具有一个连接管理器的协作设备组，并共享相同的同步事件类型列表 ；以及

由所述连接管理器维护确定编组设备和将被同步的事件类型的会话列表；并且由所述连接管理器检查将被放在所述事件队列中的所有事件，如果被检查的特定事件是所述连接管理器从应该被同步的会话列表得知的事件类型，则将所述特定事件标记为将与其它会话同步的事件，并将所述特定事件放置在所述连接管理器中列出的设备的事件队列。

55. 如权利要求 54 所述的方法，其中通过直到接收到这种确认信息，即所述事件被任何一个设备直接发送到的所有协作设备事件队列已经成功地被放置在所述协作设备的事件队列中，才允许事件被放置在所述任何一个设备的事件队列中，来串行化将被所述事件驱动的协作应用程序功能或再现程序中任何一个放置在协作设备的事件队列中的所有事件。

56. 如权利要求 55 所述的方法，其中通过直到接收到这种确认信息，即所述事件被任何一个接收设备发送到的所有协作设备事件队列已经成功地被放置在所述协作设备的事件队列中，才允许事件被放置在所述接收的任何一个设备的事件队列中，来串行化将被所述事件驱动的协作应用程序功能或再现程序中任何一个接收的放置在协作设备的事件队列中的所有事件。

57. 如权利要求 55 所述的方法，其中无论设备是直接进行通信或间接地通过直接通信的设备链进行通信，任何数量的协作设备事件队列在所有协作设备上建立一个排队事件的串行化系统。

58. 如权利要求 54 所述的方法，其中无论设备是直接进行通信或间接地通过直接通信的设备链进行通信，任何数量的协作设备事件队列在所有协作设备上建立一个排队事件的串行化系统。

59. 如权利要求 54 所述的方法，其中将被放置在来自两个或多个协作设备的协作设备队列中的事件被系统同步为所述协作设备队列中的一个串行化事件，在所述系统中仅允许一个主设备放置将被同步的所述事件类型列表中的类型的事件，从而在所有协作设备上串行化所有这种事件。

60. 如权利要求 59 所述的方法，其中通知所述主设备通过包括所述主设备将指定的事件放入所有协作设备的队列中所需要的信息的主请求事件类型事件，代表其它协作设备产生的事件。

61. 如权利要求 59 所述的方法，其中被所述协作设备组中的另一个设备招募到所述协作设备组中的每个设备认为其相对的主设备为招募其的设备。

62. 如权利要求 58 所述的方法，其中被所述协作设备组中的另一个设备招募到所述协作设备组中的每个设备认为其相对的主设备为招募其的设备。

63. 如权利要求 61 所述的方法，其中将主请求事件类型事件放入相对的主设备队列中将使得所述主请求事件类型事件被从设备传播到相对的主设备上，直到没有相对主设备的初始主设备接着使用所述主设备需要的信息形成事件，并将指定的事件放入所有协作设备的队列中。

64. 如权利要求 59 所述的方法，其中通过向所述协作设备的同步的和 / 或串行化的队列发送改变主设备类型事件，来改变指定的主设备，其中所述改变主设备类型事件处于串行化事件列表中，所述改变主设备类型事件以同步的串行化的方式通知所有设备新的主设备将代替当前的主设备。

65. 如权利要求 61 所述的方法，其中主请求事件传播通过所述协作设备队列，直到新的主设备处理所述事件。

66. 如权利要求 62 所述的方法，其中主请求事件传播通过所述协作设备队列，直到新的主设备处理所述事件。

67. 如权利要求 60 所述的方法，其中如果存在这种文件引用，则被确定为这种事件，即被指定被所述主请求事件类型事件发送的事件的一部分的可选的文件被保存在每个进行传播的设备上，并具有允许所述事件被重新与将被所述主设备发送的事件相关联，就如同其被作为所述主设备发送的事件的一部分发送一样的标识符，作为所述主设备处理的主请求事件类型的结果，这就减少了可能另外地需要作为每个事件一部分发送的信息量。

68. 一种用于招募设备组的系统，所述系统包括：

用于通过至少一个通信链接，从初始设备将可操作地寻找具有需要的资源或能力的设备的检查程序发送到不同于所述初始设备的至少一个可到达设备的装置，其中，所述检查程序包括以可执行的形式编码的检查程序指令，所述可执行的形式是所述初始设备和所述检查程序将到达的所述可到达设备所共用的；

用于接收并在每个所述可到达设备上执行所接收到的所述检查程序，以确定所述可到达设备是否存在所述初始设备需要的至少一个资源或能力的装置；

用于至少当所述可到达设备有权使用被识别为所述初始设备需要的资源或能力时，向所述初始设备发送应答的装置；

用于直接或间接地通过通信链接，接收来自每个所述可到达设备的所述应答的装置；

用于通过在所述初始设备上执行的程序，分析所接收的来自所有发出响应的可到达设备的所述应答，以确定识别所述初始设备和所述发出响应的可到达设备的能力和资源的组合的利用计划，以最佳地实现软件应用程序的目的的装置；以及

用于根据所述被确定的利用计划，通过在所述初始设备上执行的应用程序，将可执行代码、数据、内容和 / 或 Dart 中的至少一个分配到被识别为具有需要的资源或能力的每个所述可到达设备中的至少一个上的装置。

69. 如权利要求 68 所述的系统，其进一步包括：

用于通过所述至少一个可到达设备接收所述分配的可执行代码、数据、内容和 / 或 Dart 中至少之一的装置。

70. 如权利要求 68 所述的系统，其进一步包括：

用于与所述至少一个可到达设备进行互操作以实现所述初始设备的应用程序的目的的至少一部分的装置，其中，

所述可执行代码、数据、内容和 Dart 中至少之一被分配到所述至少一个可到达设备。

71. 如权利要求 68 所述的系统，其进一步包括：

用于将每个可到达设备作为第二初始设备，在所述初始设备和可到达设备上传播可执行代码、数据、内容和 / 或 Dart，进一步又按照需要递推地传播到以前到达的和编组的设备可以到达的其它设备上，以按照需要或要求将所述应用程序扩展到其它可到达设备上，直到达到实现所述初始设备的应用程序的目的所需要的设备和资源或能力的所有的或预定标准为止，以有效地形成更大的完整的设备组的装置；以及

用于根据所述初始设备应用程序的要求和需要，从所述初始设备和可到达设备组并在

其之间分配可执行代码、数据、内容和 / 或 Dart，以执行所述要求的或需要的操作和资源存取，以通过执行代码和交换实现和 / 或协调所述操作所需要的任何可执行代码、数据、内容和 / 或 Dart，来实现所述初始设备的应用程序的所述目的的装置。

72. 如权利要求 68 所述的系统，其中当所述可到达设备不能使用被确定为所述初始设备需要的资源或能力时，所述发送应答的装置也向所述初始设备发送所述应答。

73. 如权利要求 68 所述的系统，其中所述应答是简单的参数，所述参数标识所述可到达设备本身拥有或可以使用被确定为所述初始设备需要的所述资源或能力，或标识所述可到达设备本身不拥有或不能使用被确定为所述初始设备需要的所述资源或能力。

74. 如权利要求 68 所述的系统，其中所述应答包括 DartEvent、Dart 的一部分、数据、内容、可执行程序、一个 Dart、多个 Dart，以及上述任何组合中之一。

75. 如权利要求 68 所述的系统，其中所述应答包括通过通信协议识别对于所述初始设备的特定的可到达设备的资源和 / 或能力的返回数据或内容。

76. 如权利要求 68 所述的系统，其中所述检查程序包括检查所述初始设备需要的至少一个特别确定的资源或能力，以实现至少部分地在所述初始设备上执行的所述应用程序任务的所述目的的指令。

77. 如权利要求 68 所述的系统，其中所述系统有效地连接并接着允许控制进行招募的和被招募的设备的操作，如同这些设备是具有所有所述进行招募的和被招募的设备的组合资源的一个设备。

78. 如权利要求 68 所述的系统，其中所述应答包括下述中任何之一 ：没有任何应答、数据或内容应答、任何数字化编码的信息、一个或多个程序、所述可到达设备将是有用的显示、返回的事件、通过文件净载荷包括任何数量的数据或数据集的应答事件、应答程序、Dart、包括文本名称和设备描述从而人可以从所述初始设备的菜单上选择使用哪个设备的应答事件、存在于所述初始设备上或者能够在所述初始设备上生成的一组可执行代码、数据以及内容的至少一个示例的特定包的标识符、最适合于在所述可到达设备上运行的再现程序或再现程序集，或者上述的任何组合。

79. 如权利要求 68 所述的系统，其中所述至少一个资源或能力是从包括下述的组中选择的 ：(i) 可用的资源或特别需要的资源 ；(ii) 可用的能力或特别需要的能力 ；(iii) 所述可到达设备的一个或多个相互关联的资源和能力组，或所述可到达设备为实现所述应用程序的目的可用的那些资源和 / 或能力 ；以及 (iv) 这些的任何组合。

80. 如权利要求 68 所述的系统，其中所述资源或能力包括从包括下述的资源或能力组中选择的至少一个确定的能力 ：

确定的数据处理软件、确定的信息处理软件、确定的计算软件、确定的图片处理软件、确定的通信软件、确定的通信硬件、确定的介质、确定的数据集、确定的内容、确定的程序或多个程序、确定的配置信息、确定的图形加速硬件或软件、确定的存储介质而不管其是暂时的或不是暂时的、确定的打印能力、确定的传真能力、确定的扫描能力、确定的用户接口设备，而无论所述用户接口设备为输入或为输出或者既是输入又是输出、对其它设备的所述资源的存取，由于所述资源，所述设备能够进行通信并且所述其它设备能够在不断的链条中进行通信、以及这些的两个或多个的任何组合。

81. 如权利要求 68 所述的系统，其中所述检查程序包括 DartProcedures 或完整的

Darts。

82. 如权利要求 68 所述的系统，其中所述应答包括返回的程序、数据或内容和下述中任何一个或组合中的至少之一：完整的 Darts、DartParts、DartProcedures、或 DartEvents，可以将其中的一个或任何组合与相关联的事件文件一起或不与相关联的事件文件一起返回。

83. 如权利要求 68 所述的系统，其中所述应答包括返回的程序、数据或内容和 / 或 Dart 中至少之一。

84. 一种用于运行在源设备上的软件应用程序招募设备组并将所述软件应用程序分为在所述设备组上可执行的一组分别执行的图像的方法，所述方法包括：

通过至少一个通信链接，将可操作地寻找具有需要的资源或能力的设备的检查程序发送到不同于所述源设备的至少一个可到达设备，所述检查程序包括以可执行的形式编码的检查程序指令，所述可执行的形式是所述源设备和所述检查程序将到达的所述可到达设备所共用的；

直接或间接地通过通信链接，在所述源设备上接收来自每个所述可到达设备的应答响应；

通过在所述源设备上执行的程序，分析所接收的来自所有发出响应的可到达设备的应答，以确定识别所述源设备和所述发出响应的可到达设备的能力和资源的组合的利用计划，以最佳地实现所述软件应用程序的目的；

根据所述被确定的利用计划，通过在所述源设备上执行的应用程序，将可执行代码、数据、内容，和 / 或 Dart 中的至少一个分配到被识别为具有需要的资源或能力的每个所述可到达设备中的至少一个上；以及

根据可能遇到的设备的用于实现所述应用程序所述目的的一个或多个方面的可能的资源和能力，将所述可能遇到的设备分为不同等级；

根据用于实现所述应用程序所述目的的一个或多个方面所需要的不同再现或其它运行时间要求，将可能遇到的环境或操作要求分为不同等级；

对于每个等级的设备和每个等级的环境或操作要求，为要求的可执行图像实现所述应用程序所述目的的一个或多个方面指定要求的数据、代码、内容以及其它数字化表示的资源；

对于特定的候选设备列表、其资源和能力，以及要求的或希望的环境或操作参数，生成利用计划，以选择哪些设备和相应的独立分配的可执行图像将运行在每个设备上，以用于实现所述应用程序的所述目的；以及

对于每个等级的设备和每个等级的环境或操作要求，指定所述要求的数据、代码、内容以及其它数字化表示的资源，以执行和实现所述利用计划。

用于通用设备互操作性平台的设备组招募和内容再现的体系结构、装置和方法

[0001] 相关申请

[0002] 本申请根据 35U.S.C.119(e) 要求 2004 年 6 月 8 日提交的名称为“用于有效的、低成本、无缝设备互操作性软件平台的体系结构、装置及其方法”的美国临时专利申请序列号为 60/577,971 的专利申请的优先权的利益，该申请的发明人为 Daniel IIIowsky ；该申请被全部包括在此以作参考。

[0003] 下面的共同未决的美国实用新型专利申请和 PCT 国际专利申请也是相关申请，其每一个被全部包括在此以作参考 ：

[0004] 2005 年 6 月 8 日提交的名称为“用于设备招募互操作性和组成统一的进行互操作的设备群集的方法和系统”，案号为 No.186245/US/8 的专利申请 ；

[0005] 2005 年 6 月 8 日提交的名称为“用于内容再现适配和互操作性分段模型的方法、系统及数据结构”，案号为 No.186245/US/9 的专利申请 ；

[0006] 2005 年 6 月 8 日提交的名称为“用于为可互操作的设备组规定设备互操作性源、规定再现数据和代码的方法和系统”，案号为 No.186245/US/5 的专利申请 ；

[0007] 2005 年 6 月 8 日提交的名称为“用于为可互操作的设备组建立互操作性应用程序的设备互操作性构架和方法”，案号为 No.186245/US/2 的专利申请 ；

[0008] 2005 年 6 月 8 日提交的名称为“用于将互操作性应用程序规范处理为可互操作的应用程序包的设备互操作性工具集和方法”，案号为 No.186245/US/14 的专利申请 ；

[0009] 2005 年 6 月 8 日提交的名称为“用于组合互操作性应用程序包的设备互操作性格式规则集和方法”，案号为 No.186245/US/15 的专利申请 ；

[0010] 2005 年 6 月 8 日提交的名称为“在多个分离的处理单元中建立事件串行化和同步的设备互操作性运行时间以及用于协调控制数据和操作的方法”，案号为 No.186245/US/16 的专利申请 ；

[0011] 2005 年 6 月 8 日提交的名称为“用于在多个处理单元中进行线性任务分配的方法和系统”，案号为 No.186245/US/11 的专利申请 ；

[0012] 2005 年 6 月 8 日提交的名称为“用于在处理单元中在各级之间进行垂直分层以有助于在各级之间直接传输事件结构和事件队列而不需要进行转换的方法和系统”，案号为 No.186245/US/10 的专利申请 ；

[0013] 2005 年 6 月 8 日提交的名称为“用于在间断连接的可互操作的电子设备中的应用程序驱动的电源管理的系统和方法”，案号为 No.186245/US/7 的专利申请 ；

[0014] 2005 年 6 月 8 日提交的名称为“用于在间断连接的可互操作的电子设备中的互操作性应用程序驱动的差错管理和恢复的系统和方法”，案号为 No.186245/US/17 的专利申请 ；

[0015] 2005 年 6 月 8 日提交的名称为“用于互操作性指令集的设备和方法”，案号为 No.186245/US/4 的专利申请 ；

[0016] 2005 年 6 月 8 日提交的名称为“用于定制地程序化地动态地生成互操作性内容的

方法和系统”，案号为 No.186245/US/3 的专利申请；

[0017] 2005 年 6 月 8 日提交的名称为“用于可互操作的内容播放器设备引擎的方法和系统”，案号为 No.186245/US/12 的专利申请；

[0018] 2005 年 6 月 8 日提交的名称为“用于可互操作的设备实现硬件抽象层修改和引擎移植的方法和系统”，案号为 No.186245/US/18 的专利申请；

[0019] 2005 年 6 月 8 日提交的名称为“用于在间断连接的互操作的设备中维持设备完整性和安全性的系统、方法和模型”，案号为 No.186245/US/13 的专利申请；

[0020] 2005 年 6 月 8 日提交的名称为“用于间断连接的互操作的设备中的社会同步 (social synchronization) 互操作性的系统、方法和模型”，案号为 No.186245/US/19 的专利申请；

[0021] 2005 年 6 月 8 日提交的名称为“用于间断连接的互操作的设备中的社会安全性互操作性的系统、方法和模型”，案号为 No.186245/US/20 的专利申请；

[0022] 2005 年 6 月 8 日提交的名称为“用于配置和操作具有播放器和引擎的可互操作的设备的系统、设备和方法”，案号为 No.186245/US/6 的专利申请；

[0023] 2005 年 6 月 8 日提交的名称为“用于规定建立和形成可互操作的设备的智能组的方法和系统”，案号为 No.186245/US/21 的专利申请；

[0024] 2005 年 6 月 8 日提交的名称为“用于配置和使用虚指针以存取一个或多个独立的地址空间的方法和系统”，案号为 No.186245/US/22 的专利申请；

[0025] 2005 年 6 月 8 日提交的名称为“用于无缝通用设备互操作性平台的体系结构、装置和方法”，案号为 No.186245/PCT/2 的专利申请；

[0026] 2005 年 6 月 8 日提交的名称为“用于通用设备互操作性平台的设备组招募和内容再现的体系结构、装置和方法”，案号为 No.186245/PCT/3 的专利申请。

技术领域

[0027] 本发明一般涉及用于在具有相似或不同特性的设备之间提供通信并且便于在这些设备之间形成无缝互操作性的系统、设备、方法、以及计算机程序软件产品；尤其是涉及，用于在类似和不同的永久连接或间断连接的电子设备中，共享内容、应用程序、资源以及控制的软件、方法、系统以及设备。

背景技术

[0028] 在电子设备，特别是便携式设备和无线设备的数量和类型已经大量膨胀，以及应用程序的类型和设备类型膨胀的时期，已经出现了在这些各种异类的不同的设备类型之间直接共享数据和代码的相应的需要，以实现仅能通过利用多个设备的电子的和程序的资源才能实现的应用程序的目的。还出现了一种需要，并且对于设备用户来说，这种需要的程度继续增加，以能够与其它设备通信，对于用户需要的通信类型或数据传输或共享，可以提前建立或不建立所述的其它设备。例如，用户可能已经或希望在数字照相机中建立图片集，并能够接着将照片集直接传输到个人数字助理 (PDA) 类型的设备、电视，或投影仪，以进行观看；或者传输到存储设备，以进行储存；或者传输到打印机。用户还可能或者可选地希望将代码传输到另一个设备上，例如传输到比幻灯片所存在的设备具有更大的屏幕分辨率或更

好的图形处理能力的设备，其中的代码实现包含图片、标题、幻灯片索引等的幻灯片的顺序放映。用户也可能希望能够在可用的打印机上选择和打印幻灯片放映中的图片的子集。存在这种代码，数据以及内容共享的许多其它例子。

[0029] 传统的互操作性问题、难题、以及限制

[0030] 数据和代码的共享以及相关联的设备计算资源的共享，以及相似（同类）和不同（异类）设备或设备类型之间的设备控制，在所属领域被称为“设备互操作性”，或简单地被称为“互操作性”。包含在提供这种互操作性中的一些必须的和可选的突出问题包括：(i) 内容的适配；(ii) 内容的格式；(iii) 设备驱动程序；(iv) 设备到设备之间的通信；(v) 单个的设备资源和能力；(vi) 存在于设备上的应用程序；(vii) 在设备上加载应用程序；(viii) 与提供设备资源以支持互操作性相关联的费用；(ix) 可互操作的设备的电源或能量管理；以及 (x) 在设备之间的连接可能是间断和不可靠的互操作性环境中执行的代码的稳定性。此外，(xi) 实现互操作性所需要的开发、配置以及测试工作的范围；(xii) 独立地开发和或分配互操作性组件中所固有的可靠性问题，甚至在存在详细的互操作性标准的情况下也存在这种可靠性问题；以及 (xiii) 终端用户必需具有高水平的技术知识并且付出可观的时间和努力来管理互操作性的困难；(xiv) 互操作性设备，数据和内容的安全性；(xv) 关于互操作性基础设施可能产生的并引起其它的问题的尺寸，性能，功率管理以及成本折衷。。下面更加详细地论述这些问题。

[0031] 关于内容适配，需要根据这种参数（应用程序和数据类型决定的），如图片尺寸、用户界面、控制和特殊效果、内容格式、性质等，进行内容的智能调整或适配，这些都需要考虑从一种设备类型向另一种设备类型传输数据，应用程序，或控制的时间。这些被共同地称为“适配”。当共享内容，应用程序，以及控制时，适配地越复杂，可互操作的设备组越大；每个设备上的性能越高级，数据、信息和/或其它能力的传输就越有效，并且设备、代码数据和内容用于执行应用程序越容易。

[0032] 第二互操作性问题是由于这种不合乎需要的要求，即用户通常可能需要指定或至少考虑内容的格式。如果需要拥有对另一个设备的互操作能力的用户并不熟悉内容的格式和/或其它设备将如何处理内容格式，则即使该设备能够与其它设备进行通信，但是仅这一因素也可能对互操作性产生妨碍。

[0033] 第三互操作性问题是由于这种不合乎需要的要求，即在可以实现互操作性前，用户通常可能需要指定、考虑，或执行在一个或多个设备上加载一个或多个专用的驱动程序、代码、数据或内容。

[0034] 第四互操作性问题是由于这种不合乎需要的要求，即用户指定、考虑，或选择用户设备和一个或多个其它设备之间的通信所采用的物理通信机制和协议，所述的一个或多个其它设备中的每一个可能具有或要求通信机制、协议、接口等。

[0035] 第五互操作性问题是由于这种不合乎需要的要求，即用户可能需要考虑或选择哪个设备将具有与他人的设备进行互操作或者与需要的数据或应用程序进行互操作所需要的能力和存储器、处理器以及其它性能。

[0036] 第六互操作性问题是由于这种不合乎需要的要求，即用户可能需要指定、考虑，和/或加载必须存在于一些或所有被包含的和可能被包含的可互操作的设备上的应用程序。

[0037] 第七互操作性问题是由于在一个或多个设备上的代码、数据，或内容的丢失的、过

时，或不兼容的版本产生的完全的或部分的应用程序失败。

[0038] 第八互操作性问题是由于这种不合乎需要的要求，即设备需要拥有实现应用程序的所有代码，所述应用程序需要在生产时或之前的一些时间已经存在于设备上，或者被用户明确地下载到一些或所有设备上。

[0039] 第九互操作性问题是由于与提供实现通信所需要的处理器或 CPU 资源，存储器资源，电子门或逻辑或其它物理基础设施的数量和将要进行互操作的设备上的其它协议和应用程序相关联的货币成本。

[0040] 第十互操作性问题是由于需要提供有效的功率或能量管理方法以实现将进行互操作的便携式或移动设备所需要的延长电池寿命或减小电池大小的要求。尽管对于短期的互操作性没有特别地进行要求，但是也是十分需要这种电源管理的，从而与其它设备进行的互操作不会在这种设备上产生这种电池能量耗尽的问题，用户本来很少使用这些其它设备的能力或者不愿允许另一个用户对这些设备进行存取。

[0041] 第十一互操作性问题是由于对于应用程序的稳定度的要求，其中在设备之间的连接常常是间断的或不稳定的或不可靠的情况下需要继续执行这种应用程序。例如，当第二设备移出区域或另外地不能对来自第一设备的信息进行回答时，在与第二设备进行互操作和通信的第一设备上的用于实现应用程序的代码本身不会冻结、中止，或以其它方式产生严重的问题或导致设备自身的冻结、中止，或产生严重的问题。而且，理想的是，如果这种第二设备再次变得可靠和可用，则可以自动地恢复和更新实现互操作性应用程序所需要的所有代码、数据以及内容。

[0042] 第十二互操作性问题是由于应用程序的不可靠性，其中设备是被独立的制造商根据互操作性标准制造的，而该互操作性标准固有地不能预测实际的和将来的设备需求和能力，并且程序员或电路设计者不能完全地和正确地理解、执行该互操作性标准，并且也不能正确地对这种执行进行配置。第十三互操作性问题是由于缓慢的执行代码的速度，这种速度不能依赖于图形、视频、声音等所需要的优化。

[0043] 第十四互操作性问题是由于上面所列举出的所有问题阻碍用户和提供商，而不能使用可互操作的代码、数据以及内容，来实现可能用于互操作性的应用程序和设备。

[0044] 所述的十四个互操作性的问题仅是可能产生的问题类型的示例，不被看作是问题的完整列表或者也不代表在所有情况下产生的问题。例如，在制造设备时，彼此将进行互操作的两个相同的设备之间的互操作性不会产生这里所述的任何或所有问题，但是这种相同类型设备的互操作性并不代表设备用户现在所面临的更一般性的问题，并且为解决不同设备互操作性问题的适当的尝试也是不完善的、不是非常有洞察力的，并且也是明显不成功的。

[0045] 传统的静态的和程序性的解决尝试

[0046] 对于提供互操作性解决方案的传统的尝试一般被分为两种类型，即 (i) 静态的互操作性解决方案 (“静态的”)，或 (ii) 程序性的互操作性解决方案 (“程序性的”)。传统的静态的解决方案要求每个设备都支持相同的专用通信协议，并发送具有固定的字段布局的特定的严格规定的数据结构。在静态的方法中，在这些设备之间建立互操作性之前，语义、代码，以及显示能力必须存在于所有的设备上。在制造所包括的所有设备时，每个内容类型、应用程序，或设备能力必须是已知的、被执行的并被安装；或者可选地，用户必须按照

需要在启动需要的设备、软件数据或内容的互操作性之前，安装应用程序、协议，和 / 或驱动程序。由于用户可能并不是受过训练的信息技术专家，或者可能并不了解或拥有驱动程序、应用程序、操作系统组件、协议等的备份，因此就不能在可用的时间内提供需要的互操作性。而且，静态的解决方案也通常需要执行专用的静态解决方案组。

[0047] 例如，在数字照相机和电视或显示设备 (TV) 之间共享一组具有幻灯片功能的图片可能需要共同的静态协议，例如，用于将幻灯片图像数据和幻灯片次序或顺序信息发送到 TV 的蓝牙无线能力。同时还需要用于幻灯片和幻灯片次序的静态内容格式，以被 TV 识别为其知道如何进行处理的信息。在 TV 和数字照相机上必须存在至少一个可以以特定的内容格式表演和控制幻灯片放映的静态幻灯片放映程序。用户可以或不必分别地开始传递图像或图片以及幻灯片次序信息，在 TV 上关联并加入信息，以及在 TV 上运行正确的幻灯片放映应用程序。根据 TV 和数字照相机上的静态幻灯片放映程序的复杂度，可以使用也可以不使用数字照相机上的控制器来控制在照相机上启动的 TV 上的幻灯片放映。当不能使用这种基于照相机的控制时，就必须提供一些其它的用于进行控制的机制。静态方法能够产生高度优化的解决方案，以很好地理解在制造能够互操作的所有设备类型时就已知的专用应用程序。然而静态方法具有许多限制，包括在制造时要了解并且按照惯例实现几乎所有的能力，在制造后的有限的升级和改正错误的能力；以及这种传统的要求，即每个静态程序的执行必须被正确地和完整地移植，以在不同的设备上运行，并在进行互操作前就存在于所有的设备上。这种要求通常是通过为特定的应用程序、通信介质和要求互操作性的需要的设备组加载和更新特定的驱动程序来实现的。

[0048] 由于必然会存在不同版本的标准和应用程序，因此甚至在可以使用静态方案的情况下，可靠性也会被大打折扣。因此当两个设备希望共享数据或程序时，当设备是遵循不同版本的标准，或者程序遵循不同版本的标准，或者在设备上存在不同版本的应用程序时，就会出现故障。其它的可靠性问题是由于在用于进行互操作的标准集的独立执行中不留心的错误或简化操作。当任何两个实施方案试图一起工作时，这种标准的执行就可能以不可预料的方式进行互相影响。通常在所有的设备组上测试所有标准的实施方案的所有排列是不现实的或者是根本不可能的，尤其当制造初始设备时与初始设备进行互操作的所有目标设备并不存在时，更是如此。

[0049] 对静态方法的更重要的限制之一在于当设备和 / 或应用程序的数量 N 变大时，使数量为 N 个的设备或应用程序进行互操作所需要的工作量增长也极迅速。目前制造商在为内容类型、应用程序（“程序”）、通信协议，等建立成百个静态标准以试图产生更有限数量的设备组时是失败的，其中的设备组可以通过不断增大的大型设备组和应用程序进行互操作。通常这还要求每个设备都具有存储器、屏幕容量、控制器以及处理器和电池能量，以支持用于所有需要的可互操作的设备上的每个需要的互操作性选项的每个静态解决方案。否则就不能实现真正的设备和应用程序的互操作性。为了更好地说明该传统的问题和限制，考虑目前适配需要对必须与其它设备 (N-1 个设备) 类型进行共享的每个设备 (N 个设备) 类型进行软件工程开发计划。从开发的观点来看，这是一个 N×N 即 N^2 级的问题，因为在需要彼此进行互操作的所有 N 个设备的范围内，要考虑、形成、执行和测试 N×(N-1) 次适配。

[0050] 而且，随着设备的数量增加以及需要的适配次数增大到 N^2，由于总体复杂度增加，因此得到高质量产品的花费和困难将以更快的速率增加。这是因为得到高可靠性和高质量

的软件和硬件方案的困难随总体复杂度的增加而增加。这就不单纯是“源代码大小”的问题，而是由于当试图使来自不同制造商的设备共同工作时普遍遇到的这种的问题，包括行为的不可预测性，事件的不可预测性，未知的将来能力，等等。

[0051] 例如，随着设备的数量从 5 增加到 6，互操作性的适配要求则根据关系式 N×(N-1) 从 20 增加到 30。并且随着这种增加，计划的总体复杂度会以更快的速度增加。

[0052] 使 N 个设备共同工作的 N- 平方级的问题基本要求图 1 所示的 N^2 的适配。应该知道使用静态方法的传统的设备内的协同工作对于进行使用的用户来说可能是相对简单的，但是却要求更高程度的开发、管理和配置工作以及不断的更新，以在旧的设备、应用程序，以及数据类型和新的设备、应用程序，以及数据类型之间保持兼容性和互操作性。

[0053] 参照图 2 所示，表示设备内协同工作的互相作用，或使用强力法的仅八个设备类型的有限的互操作性，其需要 56 次适配和另外的 56 个测试程序。

[0054] 除去强力法的 N^2 的问题，大多数静态方法还包括组合许多标准（或多个标准）的工作。来源于微软的 UPnP（通用即插即用）方法或许拥有最大范围的静态方法。UPnP 是通过包含一组静态（不是基于程序的）标准，来解决一些与设备互操作性相关联的问题的静态的非程序化的方法，该方法试图列举所有不同类型的设备和服务，其中的每个设备和服务都具有不同的 XML 或基于数据结构的描述。然而，即使是 UPnP 方法也有许多限制，下面将简要地描述其中的一些限制。

[0055] 首先，UPnP 是大型化的方法，这在于其需要大量的模块、代码、电源，以及存储器来进行运行。这就这种方法不适于具有较小的处理器、小的随机存取存储器、以及小的电池容量的低成本电池供电的设备。

[0056] 第二，UPnP 几乎没有提供优化内容或性能的能力。UPnP 通常假设一种类型的应用程序、内容，或用户接口会适合于被包括的所有设备。在十年前对于基于微软 Windows 的桌上个人计算机 (PC) 的来说，这种假设可能是合理的，但是现在对于到处都是必须进行互操作的设备的世界来说，这已经是不合理的假设和根据了，其中的互操作的设备可以是寻呼机、数字照相机，以及个人计算机，其各不相同，更不用说可能在以后的几十年中出现的混合的和不同的电子设备组了。

[0057] 第三，UPnP 仅提供了有限的用户接口组，不能满足现在可用的大型的不同设备的组的要求。

[0058] 第四，UPnP 要求执行要求的任务所需要的程序和驱动程序在其被使用之前就已经存在于所有设备上。

[0059] 第五，尽管 UPnP 的目的至少是部分地避免 N- 平方 (N^2) 的问题，但是事实是如果独立执行复杂标准的所有排列会产生可靠的互操作性，正如上面所述的，则使用 UPnP 作为互操作性的根据仍将需要大量的 N- 平方 (N^2) 次开发 / 配置 / 测试的工作，以产生需要对所有进行互操作的设备移植、分配和测试的程序、代码、数据以及内容的新的应用程序。

[0060] 第六，UPnP 程序、设备、内容以及标准必须都是同步的，从而同时配置相同的或至少兼容的版本和更新。

[0061] 第七，随着设备和内容能力的发展，现有的基于 UPnP 程序、数据和内容的设备将不能支持新的设备。

[0062] 第八，维持现有设备、标准（包括 UPnP），和应用程序的版本的兼容性的成本更快

速地增加了全部计划的复杂度。

[0063] 第九，随着计划的复杂度的增加，由于作为基于标准－静态方法的 UPnP 的复杂性和差异性中所固有的问题，使用的方便性和可靠性降低了。

[0064] 第十，UPnP 仍不能解决在设备之间发送一个或多个数据结构的频繁要求所强加的需要，特别当这些数据或数据结构是以人类可读的文本格式表示时更是如此，因为人类可读的文本格式与二进制表示相比需要更大的传输带宽和时间。而且，在设备之间发送的许多数据结构是以 XML，人类可读的文本格式表示的，而不是以二进制或更不通用的格式表示的，其中使用 XML 格式要求更多的 CPU 操作、存储器和／或软件程序代码的数量，以执行 XML 所需要的 CPU 密集型的解析操作。

[0065] 最后是关于传统的静态方法对设备和应用程序互操作性所强加的一些限制，静态方法通常限制协议、内容类型和应用程序类型的数量，以降低总体复杂度和基于标准的执行步骤的多少。一个例子是 UPnP 仅允许 TCP/IP 作为基本的通信协议。这就有效地降低了其它重要的已有通信协议的有效利用，例如，蓝牙、USB、MOST，以及 IR 和所有其它的非 TCP/IP 协议。

[0066] 传统的程序性解决方案的尝试

[0067] 静态标准方法的一种替换方案依赖于程序性标准的建立。普遍存在的是以硬件实现或以软件仿真的程序性标准技术。存在许多的硬件微处理器，每个微处理器具有针对不同类型的问题进行的优化的指令集和接口，并且还存在许多高级软件仿真的指令集和实际的环境，关于特定的任务组对其进行优化。这些包括例如，Java（通常对程序的可移植性和简易性进行优化的方法），PostScript（通常对表示打印的页面和打印机控制功能进行优化的方法），以及 Storymail Stories（通常对有效地表示很宽范围内的丰富的多媒体消息进行优化的方法）。Java 和 PostScript 是计算机技术领域所熟知的。例如，在于 2003 年 1 月 9 日出版的名称为“用于安全通信和信息传递的硬件体系结构、操作系统及网络传输中性系统 (neutral system)、方法和计算机程序产品”，发明人为 Michael L Wenocur、Robert W.Baldwin，和 Daniel H.Illowsky，美国专利申请出版号为 20030009694 A1 的专利，于 2002 年 11 月 7 日出版的名称为“安全证书以及用于发布和使用该安全证书的系统和方法”，发明人为 Michael L Wenocur、Robert W.Baldwin，和 Daniel H.Illowsky，美国专利申请出版号为 20020165912 A1 的专利，以及其它专利申请中描述了 Storymail Stories 和相关系统和方法的各个方面。程序性互操作性方法典型地包括在所有要进行互操作的设备上建立或另外地具有或提供共同的运行时间的环境，从而除了静态数据结构和静态应用程序外还可以在设备之间发送程序、步骤、数据和内容。目前一种主要的互操作性程序性解决方案是具有 JINI 扩展的的 Java 平台。作为基于 Java 的程序性方法的例子，以 Java 编写的幻灯片放映可以包括或参考图片或幻灯片的次序或顺序数据，询问其它设备，使内容与其它设备适配，并向 TV 发送信息和 Java 幻灯片放映程序。在生产出照相机后，Java 幻灯片放映程序可以在照相机上运行，并即使幻灯片放映程序没有存在于 TV 之上，也能实现与 Java 启动的 TV 的互操作性。

[0068] 尽管已经普遍地使用 Java，并且 Java 在提供相应的有限的互操作性方面也取得了一些有限的成功，但是 Java 仍具有一些严重的不足，这些不足阻止了 Java 的更广泛的应用，特别是对于小的移动设备更是如此，在这些小的移动设备中，成本，电源效率，处理器效

率，用于存储程序代码、数据，和临时缓存的存储器效率是非常重要的问题。同时，为解决运行在不同设备上的应用程序的二进制兼容性的 Java 虚拟机 (VM) 方法恰恰与为什么设备存在的原因相冲突。Java 和其它传统的程序性互操作性方法都具有严重的限制。下面描述五个示例性的限制。

[0069] 首先，Java 虚拟机方法使或者至少试图使所有的设备对与应用程序来说如同相同的虚拟计算机，以允许相同的二进制代码（Java 二进制代码）运行在所有设备上。为了维持二进制兼容性，就需要避免试图存取并没有被预定义为虚拟机的定义和实现方案一部分的设备能力。因此如果需要本地功能来存取不是共同虚拟机定义一部分的任何设备上的能力，则在多个设备上就失去了二进制兼容性。由于多数非 PC 设备的硬件和软件通常是为特定的用途、构成因素、价格点、用户接口或功能而进行指定和优化的，因此对于经常为设备指定的应用程序，通常必须存取这些设备的基本的独特的本地功能或能力。对于多数便携式或专用设备，它们存在的原因正是因为对独特的不同能力和功能的需要。这就违背了隐藏设备之间的不同以使它们对应用程序看起来都相同的 Java 虚拟机的方法。

[0070] 第二，Java 是一种针对编程的简易性进行优化但是以执行效率和资金利用率为代价的通用语言。因此，对于许多具有一般的处理能力和很少可用的存储器的小设备来说或者当成本是非常重要的因素时，Java 不是效率最高的或最有效的解决方案。

[0071] 第三，使用 Java 程序性方法不能确保多媒体内容的响应次数。多数 Java 程序严重地依赖于大小变化的存储器结构的频繁分配和隔离，这就导致了存储器分段存储。这种存储器分段存储通常会产生设备中的处理器必须停止再现内容而执行存储器中的无用数据收集的时间段。当这种情况发生时，用户通常就会遇到平滑的音频和视频再现中的中断。

[0072] 第四，Java 存在严重的速度和大小的问题。Java 和其相关的技术和互操作性所需要的库相对较大，并需要相对较多的进行处理的 CPU 周期数，和相对大的用于进行存储的存储器空间。以 Java 编写的互操作性程序包括用户接口、多媒体再现、设备列举、平台上设备互操作性的稳定性、发送到不同类型设备的代码和数据的动态适配，这种互操作性程序需要编写和交换大量的代码，这是因为必须使用库，或专用的 Java 代码序列来建立所有这些功能，并且这些操作都不是 Java 指令集或环境本身所具有的。其结果是用于互操作性的 Java 程序非常大并且运行速度很慢，从而限制了其在具有有限的处理器能力、电池寿命的设备上或成本非常重要的情况下的应用。当设备不具有足够的资源时，基于 Java 的互操作性解决方案就不能实现。

[0073] 第五，Java 至少为互操作性提供有限的和不完全的基本实现方案。这就使得大量的库必须存在于进行互操作的所有设备上，或者具有到包含必要的程序代码的服务器的高速的持续的连接。必须在 Java 语言自身中提供没有包括在 Java 基本指令集中的操作的性能，这就大大地限制了相对于本地代码的运行时间性能，其中可用本地代码实现这些操作。失去互操作性的基本操作包括对于下述的本地支持：(i) 多媒体动画重放；(ii) 程序、数据、内容、用户接口或控制到目标的其它设备的适配；(iii) 定制的程序的计算机生成从而产生内容的设备可以自动地和容易地将该内容与互操作性程序融合在一起；(iv) 在宽范围的各种协议内发现设备、服务和资源；(v) 在不同设备上运行的过程的同步和 / 或串行化；(vi) 设备电源管理；(vii) 当设备间断地失去和恢复其连接时，应用程序和同步的恢复。

[0074] 通常当证明 Java VM 规范缺乏一种类型的设备或应用程序时，就会出现新的 Java VM 规范以说明现在已知的这些新的设备类型的本地支持要求；然而，为一个 VM 编写的 Java 程序通常不是二进制兼容的或可以与遵循不同的 VM 规范的设备进行互操作的。针对各种设备类型存在 Java VM 规范，包括 J2ME，MIDP 1.0，MIDP 2.0，以及 CDC，但是更加不具有互操作性的 Java VM 规范和执行过程的增加继续导致程序和设备的类型和形式的分段存储，从而通过使用 Java VM，实现了更小程度的互操作性。

[0075] Xerox 帕洛阿尔托 (Palo Alto) 研究联合机构 (PARC) 宣布对 Java 和 Jini 互操作性技术进行改进，其被称为“Obje”。Obje 是明显地基于 Java 的，或者作为一个替换方案，是未规定的和未实现的类似的基于虚拟机的技术。尽管 Obje 针对一些提供程序性方法的方式，其中的程序性方法是对有效的设备编组是需要的并使所有设备不需要使程序移植或存在于所有机器上，但是可以想到 Obje 执行过程具有与 Java 加 Jini 方法相似的能力和限制，因为他们没有为使用的程序性基础提供显示与 Java VM 模型的分歧的细节。

[0076] PostScript 是另一种程序性方法，已经存在了相当长的时间，提供打印页面描述语言，该打印页面描述语言可以非常有效地在 PostScript 文件和 PostScript 打印机之间建立高度的互操作性。PostScript 文件是这种程序，即当其在打印机内的 PostScript 软件引擎上执行时，控制硬件打印机引擎并重新产生打印页面的图像，同时利用其自己所存在的打印机上的可能的最高的分辨率。PostScript 被极大地限制于文件和打印机的互操作性。对于该限制的一些原因包括这种事实，即 PostScript 文件是以人类可读文本表示的。这就相对于二进制程序极大地增加了文件和程序的大小。当程序运行时，文本需要解析操作，这就要求更多的处理器周期，并且如果程序是以二进制形式表示的，则需要暂时存储器。而且，PostScript 对于如下各项并不提供任何重要的本地支持：(i) 多媒体视频 / 音频 / 动画重放；(ii) 为其它目标设备适配应用程序、数据、内容、用户接口或控制；(iii) 设备、服务以及资源发现；(iv) 运行在多个设备上的程序的同步和串行化；(v) 设备电源管理；(vi) 发现并使用其它设备；(vii) 维护设备之间的稳定的连接；或 (viii) 有效地存取各种存储介质，包括现在普通用在设备上的共用的闪存。

[0077] Storymail Stories 提供为包括多媒体消息而设计的可变长度的程序指令集。例如，在 2003 年 1 月 9 日出版的名称为“用于安全通信和信息传递的硬件体系结构、操作系统及网络传输中立系统、方法和计算机程序产品”，发明人为 Michael L Wenocur、Robert W.Baldwin，和 Daniel H.Illowsky，美国专利申请出版号为 20030009694 A1 的专利，于 2002 年 11 月 7 日出版的名称为“安全证书以及用于发布和使用该安全证书的系统和方法”，发明人为 Michael L Wenocur、Robert W.Baldwin，和 Daniel H.Illowsky，美国专利申请出版号为 20020165912 A1 的专利，以及其它专利申请中描述了 Storymail Stories 和相关系统和方法的各个方面。

[0078] Storymail 发明包括 Storymail Story 结构和相关的技术，它是由与该专利相同的发明人 Daniel Illowsky 发明的。Storymail 指令集允许以所谓的通用程序格式表示自动编码的多媒体内容，“stories”提供了相对于迄今为止已知的多媒体内容表示的许多优点。然而，Storymail 技术没有完全解决设备到设备的互操作性问题，并且如果尝试使用 Stroymail 技术来实现设备到设备的互操作性，则一些问题和限制就会马上变得很明显了。首先，Storymail 指令集是为需要使用许多临时存储器的小的引擎优化的。第二，Storymail

指令集需要在运行内容时执行专用的线程模型，加重了负担。第三，Storymail Story 甚至需要对基本的内容类型进行自动编码，例如 JPEG 或位图图像。第四，Storymail Story 技术没有提供对设备、服务，或资源发现的本地支持。第五，Storymail Story 技术没有对运行在多个设备上的程序同步的本地基本支持。因此，尽管 Storymail 技术和通用程序媒体格式相对于目前的传统技术提供了许多进步，但是并没有令人满意地解决电子和计算机领域明显存在的设备到设备的互操作性问题。

[0079] 因此，显而易见的是静态的和当前程序性方法都不能提供令人满意的设备到设备的互操作性，特别是对于不同的设备和先验未知的应用程序来说，更是如此。当考虑当前的客户机－服务器和对等设备内互操作性模型时，这种问题就变得更加复杂。

[0080] 传统的客户机－服务器和对等模型

[0081] 在所属领域的当前状态中，在多个设备上运行的进行互操作的程序通常使用客户机－服务器模型或对等模型。

[0082] 在客户机－服务器环境模型中，一个设备（例如，服务器）提供服务，而另一个设备（例如，客户机）使用该服务。这就允许多个客户机设备利用一个更有能力的设备来存储和处理数据，同时重量更轻的客户机设备仅需要足够的能量来请求和显示结果。客户机－服务器方法的限制在于通常服务器必须在网络上注册，并且在任何时刻都能被客户机以相对高速的连接进行存取。

[0083] 在对等环境模型中，通常假设任何互操作的设备都具有执行应用程序所需要的所有功能。同时在对等模型中，通常进行互操作的所有设备都必须在建立连接之前了解将被使用的程序或服务，从而它们能够对合适的连接和协议达成一致。必须具有执行应用程序的全部能力，并且要求对等的程序协议层预先存在于所有进行互操作的设备上，这种要求是将对等模型应用到设备的互操作性上的重要限制。实际上，对等设备通常会遇到软件的不同的非理想的执行过程或版本，这些软件的不同的非理想执行过程或版本对于不可预测的迭代不能进行协作。为了改正这些问题，通常必须更新、分配和安装驱动程序或其它软件。尽管这样通常能够改正互操作性问题，但是管理、执行、分配和与故障相关的失效以及修复这些失效所需要的复杂度、完善度和时间仍然是基于对等设备的互操作性的严重问题。

[0084] 在个人计算机世界中，尽管具有有限的互操作性但是当今最流行的平台是微软的 Windows™ 操作系统。在微软的 Windows™ 操作系统下，不管 PC 是由 IBM、Toshiba、Sharp、Dell 或任何其它的制造商制造的，原理上任何应用程序二进制图像可以在运行微软的 Windows™ 操作系统的任何标准的 PC 体系结构设备上运行和使用。然而，实际上，在真实世界操作环境中即使是微软的 Windows 也具有关于互操作性的限制。

[0085] 因为计算设备和信息设备从一般通用个人计算机模型中分出进入特定的专用设备，如移动电话、移动音乐播放器、远程控制器、可组网的媒体播放器和路由器，以及许多的其它设备，因此就需要应用程序平台以允许快速和有效的开发应用程序，其中的应用程序不仅在所有（或至少多数）设备上是二进制兼容，而且可以在根据每个设备资源的多个协议上动态地 (on the fly) 形成特定的设备组。应用程序需要能够在所有的设备上扩展执行才能执行应用程序，其中没有一个设备具有执行应用程序所需要的所有的软件、硬件或其它资源。

[0086] 目前在所属领域的状态中，还没有用于生成能够在多个设备上运行并传播程序的有效的软件平台，特别是当将传播到的设备是不同类型并具有不同的设备硬件、软件，以及操作系统（如果有的话）特征时，尤其如此。尽管存在许多标准化的嵌入式的操作系统，但是因为需要应用程序来存取设备的独特的能力和性能，包括显示和控制的配置，因此为一个设备生成的程序几乎不能运行在另一个不同的设备上，即使两个设备使用相同的嵌入式操作系统和处理器也是如此。因此，很明显在所属领域十分需要改选的方法和系统，以提供可靠的、便于使用的设备、应用程序、数据以及内容互操作性，同时避免迄今为止已知的现有技术装置和方法的缺点和不足。

发明内容

[0087] 本发明包括许多有创造性的系统、设备、装置、计算机程序以及计算机程序产品、程序和方法，共同使设备和系统的互操作性与目前为止所属领域的当前状态相比，变得更简单、更可靠、更稳定、更强大、更成本有效并且更安全。所采用的技术是根据程序性互操作性技术。本发明在许多方面与现有的程序性互操作性技术不同。尽管现有技术试图隐藏设备之间的区别从而相同的可执行的二进制图像可以在所有设备上运行，但是本发明向各个设备展示并提供了到设备所有资源的存取，从而应用程序能够形成特定的设备组，并有效地将其执行扩展到设备组上，就如同设备组中的所有设备是一个设备一样。

[0088] 多数现有的互操作性技术是这些年来为公司或政府网络中使用的强大的通用计算机开发的技术的扩展，这些计算机总是连接在可靠的高速网络上，很少重新配置，连续地使用足够的电源，并由受过训练的全职的专业人员进行配置和维护。当这些技术用于现在变得可用的移动的、电池供电的、间断连接的以及成本受限的设备时，并且被没有受过训练并不愿关注互操作性所需要的软件和硬件的配置和维护的人员使用时，这些技术就会显得不足了。

[0089] 本发明必须包括一种新的软件生态系统方法，这些方法共同工作以极大地改进现在以增长的速度进入市场的移动设备和其它专用设备的互操作性的简单性、稳定性、成本有效性、效率和安全性。

[0090] 图 3 表示本发明的新的程序性和软件生态系统的实施例的示例性组件，其中的系统被称为 DartPlatform，其本身是互操作性平台的形式。源，这里是 DartSource100，包括实现应用程序的目的所需要的所有代码、数据以及内容。DartTools 200 将 DartSource 处理为二进制可执行的应用程序包，该应用程序包包括实现最初由 DartSource 所指定的互操作性应用程序目的所需要的所有内容。二进制图像包与 DartFormat 300 一致。所包括的应用程序称为 Darts，它将在一个和多个 DartDevices 400 上执行。运行 DartPlayer 并具有至少一个用于与其它 DartDevices 进行通信的通信协议 401 的任何设备本身是能够运行 Darts 的 DartDevice，能够将执行延伸到其它的 DartDevices 上以利用它们的组合能力和资源，其中 DartPlayer 是本地代码可执行的程序并包括移植的 DartEngine 600。图 4 表示示例性的 DartDevice 3000。在上部表示了运行在设备 3000 上的三个 Dart 应用程序 3001。这些 Darts 的代码由 DartTools 生成（参照图 3200 所示），以与发明的 DartInstructionSet 相一致，其中 DartEngine 3010 的便携部分和 DartEngine 的设备专用部分，硬件抽象层 (HAL)3020，以安全的方式执行 DartInstructionSet 的各个操作。

[0091] 注意到 DartEngine 执行的 DartInstructionSet 操作包括互操作性所需要的处理器密集的操作，例如加密操作、图形和文本处理。此外，在该部分的后面部分将更详细地描述用于发明的互操作性方法的引擎中嵌入的支持。HAL 包括通过 Dart 的 PROFILE_INSTRUCTION 指令存取的轮廓 (profile) 方法，以确定设备专用的设备特征和功能。HAL 还为引擎提供了存取它们所位于的共同的 Dart 标准硬件功能的方法。可能多数深奥的发明是 HAL 的这种部分，即可以用于将所有本地应用程序，设备的功能和资源显示给运行在设备上的 Darts 和 Darts 的部分，接着执行过程延伸到其它设备上的 Darts 可以利用该设备。

[0092] 互操作性的简单性、可靠性以及稳定性部分地是通过包括特定的应用程序目的所需要的所有代码、数据以及内容和源代码、数据以及内容，以将其智能地和有效地传播到不同的设备上而实现的。由于运行在互操作性设备上所有应用程序代码、数据和内容都来自于单个 Dart 包，就不存在与独立生成和分配的组件相关的不兼容或管理的问题。将数据和代码打包在一起还消除了共同的互操作性的问题，该问题是由数据格式和被选择用于管理数据的应用程序之间的版本不兼容引起的。

[0093] 存在至少 21 个单独描述的独立发明的系统、方法、计算机程序以及计算机程序产品，和 / 或装置，它们相对于现有的互操作性技术来说有助于增强设备互操作性的稳定性、能力、效率以及安全性，其中现有的互操作性技术主要是这些年来为强大的通用计算机网络开发的技术的扩展。下面简要地说明了这些革新并接着在说明书的后续部分中进行了更详细的描述。当对独立的创造性的系统、方法、计算机程序产品，和 / 或其它装置进行有用的组合时就会有更多的技术。当所采用的许多技术形成特定的组并共同工作来执行特定的任务时，就享有了人类所采用的社会化方面。与通用计算机网络相比，更加专用的设备和移动设备的快速增长的世界通常与人类具有更多的共同点，例如协作，其中专用的设备和移动设备需要形成特定组并仅是间断连接的，传统的互操作性技术是借用通用计算机网络的。

[0094] 招募互操作性模型 招募是现有的客户机 / 服务器和对等设备互操作性模型的有益替换。单个软件应用程序包或 Dart 采用招募，根据设备的能力和内容形成设备组，并接着将其自身的智能扩展部分传播到设备组上，设备组就能够共同工作以实现 Dart 应用程序包的目的。

[0095] 再现适配和互操作性分段模型 再现允许将互操作性应用程序分段为多个紧密结合的，但是独立的可执行程序。在招募过程中选择各个再现程序以将其发送并在其它设备上运行，从而提供对各个设备的能力和内容的协同存取。

[0096] DartSource/ 互操作性资源 DartSource 是用于为打包的 Dart 互操作性应用程序指定所需要的所有程序再现以及代码、内容和数据的方法。DartSource 将通常用于指定面向特定设备的单个可执行程序的语言结构扩展为下面这种语言，即也可以指定设备组的智能招募所需要的程序和所需要的再现程序，从而合适的再现程序被发送并在每个被招募的设备上运行，以实现被指定的应用程序的目的中与设备相应的部分。

[0097] DartFramework/ 互操作性框架 DartFramework 是程序员在建立互操作性应用程序中所使用的 DartSource 的部分，其中的互操作性应用程序包括利用 DartPlatform 的许多有益特点，而不要求程序员必须理解并实现 DartPlatform 的许多希望的互操作性特点。

[0098] DartTools/ 互操作性工具 DartTools 将 DartSource 应用程序规范处理为 Dart

应用程序包。

[0099] DartFormat/ 互操作性格式 DartFormat 是用于将 Dart 包放在一起的规则，其中的 Dart 包包括互操作性应用程序所需要的所有代码、数据，和内容，可以加载所述的代码、数据，和内容，并在包括运行的 DartPlayer 的 DartDevice 上运行。

[0100] DartRuntime/ 互操作性运行时间 DartRuntime 是用于在运行 Dart 的独立的处理单元之间建立控制、数据以及操作的密切配合的系统，而无论所述处理单元是运行在单个设备上还是运行在被招募的设备组上。这是借助于事件驱动的系统实现的，其中所述事件驱动的系统确保了贯穿应用程序的所有处理单元的事件的串行化和同步，从而所有的处理单元能够以需要的相同顺序存取所有的指令，以协调和同步处理单元之间的数据和操作。

[0101] 线性任务分配线性任务分配是对通常用在多数设备上传统的有优先权的和协作的线程模型的有益替换，从而可以指定和运行多个操作，就如同这些操作的动作是被同时执行的一样。线性任务分配为处理单元确保了一种简单的、可靠的、灵活的以及可扩展的方法，以非常确定的并且易于测试的方式协调其动作。线性任务分配是在单个设备中操作的 DartRuntime 的一部分。

[0102] 垂直分层垂直分层是对通常使用在多数设备上的协议的水平分层的有益替换，水平分层要求不同等级的协议通过所有中间级协议进行通信，通常需要对信息进行转换以符合每个协议接口的不同的要求。Dart 处理单元采用垂直分层，从而不考虑其等级，通过利用事件结构和事件队列，处理单元能够直接与所有其它处理单元进行通信，其中的事件结构和事件队列是所有处理单元能够存取和理解的而不需要进行转换。

[0103] 应用程序驱动的电源管理 Dart 应用程序是采用至少部分地包括在 DartFramework 中的线性任务分配和或垂直分层构建的，因此总是能够跟踪其准确的响应时间需求，从而有效的电源管理技术，如使处理器速度减慢，能够延长电池的寿命，限制消耗的能量，或者限制设备上产生的热量。在所属领域的当前状态中，大多数应用程序不能跟踪其响应时间需求，并且如果这些应用程序能够跟踪的话，也不能通过现有的协议层传送这些需求，其中现有的协议层符合的规范不包括用于从应用程序到设备的硬件的传输响应时间需求的接口的规范。

[0104] 互操作性应用程序驱动的错误恢复由于低容量电池电源引起的干扰、距离限制，以及突然断电，设备之间的无线通信连接通常是不可靠的。在传统的水平分层协议中，在设备的软件执行中，任何层中的重大错误将会导致不可恢复的错误，这是因为应用程序不具有标准的接口以容易地重新建立连接和连接环境，并且因为传统的应用程序不具有许多在运行在不同设备上的应用程序之间跟踪和重新建立共享的状态的基础设施，因此对于应用程序来说这种错误将难于恢复。DartFramework 跟踪共享的状态，并使用再现容易地在设备和垂直分层之间重新建立失去的状态，这就使得通信错误易于被传播到 Dart，并且 Dart 易于将恢复信息直接传播到通信处理单元。因此当恢复连接时，甚至当先前失去的设备已经丢失其所有应用程序状态时，运行在多个设备上的 Darts 能够无缝地从协作的设备之间的间断的完全失去的通信中恢复，并且恢复设备的共享状态。

[0105] 互操作性指令集互操作性指令集用于表示 Dart 的代码部分。DartEngine 执行这些指令。互操作性指令集包括提高操作速度，执行互操作方法以及将设备的能力和内容展示给其它设备的指令，并且包括传统处理器的传统的读取、存储、测试、计算以及分支指

令。需要特别注意的是还存在这种指令，即甚至当其它设备之前并不了解独特的能力和内容时，也将设备的独特的能力和内容的使用展示给其它设备的指令。

[0106] 创建法创建法是一种 Darts 采用的用于动态地生成为特定的目标设备和或通信会话和或目的定制的 Darts 的方法。DartInstructionSet 中的指令用于从运行的 Dart 部分自身中和通过运行 Dart 收集或计算的任何信息中程序性地生成 Dars。

[0107] 互操作性引擎 /DartEngine DartEngine 是用于在设备上执行 Darts 指令并实现其目的的软件和或硬件。DartEngine 和设备专用的 DartPlayer(DartEngine 包括在其中)，提供了共同的执行过程和 DartRuntime 环境，该环境允许招募和再现建立高效的设备组，并尽可能地传播其代码、数据以及内容，以实现 Darts 的目的。

[0108] 互操作性设备授权互操作性设备授权是通过将 DartEngine 移植为 DartPlayer 的部分，将传统的设备变为具有高度互操作性的 DartDevice 的过程。此外，还需要存取设备专用信息、能力和内容所需要的硬件抽象层的执行过程。在具有 DartPlayer 的设备变为 DartDevice 之前，必须执行至少一个通信协议。

[0109] 互操作性安全模型 /DartSecurity DartSecurity 是用于提供需要的基础设施以保护设备的完整性和设备内容不会被恶意的和意外的破坏的系统。

[0110] 社会同步互操作性方法 /Dart SocialSynchronization 社会同步是一种有效的并易于管理的方法，用于在任何数量的设备和协议上同步特定的数据组和或操作，而不需要每个设备都与主设备相接触，也不需要任何设备充当主设备。设备和内容的社会同步类似于人类共享信息和任务的方式，是所属领域的当前状态中通常采用的主同步技术的有益替换。

[0111] 社会安全互操作性模型 /Dart SocialSecurity 社会安全是一种特别便于管理的方法，用于在可能进行间断连接的设备之间形成安全的网络。社会安全的工作方式类似于人类如何彼此信任的方式。社会安全的基础是使用 SocialSynchronization 传播利用 DartSecurity 系统生成的唯一的 ID 以及可传递地从一个设备传播到另一个设备的允许的存取权限。从来没有直接地进行通信的设备通常会发现它们是设备组的一部分，其中允许所述的设备以一定的存取权限进行互操作而不需要进行其它的获得允许的操作。

[0112] 具有互操作性的设备 /DartDevice 借助于运行包含 DartEngine 的 DartPlayer 以及用于连接到其它 DartDevice 上的至少一个通信协议，DartDevice 是能够进行高度的互操作的设备。

[0113] 互操作性平台 /DartPlatform DartPlatform 是 Dart 方法的任何组合，能够实现 DartDevice 的指定、生成、智能编组，并且有助于在一个或多个 DartDevice 上传播并运行 Dart 互操作性应用程序。

[0114] 虚指针虚指针是一种用于为程序员提供简单和有效的方式以存取和使用一个程序中的一个或多个独立的数据地址空间的方法。软件程序所使用的虚指针能够改变对主存和存储器设备的应用，以在具有不同大小和速度的主存（速度快但是容量小）和存储器（容量大但是速度慢）的设备上进行有效地运行。

附图说明

[0115] 应该结合随附的附图理解说明性实施例的详细描述，其中：

[0116] 图 1 的图示表示使 N 个设备协同工作或一同工作的情况通常需要大约 N^2 次适配；

[0117] 图 2 的图示表示使八个设备直接共同工作的复杂度，使用大约 56 次适配，并需要 56 次不同的测试配置以测试和验证操作；

[0118] 图 3 的图示表示本发明的主组件和子组件的模块图；

[0119] 图 4 的图示表示根据本发明的实施例的示意性的典型的 DarDevice 的概述；

[0120] 图 5 的图示以流程图的形式表示招募过程的示例性实施例；

[0121] 图 6 的图示表示说明本发明招募方法的图表，该方法用于扩展第一 DartDevice 上运行的 Dart 应用程序（例如幻灯片放映 Dart 应用程序）并将该程序与第二 DartDevice 共享，所述第二 DartDevice 最初根本不了解初始的第一 DartDevice 或 Dart 应用程序；

[0122] 图 7 的图示表示说明招募的例子的图表，其中招募用于在另一个 DartDevice 上远程打印从初始 DartDevice 上运行的幻灯片放映应用程序，所述另一个 DartDevice 最初根本不了解初始的第一 DartDevice 或幻灯片应用程序；

[0123] 图 8 的图示表示说明打印图片应用程序的示例性实施例的图表，打印图片应用程序被表示为在三个设备构成的组中运行的 Dart，其中每个设备运行初始 Dart 的不同再现程序；

[0124] 图 9 的图示表示 Dart 应用程序的同步和串行化的实施例，其中 Dart 应用程序在被招募的协作的设备组中运行；

[0125] 图 10 的图示表示招募如何使一组设备作为一台设备工作，以及如何限制实现 N 台设备的互操作性的所需要的工作，从而所述工作简单地与 N 成比例；

[0126] 图 11 的图示表示主 DartFramework 对象及其在类的体系结构中的相对位置的实施例的模块图；

[0127] 图 12 的图示表示说明应用程序开发设备和其用于产生 Dart 应用程序的静态的和程序性的组件的模块图；

[0128] 图 13 的图示表示说明 Dartformat 的实施例的各部分组件的结构的实施例的结构；

[0129] 图 14 的图示表示 DartFormat 的 PartTable 组件的实施例的结构和 PartTable 的 PartTable Record 组件的格式的模块图，同时单独地描述了 DartProcedure 的实施例的结构；

[0130] 图 15 的图示表示 DartRuntime 处理的流程图，用于 Dart 应用程序中通过 Gizmo 导出的对象的体系结构；

[0131] 图 16 的图示表示说明 DartRuntime 的 Process Event 处理部分的各方面的流程图；

[0132] 图 17 的图示表示说明 Dart 应用程序的实施例所使用的 DartDevice 之间的连接的模块图，其中 Dart 应用程序在一个 DartDevice 上启动，接着使用招募将其自己扩展到在其它 DartDevice 上运行，所述其它 DartDevice 以所表示的事件数据结构的形式交换消息，以协调其活动；

[0133] 图 18 的图示表示说明分级处理设置和 Gizmo 处理单元的执行顺序的模块图，其中 Gizmo 处理单元用在幻灯片放映或媒体显示应用程序的示例性实施例中；

[0134] 图 19 的图示表示根据本发明的实施例的 Dart 应用程序级错误恢复的实施例；

[0135] 图 20 的图示表示说明 DartEngine 执行的 BuiltinInstruction 和 OEM_BuiltinInstruction 处理的流程图；

[0136] 图 21 的图示表示假设的中微子检测器 / 移动电话例子的实施例，表示一个设备如何将其独特的能力展示给事先并不了解这些独特的能力的设备；

[0137] 图 22 的图示表示说明 DartPlayer 与 DartEngine 的实施例的特定实现方案的两个主要组件的示例性实施例的模块图，其中两个主要组件即便携组件，和非便携硬件抽象层组件；

[0138] 图 23 的图示表示 DartPlayer 的特定实施例的主环路处理流程的流程图；

[0139] 图 24 的图示表示在 DartPlayer 对 DartEngine 初始化函数调用期间进行的处理的流程图；

[0140] 图 25 的图示表示说明 DartPlayer 中进行的 DartRuntime 指令处理的流程图；

[0141] 图 26 的图示表示用于 DartPlayer 的文件系统指令处理的流程图；

[0142] 图 27 的图示表示说明 DartEngine 的实施例的非便携硬件抽象层组件的组件模块图；

[0143] 图 28 的图示表示说明文件系统的非便携 DartDevice 特定部分的图表，其中文件系统可以提供从 Dart 到文件的访问，但是不提供任何用于访问不被看做运行的 Dart 的保护性的沙坑 (Sandbox) 部分的文件的机制；

[0144] 图 29 的图示表示 DartDevice 的 DartSecruity 组件的实施例的模块图；

[0145] 图 30 的图示表示社会同步接触列表例子的实施例；

[0146] 图 31 的图示表示示例性 DartDevice 和其 SocialSecurity 组件的模块图，以及表示在编组前和编组后，旧的 DartDevice 和新的 DartDevice 的 SocialSecurity 内容的模块图；

[0147] 图 32 的图示表示用于访问特定虚指针地址的数据元素的示例性 Dart 虚指针。

[0148] 本发明的示例性实施例的详细描述

[0149] I. 概述和简介

[0150] 在一个方面，本发明涉及用于使设备在相似的和更重要的以及不同的设备和系统组之间有效地共享各种内容、控制、资源以及应用程序组的方法和系统。本发明的各个方面实施为“DartPlatform”。名称“Dart”旨在传达这种意思，即应用程序、数据，和 / 或其它混合形式的过程、程序、数据、概念、以及最可能宽意义下的其它信息被组合为智能化的数据，并且也旨在表示这种方法，其中程序，内容以及数据的完整智能包以 Dart，DartProcedures，以及 DartParts 的形式在各设备之间逐字地发出 (或传输)，以实现设备，设备资源以及设备能力的各个组中的高度的简洁性，有效性，稳定性安全性以及互操作性。

[0151] 而且，在这里所包含的描述中，应该理解术语 Dart 意味或意指本发明的一个特定实施例，术语互操作性用作要求保护的本发明的各个方面的一般术语，其中 Dart 是互操作性的特殊形式。

[0152] 本发明介绍了许多新的技术系统，设备，方法，以及计算机体系结构和程序特点，这些特点建立了在文章中在此以前没有描述过的新的示例。至少部分地因为技术的和计算机术语仍没有提供简洁的术语，通过这些术语自身可以完全从传统的元素中识别新的元素，所以该说明书频繁地使用术语“Dart”或其它的“Dart”的说法作为另一个术语的前缀

或限定词，例如在短语 Dart Procedures，Dart Parts 等中使用的。在一些情况下，两个术语被连接起来，例如 DartProcedures，DartParts，DartDevice，等。而且，应该理解更简洁的形式，如设备 (device)，部分 (part)，或程序 (procedure) 可以用于描述本发明的各方面。从说明书的内容中，所指的意义通常是明显的；然而，应该理解当描述本发明的一个方面时，被大写的单个字的形式，例如“DartProcedures”，等效于多个字的形式，例如“Dart Procedures”，“Dart proceduces”，或者甚至“procedures”。这也用于其它的“Dart”的说法，例如 Dart Parts，Dart Player，Dart Platform，等等。

[0153] 尽管 Darts 可以是任何组件并且可以组合所有组件，但是 Darts 在其最一般的形式中不是简单的数据，简单的程序，和简单的内容。可以将“Darts”考虑为软件、代码、数据，和 / 或内容、以及可以用于在各个设备中智能地汇编，储存以及分配其本身或其各部分以实现互操作性应用程序的目的代码数据和内容的特定的完整的数字式二进制包。本发明产生同类和异类设备之间的简单的，有效的，可靠的以及安全的互操作性。

[0154] 不幸地是，用于计算和信息系统和设备的现代词形变换在分离的和不同的操作系统 (OS) 组件，设备驱动程序，计算机程序应用程序，以及用户或其它数据组件的一般惯例中已经如此根深蒂固，从而在计算机科学领域中使用的一些共同接受的术语并没有严格地用于本发明的组件。因此在可能的情况下，当共同的计算和计算机科学术语的意义适当时，即使其意义并不准确，也使用这些共同的计算和计算机科学术语，并且当想要更加特定的意义以避免使用具有许多限定词的一般语言时，使用各种“Dart”表示和术语。

[0155] 首先描述本发明的实施例的一些组件，特点，以及优点，以将读者引导到 Dart Platform、系统、方法以及计算机程序元件。接着在该详细说明的其余部分进一步描述这些和其它组件、特点以及元素。应该理解，并不是本发明的每个实施例提供的所有组件，特点，或优点都能够容易地列举在几个段落中，因此下面的描述是在一些实施例中发现的组件，部件，以及优点的示例，包括一些可选的但是有益的组件和特点，并且不应看作是限制性描述。显而易见的是，本发明的实施例可以提供和利用或者不提供或不利用这里所描述的一些或许多特点，并且其它实施例将提供和利用这里所描述许多或大多数（即使不是全部的话）特点和组件。

[0156] 对于更高效的通用计算机的领域，用于操作系统、程序格式、编程工具，以及通信协议的现有方法得到极大的改进，其中的通用计算机可以使用充分可靠的电源以及可靠的，高速的，始终运行的通信协议。而且，计算机网络是相对静态的，很少必须为共同工作而被配置或重新配置。同时，假设将有知识丰富的人类系统管理员团体进行安装，配置，重新配置，更新，调整以及其它方式维护计算机，网络，操作系统以及程序。

[0157] 这些已有的方法并不适合共同应用于现在快速发展的专用设备领域，其中这些专用设备通常靠电池运行，具有有限的计算资源，通过低速不可靠的无线或电力线协议进行通信，并且需要动态地经常地重新配置以与其它设备共同工作，所述其它设备中许多是便携式的设备并且仅被间断地连接。如果不考虑增加的动态配置需要和设备多样性及其必须共同工作的能力，通常非常希望由不如系统管理员博学的人来使用和维护这些设备和相关的软件。

[0158] DartPlatform 包括一组新的发明的方法和软件的组件，以及可选的硬件，特别是关于现在快速发展的专用通信设备领域设计的生态系统。

[0159] 根据本发明的一个实施例，本发明的 Dart Platform(DP) 可以有益地包括下面的组件：

[0160] 1.DartInstructionSet- 互操作性指令集

[0161] 2.DartEngine- 便携式互操作性引擎

[0162] 3.DartFormat- 包括数据、内容、编码、程序、语义或其它信息的任何可能的组合的文件或位图格式，所说信息是最优地（或接近最优地）在任何设备或子系统上运行、存储，优化、复制和 / 或共享数据、内容、编码、程序、或其它信息所需要的，所述设备或子系统包含 DartEngine 和用于将 DartFormat 信息（通常以一组数字二进制的形式，例如位文件（或多个位文件）或位图（或多个位图））传送到设备以由 DartEngine 进行处理的的机制。

[0163] 4.Dartools- 一组用于创建 Dart Format 位图和文件的软件工具。Dart Tools 可以包括 DartComplier，DartLinker，以及 DartMasterPlayer，以及其它工具，其中每一个将在下面更详细地描述。

[0164] 5.Dart 或 Darts-Dartools 创建的比特位包，位图和 / 或文件实例，由包含 DartEngine 的设备处理，DartEngine 与 DartFormat 文件或位图格式一致。DartFormat 可以包含在任何设备或设备组上最优地运行、存储、优化、复制和 / 或共享数据 / 内容 / 代码所必须的组合的数据 / 内容 / 编码以及语义，其中所述设备包括 DartEngine 和用于将 DartFormat 位图传送到设备以由 DartEngine 处理的机制。Darts 是多个 Dart 的形式。

[0165] 6.DartProcedure 或多个 DartProcedure- 自含式的轻质程序 (lightweight procedure)（多个程序）和数据，由来自 DartInstructionSet 的指令序列与指令所操作的数据图像和数据图像值组成。

[0166] 7.DartPlayer- 被设计为在特定的设备子组上运行的软件程序，其中的设备子组利用 DartEngine 处理 Darts。

[0167] 8.DartMaster- 通过使用被称为 DartMasterPlayer 的特定 Dartool 处理 DartMaster，将 Dart 进一步优化为一个或多个有效的和 / 或专用的 Darts 或 Dart。

[0168] 9.DartFramework 或 DartObjectFramework- 一组源代码，用于定义构建 Dart 中的一组基类的数据和方法。DartObjectFramework 假设某个初始执行点，初始目标数据指针，以及管理处理编码和数据以及输入数据和编码的顺序的结构和语义组。这就包括了 DartRuntime 的设备外部设备运行时间部分，DartRuntime 还将外部设备运行时间扩展到内部设备运行时间，内部设备运行时间同步和串行化任何数量的被编组的设备上的运行时间行为。

[0169] 10.DartRuntime-DartFramework 或 DartObject Framework 有益地假设某个初始执行点，初始目标数据指针，以及管理处理编码和数据以及输入数据和编码的顺序的结构和语义组。这被称为“DartRuntime”的外部 - 设备部分。还存在内部 - 设备运行时间，用于假设在事件队列之间自动地串行化和同步事件结构示例，其中同步事件结构示例驱动对行为同步和协调的管理以及被编组设备之间的数据交换，并且其中的事件序列之一被每个设备的 DartEngine 维护。外部设备 DartRuntime 和内部设备运行时间共同提供了易于使用但高度稳定和有效的系统，以实现 Dart 的目的，其中 Dart 的每个可能不同的再现程序在一组被编组的设备上运行。

[0170] 在其它方面中，本发明被设计为改进任何类型的两个设备之间和任何类型的多个

设备之间的互操作性，并不被限制地解决间断地互连或总是互连的设备，系统，或子系统中互操作性的下述和其它问题：

[0171] 1. 当在不相似或不同的设备之间移动内容、应用程序以及控制时，显示、控制、代码、数据以及功能的适配和优化。

[0172] 2. 当设备用户的所有需要是起作用的内容（优选地，尽可能地起作用）时，无论程序、文件和或内容传递到哪里，为设备用户消除必须考虑程序、内容格式、驱动程序和/或文件的需要。

[0173] 3. 为用户消除必须指定或者甚至知道在一个或多个设备之间使用的设备连接（或多个连接）或通信的类型的需要。

[0174] 4. 允许在所有被激活的设备之间简单有效地共享内容、控制，和/或资源。

[0175] 5. 从昂贵的和复杂的计算机辅助的 X 线断层摄影 (CAT) 扫描到（小型处理器/存储器）电灯开关或其它简单的设备，激活任何设备或系统，以利用、评述和准予存取其功能、资源、能力和到其它连接的设备的需要。

[0176] 6. 将为实现上述一些或所有要求采用传统的非程序性方法而产生的大量的开发工作减少为能够以数量低的多的开发项目和成本容易地实现的水平。

[0177] 7. 足够地轻质（在代码大小、执行逻辑，和存储器方面）以适合于低端、成本受限，和/或电源受限的设备。

[0178] 8. 在有意义的大多数任何方式下，例如通过包括对下述一个或任何组合的紧密联系的本地支持，确保任何数量的 DartDevices 能够无缝地与所有 DartDevices 进行互操作：

[0179] (a) 无论何时只要设备硬件可以支持，都具有动态的多媒体富界面 (rich interface)，如果设备硬件不支持则具有分级的更少的富界面；

[0180] (b) 设备电源管理；

[0181] (c) 设备发现；

[0182] (d) 服务发现；

[0183] (e) 资源发现；

[0184] (f) 将 Dart 应用程序与需要相隔离，以了解各种通信物理层和协议层的详细情况；

[0185] (g) 将 Dart 应用程序与需要相隔离，以了解物理显示格式的详细情况；

[0186] (h) 将 Dart 应用程序与需要相隔离，以了解关于与运行在多个类似或不同的设备上的应用程序保持同步的详细情况；

[0187] (i) 请求、检索以及运行优化的控制面板以本地地运行、但实际控制从中检索所说控制面板的那个设备；以及

[0188] (j) 分别地加载、运行以及优化所产生的 Dart 内容，作为一个或多个双亲 Darts 的外部设备 DartRuntime 环境的子女 Dart 扩展。

[0189] 9. 消除对预先存在于除启动互操作性应用程序的设备（原始设备）外的任何设备上的程序、数据或内容的需要，其中的互操作性应用程序实施在初始设备上的 Dart 中。

[0190] 10. 允许设备根据设备能力和限制的所有或一些子集，有效地共享其资源，而不需要任何设备作为主机或从机。

[0191] 11. 允许应用程序的代码、数据、内容及其混合（在 Darts 中实施）以一方式动态地将其自己扩展到连接的设备，所述方式允许连接的设备存储应用程序使用和进一步无限地传输到其它设备上，即使在原始和初始设备不再连接后也是如此。

[0192] 12. 通过在一个包（或包集）中包括互操作性应用程序的所有代码、数据和内容来改进多设备操作的可靠性，所述包接着按照要求将其扩展到其它设备上，从而消除与分别地混合和匹配生成的和分配的应用程序或协议软件的执行、代码、数据或内容相关联的问题，其中这些应用程序或协议软件的执行、代码、数据或内容将以其它方式被分别地生成和 / 或分配到每个设备上。

[0193] 13. 以简单的、可靠的和安全的方式实现上述所有要求。

[0194] 参照图 8 所示，表示了运行在蜂窝电话上的示例性 PrintPicture Dart，其中的蜂窝电话招募通过网络连接的存储设备以将其用作图片源，并且招募打印设备以用作执行图片打印的目的设备。假设所有的设备包括已经被移植到每个设备的 DartPlayer。

[0195] 蜂窝电话上的 Dart 包含三个再现程序 (R1、R2 以及 R3)，当每个再现程序运行在 DartPlayer 上时，都能够作为单独的可执行的图片提供服务。再现程序 R3 在蜂窝电话上运行，并且在此之前通过发送 DartProcedures 在候选设备上执行，来实现其它两个设备的招募，这就确定特定的通过网络连接的存储设备将作为最佳的图片源发挥作用，打印机设备将最佳地作为打印图片的目的设备起作用。

[0196] 接着蜂窝电话上的再现程序 R3 为仅包含再现程序 R1 的 Dart 生成图像，其中 R1 包含用于从通过网络连接的存储 (NAS) 设备识别和检索图片的事件处理代码。接着包含再现程序 R1 的 Dart 作为 RUN_DART 类型事件的一部分被发送到通过网络连接的存储设备。当 NAS 设备上的 DartEngine 处理 RUN_DART 类型事件时，加载再现程序 R1 并开始执行，其中再现程序 R1 处理请求存储在 NAS 设备上的信息或图片的任何同步事件。类似地，形成处理作为 PRINT_PICTURE 类型事件一部分的图片打印的再现程序 R2，并进行发送以在选择的打印机设备上运行。

[0197] 注意到三个再现程序都是从相同的最初仅存在于蜂窝电话上的 PrintPicture Dart 生成的。由于实际上应用程序与其自身的各部分进行对话，而不是独立地执行、移植，以及分配传统的互操作性方法所需要的组件，这就确保了兼容性的高度可能性。

[0198] 进一步注意到，再现程序还共享代码、数据和 / 或内容，并理解 PrintDart 应用程序专用的相关的事件类型集。R3 再现程序能够智能地将 PrintPicture Dart 的各部分扩展到 DartDevices，即使事先并不了解其它两个 DartDevices 也是如此，并且 DartDevices 事先不了解蜂窝电话或 PrintDart。

[0199] 现在，在蜂窝电话上运行的再现程序 R3 能够向事件发送信号以通过现在在三个设备上或三个设备之间建立的事件驱动的 DartRuntime，来控制被招募的 NAS 和打印机设备之间的图片选择和打印。

[0200] 此外注意到，可以根据蜂窝电话和 NAS 设备的共同协议来使用协议的任何组合，并且可以独立地选择这些协议的任何组合以使用被招募的打印机设备的任何共同协议。此外，由于所有的设备包括作为 DartPlayer 的一部分的移植的 DartEngine，因此即使设备在不同的处理器上运行不同的操作系统，设备也能够进行互操作。

[0201] 由于本发明包括具有大量组件的系统，因此这些组件可能具有联系密切的复杂的

相互关系，首先以概述的方式描述一些组件，从而理解为何可能出现一种组件以及组件如何工作，如何与其它组件相互作用。接着，关于一种组件与其它组件的相互作用将更详细地描述每个组件。在本发明所包括的互操作性中，对所属领域的状况进行明显的改进要求产生大量的新型的软件生态系统，类似于从数据传输功能变换到现在广泛使用的面向对象的方法。为了更便于描述发明、系统，以及数据和代码结构，首先引入一些新的术语来描述本发明的实施例的原理、概念、特征以及组件。

[0202] 在一个实施例中，本发明本身实现了下述九个实现方法，其中一些是可选的，但是有益地包括：招募、互操作性指令集、再现、创建法、垂直分层、线性任务分配、社会同步、社会安全性，以及虚指针。

[0203] 设备“招募”可能包括设备交互作用模型以及相关的结构和方法，设备“招募”是现有的客户机/服务器和对等模型的有益的替换方案。“互操作性指令集”是基于便携的引擎的指令集，用于在设备上提供了共同有效的可执行环境，并且有创造性地提供了程序性机制以将设备的独特的能力展示给其它设备。“再现”是指实现应用程序、数据、内容，和/或其它信息的容易测试和有效适配的结构和方法。在一些方面中再现可以包括将代码、数据和内容组有效地分为独立的可执行图像，“再现程序”，它是在多个设备上智能地生成和分配的。“创建法”指的是这种结构和方法，即实现在连接的和间断连接的设备上以不同的形式动态有效地生成和分配应用程序、数据、内容，和/或其它信息。“垂直分层”实现高效地和有效地执行特性，这些特性就其本性而论涉及在工具、应用程序、框架、引擎和指令集级的密切合作。“线性任务分配”实现设备处理单元之间的处理器控制和设备资源的简单的确定的流动，其中处理单元可以被容易地重新配置为单级结构，该单级结构包括处理单元编译成的 Dart，分别编译的 Darts，以及甚至在其它设备上运行的 Darts 或再现程序。“社会同步”是指用于在任何数量的设备上同步数据或内容的发明的有效的系统和方法，其中具有最少的或根本没有用户的参与。“社会安全性”是指用于简单地建立和维护具有进行互操作的适当的授权的设备组的有效的系统和方法，其中包括最少的用户参与。可以以计算机程序代码段的形式实现这些和其它组件，其中的代码段包括用于在设备的处理器中执行的可执行的指令。而且，这些组件的元件可以被实现为硬件和或硬件与软件的组合和/或固件和或微代码。

[0204] 另一个提供的可选但是有益的实现方法被称为“虚指针”，该方法提供了对具有一些有益特征的虚拟内存的各种和重要的改进，例如包括：

[0205] (a) 多个大的独立的地址空间。

[0206] (b) 对真正存储页面的数量的应用程序指定的控制。

[0207] (c) 对真正存储页面大小的设备指定的控制，以与存储设备性能特征相匹配。

[0208] (d) 程序员不需要预测或管理数据结构或列表所需要的存储器的数量。

[0209] (e) 以不依赖于页面大小或页面数量的形式自动地存储数据。

[0210] (f) 从更大的并且可能是更慢的数据存储器有效地缓存数据，最少数量的非常快速的 RAM 允许运行应用程序，就如同它们具有比它们实际上拥有的大得多的 RAM 存储器。

[0211] (g) 用于索引的数据库操作的简单和有效的基本架构，其中数据和索引被保存在不同的虚指针地址空间中。

[0212] 现在以概述的方式，已经描述了本发明的许多方法、组件，和特征，现在将注意力

转向主要的方法、结构，和方法的实施例的详细描述。应该注意许多程序和方法可以由一个或多个计算机程序或计算机程序产品执行，其中的计算机程序产品可以在通用或专用处理逻辑上执行，所述通用或专用处理逻辑例如是能够对计算机程序代码进行执行或操作的微控制器、处理器、微处理器、中央处理单元 (CPU)、或其它处理硬件或逻辑，所述通用或专用处理逻辑可以是软件、固件，或两者组合的形式。

[0213] 所提供的部分标题仅旨在将读者的注意力引导到描述一个特殊的方面或方法的说明书部分，但是应该知道在整个说明书和附图和权利要求中描述了所有方法和结构的各个方面，在子标题部分中包括或不包括本发明的任何方面并不意味任何限制。II. 招募互操作性模型

[0214] “招募”包括在本发明的整个实施方案中所体现的设备交互模型和方法。它是所属领域现行状态中使用的客户机 / 服务器和对等设备交互模型和方法的有益替换方案。招募模型和方法使用在所有设备上运行的共同的程序性环境，其中的设备将进行互操作，或在预期可能的互操作时针对资源检查所说的设备。在本发明的一个实施例中，通过一个指令集来提供该共同的程序性环境，其中的指令集例如 Dart 指令集（例如，DartInstructionSet)，或满足这里描述的共同程序性环境或其等效环境的要求的任何其它名称的指令集。参照图 5 所示，表示招募程序的实施例的流程图，招募程序提供了用于软件应用程序招募并且其后在多个设备组上有效地运行，就如同这些设备是具有所有设备合成的资源的一个设备的方法。同时参考用于共享的幻灯片放映 10000（图 6) 的招募的例子，并执行从幻灯片放映 20000（图 7) 设备招募（或者更简单地，“招募”）远程打印一个或多个幻灯片的例子。招募提供用于软件应用程序招募并在集群、组，或其它多个设备和 / 或系统上有效地运行，就如同这些设备是具有所有设备合成的资源的一个设备的方法。

[0215] 资源实际上可以是任何硬件、软件、固件、通信能力、配置、数据或数据集、设备或系统所拥有的或可存取的能力。例如，资源可以是处理器、CPU、存储器、接触列表、声音输出设备、显示类型、DARTs、图片，或任何其它的程序性、结构性，或信息事项，而没有任何限制。例如能力可以是对列表进行分类的计算机代码、对 MPEG 文件进行译码、与 Bluetooth™ 设备通信的硬件或软件，等。招募程序具有智能（由于这里所描述的其程序的和支持的框架、平台、引擎等)，独立于初始设备，关于可招募的或被招募的的设备的合适性进行复杂的决定，以实现发送程序的应用程序的目的，或者实现应用程序的目的一些部分。

[0216] 在一个实施例中，初始设备首先通过任何数量的通信协议，将检查程序（一个或多个）形式的消息以共同的可执行形式，从其本身（作为源）或初始设备发送或广播到任何数量的可到达的设备，参照图 6 10011 和图 7 20011 所示。初始设备形成并发送该消息，以试图发现具有所需要的资源或能力的其它设备，通过一个连接或多个连接将结构化的，编码的，或在共同的程序环境中以其它方式执行的一个或多个程序，发送、传输、广播，或以其它方式传输到其它设备（在传输时已知或未知)，这些其它设备也包括了共同的程序环境。当初始设备发送或广播消息时，不需要了解哪些设备是可到达的。其它的设备是否可到达例如依赖于与可能的候选被招募的设备组相比时，初始设备拥有的通信信道和 / 或协议。

[0217] 可选地，该源设备被称为初始设备，因为它发起了交互作用（参考图 6 10100 和图 7 20100)，源设备可以是招募程序的来源，招募者设备是试图招募其它设备的设备，其中的

其它设备可以被称为被招募的设备，目标设备，目的设备，或简单地不同于初始设备的其它设备。

[0218] 例如，如果初始设备包括 Bluetooth™ 无线通信链接、红外通信链接，以及 IEEE 802.11(a)、(b) 或 (g) 通信链接并通过每个这些信道广播消息，则只有被配置为通过这些通信链接接收通信的候选的可招募的设备可以接收招募消息。其中，只有那些提供操作环境、理解并能够执行检查程序的设备会进行响应。注意到即使在这种可能进行响应的设备中，另外的可招募的设备的用户可以例如根据安全性设置，选择性地或全部地中断这种招募或询问检查程序。

[0219] 第二，检查程序接着在进行响应并被发现的每个设备上执行其指令，以识别所需要的资源和能力和 / 或设备的相关资源和能力组，或者设备可以得到的用于实现应用程序目的或部分目的的资源和能力组。在一个实施例中，使用指令集轮廓或得到轮廓指令来执行该检查，例如 Dart 指令集轮廓指令（例如，DartInstructionSet PROFILE_INSTRUCTION），这就使得存取或调用候选可招募的设备的硬件抽象层 (HAL)，以存取在设备专用的硬件抽象层（例如参照图 27 的硬件抽象层 HAL 652）中存储或计算的关于特定设备和其资源和能力的信息。

[0220] 第三，检查程序通过通信信道和协议将程序、数据、内容或其它信息返回到初始设备，以表示其已接收到消息（参照图 6 10012 和图 720012）。进行响应的设备可以识别和存储通信信道和接收的协议的表示，并且可选地识别和存储接收招募消息的设备的身份，，或者可以广播可能被初始设备接收的响应。

[0221] 在一些实施例中，进行响应的设备仅回答发起者关于特定资源或资源组（例如彩色打印能力）的可用性的查询，而在其它实施例中，检查程序可以识别并响应特征、能力，以及资源的完整列表。通常优选的是 :简单的“是，我具有所需要的资源”（响应是单个比特或字节或字），从而将通信量降低到最小。可选地，可以使用识别一个或多个资源类型或类或资源子类的代码。在一个实施例中，检查程序将关于它们所运行在的设备上的资源和能力的信息返回给源设备。例如，该信息可以是静态数据结构、程序、Dart、自由形式标签语言的形式，或任何其它的信息形式。在图 6 的幻灯片放映例子 10000 中，参照返回图 610012 的“是”的响应，以及 MIPS 中的处理器速度和被招募的 dart 设备图 6 10200 中的优选的显示分辨率，以及在图 7 的幻灯片打印例子中，返回图 7 20012 的“是”的响应，以及被招募的 dart 设备图 7 20200 的优选的打印机分辨率。

[0222] 第四，初始设备上的应用程序代码收集可能包括程序、数据、内容的所有返回的信息，或来自所有进行响应的可到达的设备的其它信息，执行任何接收到的程序，并对其进行检查，以决定如何利用可到达的设备和其资源，来实现应用程序的目的。

[0223] 第五，初始设备上的应用程序专用的代码根据第四步骤中的决定，按照需要将代码、数据、内容，或任何其它信息传送到每个可到达的设备（参照图 6 10020、10021 以及图 7 20020、20021 所示）。可以以单个共同的形式发送应用程序代码、数据、内容，或其它信息，或者可以根据本发明的实施例对其进行定制，例如通过使用关于再现程序和创建法描述的方法，或者根据其它方法和技术。

[0224] 第六，可选地但是有益地递推地使代码、数据和内容在初始设备和初始可到达的设备上传播，可到达设备上的应用程序代码、数据和内容进一步按照需要递推地在初始可

到达设备组上使用第一到第五步骤，其中初始可到达的设备组现在按照需要作为初始设备（第二或后续的初始设备），将应用程序按要求扩展到其它可到达的设备，直到达到了实现最初的应用程序目的所需要的所有希望的设备和资源为止，从而有效地形成设备和其相关资源的完整组。应该知道，如果设备在第一循环中就具有需要的资源，则可能就不需要第二或后续的招募循环了。在另一方面，如果最初不能找到需要的资源，则就可能需要第二或后续的递推的招募循环。递推的招募也增加了第一循环被招募的设备作为桥梁或转换器的可能性，从而通过第一循环被招募的设备起作用的最初的发起者能够与第二循环被招募的设备进行通信（例如，将 Bluetooth 通信转换为硬连线的网络连接）。

[0225] 第七，根据初始应用程序的要求，使现在从设备组或在设备组中分配的代码、数据和内容执行需要的操作和资源存取，以通过执行代码和交换任何需要的代码、数据、内容或其它信息，来执行或协调在所述设备组上执行的操作，来实现初始应用程序的目的，从而实现最初的应用程序的目的。

[0226] 这种分配可执行的代码、数据，和内容的步骤可以是应用程序自身正在进行的操作 - 直到这一时刻为止，即最初关注的过程形成设备组传播数据、代码和内容，以建立设备组的各个成员，与这些成员一起实现其相应的应用程序的目的。该过程继续实现应用程序或任务的目的。在幻灯片放映的例子中，幻灯片（实际上是数字图像或数据集）被添加到联合的幻灯片放映中，或者幻灯片可以被翻页或顺序地显示在所有设备上，以在所有设备上进行浏览。这里在除该初始招募阶段的其它地方描述了用于继续互操作的程序。

[0227] 这就完成了初始的招募阶段，并为招募者（发起者）和被招募者（其它编组的设备）提供了进行互操作和共享资源的机会，如这里其它地方所描述的以及图 6 10031、10032、10033 中所表示的。

[0228] 接着通过事件 800（参照图 17 所示）可以同步设备上的应用程序的操作，其中如这里其它地方所述，事件 800 驱动应用程序的处理，并有益地通过事件 800 得到所有到应用程序的输入，将其放置在所有招募和被招募的设备的队列中，其中招募和被招募的设备被标记为针对其事件类型是同步的。下面（包括参照图 17）更详细地描述了事件和事件排队。

[0229] 参照图 17 所示，事件 800 可以是数据结构示例，在图 17 的 800 中表示字段，在该实施例中包括事件类型 801、事件参数 802，以及事件相关文件 810，其中事件相关文件 810 可以包括程序 811、内容 812，和 / 或 Dart813。这些事件 800 可以提供输入、通信数据和语义，并当它们经过运行时间时实现同步功能，例如，经过 DartRuntime 8000（参照图 15），Gizmo 体系结构 10000（参照图 18），或者沿着被标记为 451-x 的线在 DartDevices400 之间（参照图 17）。

[0230] 事件 800 可以携带任何类型（一种或多种）或数量的数据参考文件 810。优选地参考开放文件。典型地，文件包含一个或多个在另一个 DartDevice 上运行的 DartProcedures，将 Dart 应用程序延伸到另一个 Dart Device 上的完整的 Darts，包含到另一个设备的控制面板的 Dart，或者一般的列举或输入信息，其中所述信息不适合图 17 中 800 所示的事件的其它参数。这里在其它地方更详细地描述了这些和其它的 Dart 类型和一般的列举或输入信息。在图 15 所述的 DartRuntime 流程图 8000 和图 16 的引擎事件处理内置功能流程图 5000 中更详细地表示了事件的过程。因为在本发明的一个实施例中，发送事件的结构的和方法的基础随着事件一起自动地发送相关文件，因此与事件 800 相关

的文件 810 可以将与事件相关的文件看作与事件相分离，或者优选地作为事件本身的一部分。因此，在至少一个实施例中，在将事件置于事件接收者的事件队列之前，在发送方的文件被通信基础设备备份到接收方的文件，使用与接收方设备上的备份文件图像相关的文件标识符（例如，字段）可以读取该备份到接收方的文件。在发送方或接收方可以使用共同的或不同的文件名或文件标识符。在运行 Dart（例如，RUN_DART 类型事件）或运行程序（例如，RUN_PROCEDURE 类型事件）的情况下，当事件达到队列的最前面或开头从而是队列中的下一个时，引擎将使用将被执行的事件中的字段产生现在可以被读取的 DartProcedure 或 Dart。通常可能存在被 Dart 引擎自身处理的事件，即使当目前没有运行应用程序时也是如此。在一个实施例中，驱动事件队列的总是运行的 Dart 或者是建立在引擎中的空闲的程序（例如，DartIdleProcedure），所说引擎保持调用引擎事件处理程序来保持通信继续下去。这本质上是保持运行的循环，直到出现要被处理的事件为止。当应用电源管理（这里在其它地方描述）时，可以执行用于停止接着唤醒或重新开始该循环的各种技术。

[0231] 因此现在应该知道，招募模型、方法，以及相关结构执行特定的设备、服务，以及资源发现，以识别所需要的设备，接着通过使用事件数据结构，例如事件 800，向设备发送授权程序和信息，并智能地和有效地形成设备组，接着再次使用事件数据结构事件 800 进一步协调设备组，以实现 Dart700 或最初运行在源设备上的应用程序的初始目标。

[0232] 图 10 表示招募如何使得连接的设备组作为单个设备进行工作。一个最深刻的影响是新的互操作性设备和基于 Dart 的应用程序的执行和测试仅需要与设备数量 N 成比例的工作。需要分别地执行和向所有需要进行互操作的设备分配组件的传统的静态和程序性方法需要与 N^2 成比例的速率增长的工作。

[0233] 在包括例子的本说明书的其它地方描述了招募和招募互操作性模型的其它方面，包括被招募的设备之间的事件的串行化和同步。下面还说明了招募互操作性模型的一些特定的示例性实施例。

[0234] 在一个实施例 (1) 中，本发明提供了一种用于运行在源设备上以招募一组设备的软件应用程序的方法，所述方法包括：通过至少一个通信链接，向至少一个可到达的设备发送能够运行以找到具有所需要的资源或能力的设备的检验程序，其中的可到达的设备不同于初始的源设备，所述检验程序包括以可执行的形式编码的检验程序指令，所述可执行的形式是初始的源设备和检验程序将到达的设备所共用的；通过通信链接，在初始设备上从每个可达到的设备直接或间接地接收回答响应；通过在初始设备上执行的程序，分析来自所有响应的可到达的设备的接收的回答，以确定识别初始设备和响应的可到达的设备的能力和资源的组合的利用计划，以最佳地实现软件应用程序的目的；以及通过在初始设备上执行的应用程序，将可执行代码，数据，内容，和 / 或 Dart 的至少之一分配给每个可到达设备的至少之一，根据识别的利用计划将其中的可到达的设备识别为具有需要的资源或能力。

[0235] 在另一个实施例 (2) 中，本发明提供了一种用于招募一组设备的方法，所述方法包括：通过通信链路在候选设备上接收并执行检验程序，所述检验程序可被操作以确定接收的可到达的候选设备是否具有另一个招募设备所需要的资源或能力，所述检验程序包括以可执行的形式编码的检验程序指令，其中的可执行的形式是接收设备和招募设备都已知的；为接收的检验程序识别源设备，并关于接收的可到达的设备是否可存取被识别为初始

源设备所需要的资源或能力或资源和能力组，发送对源设备状态和信息的回答；以及在源设备确定可到达的设备具有形成一组以实现软件应用程序的目的的资源或能力的情况下，从源设备，招募设备，或其它候选设备接收可执行代码，数据，内容，和/或Dart中的至少之一。

[0236] 在另一个实施例(3)中，本发明提供了一种用于招募一组设备的方法，所述方法包括：(a)通过至少一个通信链接，从初始源设备向至少一个可到达的设备发送能够运行以找到具有所需要的资源或能力的设备的检验程序，其中的可到达的设备不同于初始的源设备，所述检验程序包括以可执行的形式编码的检验程序指令，所述可执行的形式是初始的源设备和检验程序将到达的设备所共用的；(b)在每个可到达的设备上接收并执行接收的检验程序，以识别是否存在至少一个初始源设备所需要的资源或能力的可到达的设备；(c)至少当可到达的设备存取被识别为初始源设备所需要的资源或能力时，向初始源设备发送回答；(d)通过通信链路从每个可达到的设备直接或间接地接收回答；(e)通过在初始设备上执行的应用程序，分析来自所有响应的可到达的设备的接收的回答，以确定识别初始设备和响应的可到达的设备的能力和资源的组合的利用计划，以最佳地实现应用程序的目的；以及(f)通过在初始设备上执行的应用程序，将可执行代码，数据，内容，和/或Dart的至少之一分配给每个可到达设备的至少之一，根据识别的利用计划将其中的可到达的设备识别为具有需要的资源或能力。

[0237] 在另一个实施例(4)中，本发明提供(3)所述的方法，所述方法进一步包括至少一个可到达的设备接收分配的可执行代码数据，内容，和/或Dart的至少之一。

[0238] 在另一个实施例(5)中，本发明提供(3)所述的方法，所述方法进一步包括：与可执行代码数据，内容，和/或Dart的至少之一所分配到的至少一个可到达的设备合作，以实现初始设备应用程序目的的至少一部分。

[0239] 在另一个实施例(6)中，本发明提供(3)所述的方法，所述方法进一步包括：每个可到达的设备作为第二初始源设备，在初始设备和可到达的设备上传播可执行的代码，数据，内容，和/或Dart，可选地还要传播存在于可到达的设备上的应用程序代码，数据，内容，和或Dart，并且如果需要的话，递推地传播到以前可到达的和编组的设备可到达的其它设备，以按需要将应用程序扩展到其它可到达的设备，直到实现了需要的设备的所有或预定标准，和实现初始设备应用程序的目的所需要的和希望的资源或能力为止，以有效地形成更大型的完整的设备组；根据初始设备应用程序的需要和希望，从初始设备和可到达的设备组和在该组中分配可执行的代码，数据，内容，和/或Dart，以执行实现初始设备应用程序的目的所需要或希望的操作和资源存取，这是通过执行代码和交换实现和/或调整操作所需要的任何可执行的代码，数据，内容，和或Dart实现的，其中的操作在设备组上执行，以实现初始应用程序的目的。

[0240] 在另一个实施例(7)中，本发明提供(3)所述的方法，其中初始源设备直接从初始设备发送招募消息的初始可到达的设备接收回答，或者以顺序或递推的方式，通过一个或多个中间可到达的设备间接地从另一个可到达的设备接收回答。

[0241] 在另一个实施例(8)中，本发明提供(3)所述的方法，其中当可到达的设备不存取被识别为初始源设备所需要的资源或能力时，也发送到初始源设备的回答。

[0242] 在另一个实施例(9)中，本发明提供(3)所述的方法，其中到初始源设备的回答是

简单的参数，其标识为：可到达的设备或者存取（例如，真或逻辑“1”）或者没有存取（例如，假或逻辑“0”）被识别为初始源设备所需要的资源或能力。

[0243] 在另一个实施例（10）中，本发明提供（3）所述的方法，其中到初始源设备的回答包括DartEvent，一部分Dart，数据，内容，可执行的程序，Dart，多个Dart之一，以及其任何组合。

[0244] 在另一个实施例（11）中，本发明提供（3）所述的方法，其中到初始源设备的回答包括回答的数据或内容，该回答的数据或内容标识特定的可到达的设备通过通信协议到初始设备的资源和/或能力。

[0245] 在另一个实施例（12）中，本发明提供（3）所述的方法，其中检验程序包括检验至少一个初始源设备所需要的被特定识别的资源或能力，以实现至少部分地在初始源设备上执行的应用程序任务的目的的指令。

[0246] 在另一个实施例（13）中，本发明提供（3）所述的方法，其中所述的方法至少部分地被编码为计算机程序产品的软件应用程序，其中计算机程序产品用于在一组设备上招募和有效地运行应用程序。

[0247] 在另一个实施例（14）中，本发明提供（3）所述的方法，其中所述的方法有效地连接并接着允许控制招募和被招募的设备的操作，好像被招募的的设备是具有所有招募和被招募的设备的组合资源的设备。

[0248] 在另一个实施例（15）中，本发明提供（3）所述的方法，其中回答包括下述中任何之一：无回答，数据或内容回答，任何数字编码的信息，一个或多个程序，设备将是有用的指示，回答的事件，通过其文件有效载荷包含任何数量的数据或数据组的回答事件，回答程序，Dart，包括文本名称和设备的描述从而可以从初始设备上的菜单中选择使用哪个（些）设备的回答事件，一组可执行代码的至少一个示例的特定包的标识符，存在于源设备上和能够在源设备上生成的数据和内容，最适合于在设备上运行的再现程序或再现程序组，或这些的任何组合。

[0249] 在另一个实施例（16）中，本发明提供（3）所述的方法，其中至少一个资源或能力从包括下述的组中选择：(i) 可用的资源或特别需要的资源；(ii) 可用的能力或特别需要的能力；(iii) 一个或多个相互关联的可到达的设备的资源和能力组，或可到达的设备可用以实现应用程序的目的的那些资源和/或能力；以及 (iv) 以上这些的任何组合。

[0250] 在另一个实施例（17）中，本发明提供（3）所述的方法，其中资源或能力包括从包括下述的资源或能力组中选择的被识别的能力的至少之一：被识别的数据操作软件，被识别的信息处理软件，被识别的计算软件，被识别的图像处理软件，被识别的通信软件，被识别的通信硬件，被识别的介质，被识别的数据组（多个数据组），被识别的内容，被识别的程序或多个程序，被识别的配置信息，被识别的图形加速硬件或软件，被识别的无论是临时的或不是临时的（永久的）存储介质，被识别的打印能力，被识别的传真能力，被识别的扫描能力，被识别的用户接口设备（无论输入或输出或既是输入又是输出），对其它设备的资源的存取（由于该资源设备可以进行通信并且其它设备可以在永久的通信链中进行通信），以及其中两个或多个的任意组合。

[0251] 在另一个实施例（18）中，本发明提供（3）所述的方法，其中以共用的可执行形式的检验程序包括由Dart顺应指令集（DartInstructionSet）或任何其它互操作性指令集形

成的至少一个检验程序，该Dart顺应指令集在Dart指令可兼容的引擎(DartEngine)中实施。在另一个实施例(19)中，本发明提供(3)所述的方法，其中至少一个通信链接包括任何数量的通信链接，信道，和/或协议，其中的协议包括任何数量或任何组的同类或异类的通信协议，而无论是有线的或无线的，无论是永远可用的或间断可用的。

[0252] 在另一个实施例(20)中，本发明提供(3)所述的方法，其中同类和异类通信链接，信道，和协议被识别的硬件抽象层实现方案支持，其中的硬件抽象层实现方案是在任何两个或多个通信设备上运行的播放器部分。

[0253] 在另一个实施例(21)中，本发明提供(20)所述的方法，其中被识别的硬件抽象层实现方案包括作为DartEngine的组件的Dart硬件抽象层实现方案。

[0254] 在另一个实施例(22)中，本发明提供(3)所述的方法，其中至少一个通信链路和通信协议用于发送运行程序类型的事件，事件的事件标识符参考用于识别在可到达的设备上运行的程序的文件。

[0255] 在另一个实施例(23)中，本发明提供(22)所述的方法，其中事件包括DartEvents，运行程序类型事件包括Dart RUN_PROCEDURE类型事件。

[0256] 在另一个实施例(24)中，本发明提供(3)所述的方法，其中参考用于识别在可到达的设备上运行的程序的文件的事件标识符包括事件的文件标识符，其中文件标识符是指包括在可到达的设备上运行的程序的图像的文件。

[0257] 在另一个实施例(25)中，本发明提供(24)所述的方法，其中文件包括遵循DartFormat的Dart顺应文件(DartFile)，程序的图像包括Dart Procedure(DartProcedure)的二进制数据图像。

[0258] 在另一个实施例(26)中，本发明提供(3)所述的方法，其中检验程序包括DartProcedures或完整的Darts。

[0259] 在另一个实施例(27)中，本发明提供(3)所述的方法，其中检验程序作为与事件相关联的文件发送，检验程序作为与事件相关联的文件被可到达的设备接收，这使得检验程序开始在可到达的设备上的执行。

[0260] 在另一个实施例(28)中，本发明提供(3)所述的方法，其中检验程序包括DartProcedure。

[0261] 在另一个实施例(29)中，本发明提供(3)所述的方法，其中通过使用指令集轮廓指令，确定包括可到达的设备的基本资源和能力的资源和能力。

[0262] 在另一个实施例(30)中，本发明提供(29)所述的方法，其中指令集轮廓指令包括Dart顺应指令集(DartstructionSet)的Dart顺应轮廓指令(DartProfileInstruction)。

[0263] 在另一个实施例(31)中，本发明提供(3)所述的方法，其中每个可到达设备中的检验程序的执行根据再现确定规则确定包含在应用程序中的初始Dart的最佳再现，其中的再现确定规则被发送到每个特定的可到达的设备，并作为回答的数据的一部分送回确定的最佳再现的标识符。

[0264] 在另一个实施例(32)中，本发明提供(31)所述的方法，其中再现确定规则被包含在至少一个适于执行任何复杂度的任何需要的计算的程序中，并通过轮廓指令访问任何需要的轮廓信息，以确定可到达设备的资源，能力，和/或状态。

[0265] 在另一个实施例(33)中，本发明提供(31)所述的方法，其中检验程序执行过程通

过参考定义再现的顺序的规则从多个再现中确定最佳再现，并检查每个可到达的设备以确定是否以预定义的再现优先顺序满足多个再现中每一个的所有要求，直到在排序的多个再现中发现第一个满足所有再现的要求的再现。

[0266] 在另一个实施例 (34) 中，本发明提供 (3) 所述的方法，其中检验程序（多个程序）使用事先已经理解的通信协议，通过通信链路向初始设备返回 Darts，程序，数据，或内容。

[0267] 在另一个实施例 (35) 中，本发明提供 (3) 所述的方法，其中回答包括至少一个回答的程序，数据，或内容和其中任何一个或其组合：完整的 Darts，DartParts，DartProcedures，或 DartEvents，并且可以返回其中一个或任何组合，带有或不带有相关联的事件文件。

[0268] 在另一个实施例 (36) 中，本发明提供 (3) 所述的方法，其中回答包括返回的程序，数据，内容，和 / 或 Dart 中至少之一，并且可选地包括显示发生错误的应答代码，表示或者已经发生特定的错误或者已经发生非特定的错误的错误代码，并且错误代码可选地包括在改正或传输特定的错误和 / 或错误性质中使用的信息。

[0269] 在另一个实施例 (37) 中，本发明提供 (3) 所述的方法，其中应用程序代码至少是 Dart、DartProcedure、或可以在初始设备或初始设备存取的设备上执行的任何形式的程序中之一，其中初始设备对所述设备进行存取以开始传输或执行程序并利用该执行的结果。

[0270] 在另一个实施例 (38) 中，本发明提供 (3) 所述的方法，其中应用程序代码包括 Dart、DartProcedure，或可以在可到达的设备（多个设备）上执行或另外地将信息传送到可到达的设备（多个设备）的另一个程序格式，以在可到达的设备上执行，而不考虑程序格式是否利用 DartInstructionSet。

[0271] 在另一个实施例 (39) 中，本发明提供 (3) 所述的方法，其中在用于执行应用程序的操作时间或确定的时间段内，被招募的的设备组可以按照需要动态地扩展该组以包括其它可到达的设备，或减少设备组以排除可到达的设备。

[0272] 在另一个实施例 (40) 中，本发明提供 (3) 所述的方法，其中通过发送 Darts、DartProcedures、数据、内容，或其它信息，或其任何组合中的至少之一，来实现分配，无论事件中字段参考的信息是否随事件被发送或作为事件的一部分被发送，这些都作为 dart 事件 (DartEvents) 的一部分包括在一起。

[0273] 在另一个实施例 (41) 中，本发明提供 (3) 所述的方法，其中根据初始应用程序的需要，从设备组并在设备组中分配的代码、数据，以及内容，执行所要求的操作和资源存取，以通过执行代码和可选地交换任何额外的或不同的代码、数据，以及内容来实现初始应用程序的目的，其中的执行和交换是实现或协调将在设备组上执行的操作以进一步实现初始应用程序的目的所必须的。

[0274] 在另一个实施例 (42) 中，本发明提供 (3) 所述的方法，其中应用程序包括在单个二进制图像中，该单个二进制图像包括作为招募程序的一部分被分配到所有设备上的所有代码。

[0275] 在另一个实施例 (43) 中，本发明提供 (3) 所述的方法，其中所述方法进一步包括同步、串行化，以及协调设备组的行为，并且所述同步、串行化，以及协调是完全地或部分地通过单独地或者可选地随着与 DartEvents 或事件相关的文件一起传递或交换事件或 DartEvents 而实现的。

[0276] 在另一个实施例(44)中，本发明提供(43)所述的方法，其中所述事件参考至少一个文件，从而借助于该参考它们有效地包括任何复杂度的具有文件图像的一个文件或多个文件，并且随着事件结构内容传输这些文件图像。

[0277] 在另一个实施例(45)中，本发明提供(3)所述的方法，进一步包括在进行互操作和通信的初始设备和被招募的设备之间单独地或可选地随着与DartEvents或事件相关的文件传递或交换DartEvents事件，并且借助于DartEvent结构中的文件标识符(字段)参考，事件有效地包括任何复杂度的文件或其它数据或数据结构，当参考该文件图像时，总是随着事件结构内容一同传输。

[0278] 在另一个实施例(46)中，本发明提供(43)所述的方法，其中通过DartFramework，DartRuntime，和/或DartEngine，或者可选地通过非DartPlatform的专用的事件驱动的执行方案，在任何数量的编组的设备的事件队列上自动地备份和/或同步DartEvents或事件，从而被识别为自动同步或被Dart放置在事件队列中的DartEvents或事件以自动串行化和同步的方式出现在任何数量的编组设备的事件队列中。

[0279] 在另一个实施例(47)中，本发明提供(46)所述的方法，其中通过用于事件驱动的协作的应用程序的自动串行化和同步的程序、在包括初始设备的多个协作的设备上运行的功能或再现程序，来实现自动的串行化和同步，所述方法包括：借助于用于需要的每个设备间协作功能的应用程序，在初始设备上用例子说明连接管理器对象示例；程序性地借助于应用程序将在所有协作设备上同步的事件类型列表传输到连接管理器；建立协作的设备组，该设备组在每个设备上具有一个连接管理器并共享相同的同步事件类型列表；借助于连接管理器维持识别编组的设备和将被同步的事件类型的会话列表；借助于连接管理器检查将被放在事件队列中的所有事件，如果被检查的特定事件是连接管理器从应该被同步的会话列表中得知的事件类型，则将该事件标记为将与其它会话同步的事件，并且将该事件放置在连接管理器中列出的设备的事件队列中。

[0280] 在另一个实施例(48)中，本发明提供(47)所述的方法，其中通过直到其接收到确认信息才允许将一个事件放置在任何一个设备的事件队列中，来串行化将被任何一个事件驱动的协作应用程序功能或再现程序放置在协作的设备事件队列中的所有事件，所述的确认信息是任何一个设备将事件直接发送到的协作设备事件队列已经被成功地放置在协作的设备事件队列中。

[0281] 在另一个实施例(49)中，本发明提供(48)所述的方法，其中通过直到其接收到确认信息才允许将一个事件放置在接收的任何一个设备的事件队列中，来串行化将被任何一个事件驱动的协作应用程序功能或再现程序放置在协作的设备事件队列中的所有事件，所述的确认信息是任何一个接收设备将事件发送到的协作设备事件队列已经被成功地放置在协作的设备事件队列中。

[0282] 在另一个实施例(50)中，本发明提供(48)所述的方法，其中无论设备是通过直接的通信设备链直接地或间接地进行通信，任何数量的协作的设备事件队列都在所有协作的设备上建立一个排队事件的串行化系统。

[0283] 在另一个实施例(51)中，本发明提供(49)所述的方法，其中无论设备是通过直接的通信设备链直接地或间接地进行通信，任何数量的协作的设备事件队列都在所有协作的设备上建立一个排队事件的串行化系统。

[0284] 在另一个实施例 (52) 中，本发明提供 (47) 所述的方法，其中将被放置在来自与两个或多个协作的设备的协作设备队列中的事件被一个系统同步为协作设备队列中的一个串行化事件，其中只允许一个主设备将事件类型放置在将被同步的事件类型列表中，从而将在所有协作的设备上串行化所有这种事件。

[0285] 在另一个实施例 (53) 中，本发明提供 (52) 所述的方法，其中借助于主请求事件类型的事件通知主设备将代表其它协作的设备发布的事件，其中主请求事件类型的事件包括主设备将意图的事件放置在所有协作的设备队列中所需要的所有信息。

[0286] 在另一个实施例 (54) 中，本发明提供 (52) 所述的方法，其中被另一个设备招募进入协作设备组的每个设备将其相对的主设备看作招募该设备的设备。

[0287] 在另一个实施例 (55) 中，本发明提供 (53) 所述的方法，其中被另一个设备招募进入协作设备组的每个设备将其相对的主设备看作招募该设备的设备。

[0288] 在另一个实施例 (56) 中，本发明提供 (54) 所述的方法，其中将主请求事件类型事件放置在相对的主设备队列中，将使得事件从设备传播到相对的主设备上，直到不具有相对的主设备的初始主设备使用主设备将目的事件放置在所有协作的设备队列中所需要的信息来形成事件为止。

[0289] 在另一个实施例 (57) 中，本发明提供 (52) 所述的方法，其中通过向同步的和 / 或串行化的协作的设备队列发布变化的主类型事件而改变指定的主设备，自身处于串行化的事件列表上的主类型事件以同步的串行化的方式通知所有设备新的主设备将代替当前的主设备。

[0290] 在另一个实施例 (58) 中，本发明提供 (54) 所述的方法，其中主请求事件通过协作的设备队列进行传播，直到新的主设备处理事件为止。

[0291] 在另一个实施例 (59) 中，本发明提供 (55) 所述的方法，其中主请求事件通过协作的设备队列进行传播，直到新的主设备处理事件为止。

[0292] 在另一个实施例 (60) 中，本发明提供 (53) 所述的方法，其中被标识为指定的事件的一部分的可选的文件被主请求事件类型事件发送，如果存在这种文件参考，则该可选的文件被保持在每个传播的设备上并具有标识符，该标识符允许文件重新与将被主设备发送的事件关联，如同其已经被主设备作为事件的一部分发送，为了降低信息量，将必须作为每个事件的一部分进行发送的文件作为主设备处理的主请求事件类型的结果进行发送。

[0293] 在另一个实施例 (61) 中，本发明提供一种初始设备，包括 ：与存储器相连并适于执行程序的处理器，其中的程序包括用于实现目的任务的指令 ；用于至少部分地在处理器和存储器中执行以招募至少一个不同于所述设备并处于所述设备外部的一个被招募的设备以参与目的任务的实现的装置，所述被招募的设备至少包括目的任务的性能版本所需要的硬件资源 ；以及完全存储在所述设备中以补充所述被招募的设备的资源的装置，从而硬件资源和授权补充的资源使被招募的设备完全能够实现目的任务。

[0294] 在另一个实施例 (62) 中，本发明提供 (61) 所述的初始设备，其中所述设备和被招募的设备中每个都在共同的程序性环境中操作，至少部分地在处理器和存储器中执行以进行招募的装置包括用于在共同的程序性环境中通过至少一条连接将执行的程序广播到也包括或操作在共同的程序性环境中的其它设备的装置。

[0295] 在另一个实施例 (63) 中，本发明提供 (62) 所述的初始设备，其中用于进行招募的

装置进一步包括用于在其它设备上启动程序的执行以程序性地检查每个其它设备的资源和能力，从而确定每个设备是否具有参与实现特定的任务所需要的资源和能力的装置。

[0296] 在另一个实施例 (64) 中，本发明提供 (63) 所述的初始设备，其中在每个特定的其它设备上，至少部分地通过存取存储在特定设备上或关于特定设备计算的设备专用的硬件抽象层信息，来执行所述检查。

[0297] 在另一个实施例 (65) 中，本发明提供 (64) 所述的初始设备，其中所述用于进行补充的装置进一步包括用于发送和安装授权程序、数据、和 / 或内容的装置，其中的程序、数据、和 / 或内容是使每个设备实现其特定任务的相应部分所需要的。

[0298] 在另一个实施例 (66) 中，本发明提供 (61) 所述的初始设备，进一步包括用于在初始设备和其它设备上临时地或永久地同步操作的装置，所述用于同步的装置包括任务事件队列和用于维护该任务事件队列的装置。

[0299] 在另一个实施例 (67) 中，本发明提供被招募的设备，它包括 ：包括处理器和连接到该处理器的存储器的一组硬件资源 ；适合于实现一组任务的计算机程序代码资源 ；硬件资源能够或至少执行特定任务的执行版本，但是计算机程序代码资源最初并不能执行理想的版本或方法，以实现特定的任务或特定任务的一个方面 ；用于接收通信的装置，其中的通信至少包括计算机程序代码通信和数据通信之一，所述计算机程序代码通信包括使设备能够执行特定任务的理想的版本、方法或方面的补充的计算机程序代码资源。

[0300] 在另一个实施例 (68) 中，本发明提供 (67) 所述的被招募的设备，其中被招募的设备和初始设备中每一个都操作在共同的程序性环境下。

[0301] 在另一个实施例 (69) 中，本发明提供 (68) 所述的被招募的设备，进一步包括用于执行从初始设备接收的程序以程序性地检查被招募的设备的资源和能力，从而确定被招募的设备是否具有参与实现特定任务所需要的资源和能力的装置。

[0302] 在另一个实施例 (70) 中，本发明提供 (69) 所述的被招募的设备，其中在被招募的设备上，至少部分地通过存取存储在被招募的设备上或关于被招募的设备计算的设备专用的硬件抽象层信息，来执行所述检查。

[0303] 在另一个实施例 (71) 中，本发明提供 (70) 所述的被招募的设备，进一步包括用于安装授权程序、数据、和 / 或内容的装置，其中的程序、数据、和 / 或内容是使每个被招募的设备实现其特定任务的相应部分所需要的。

[0304] 在另一个实施例 (72) 中，本发明提供 (61) 所述的被招募的设备，进一步包括用于在初始设备和被招募的设备上临时同步操作的装置，所述用于同步的装置包括任务事件队列和用于维持该任务事件队列的装置。

[0305] 在另一个实施例 (73) 中，本发明提供在多个不同的设备上形成集成的特定的动态分布式信息处理系统的以参与特定任务的执行的方法，所述方法包括 ：初始设备启动分布式信息处理系统的形成，所述形成包括使用至少一个通信信道和协议广播消息，以识别和招募可能拥有参与执行特定任务的资源和能力的其它被招募的设备 ；按照需要将程序、数据，以及内容中的至少之一传输到每个被招募的设备，从而使每个被招募的设备都能够实现其相应的特定任务的部分。

[0306] 在另一个实施例 (74) 中，本发明提供 (73) 所述的方法，进一步包括 ：至少部分地在初始设备的处理器和存储器中执行用于招募至少一个不同于初始设备并处于初始设备

外部的被招募的设备以参与目的任务的执行的程序，所述被招募的设备至少包括目的任务的执行版本所需要的硬件资源；在初始设备中全部存储用于补充被招募的设备资源的程序和可选的数据，从而硬件资源和授权的补充的资源使被招募的设备能够完全实现目的任务。

[0307] 在另一个实施例 (75) 中，本发明提供 (74) 所述的方法，其中所述设备和被招募的设备中每个都在共同的程序性环境中操作，至少部分地在处理器和存储器中执行以进行招募的程序包括在共同的程序性环境中通过至少一条连接将执行的程序广播到也包括或操作在共同的程序性环境中的其它设备。

[0308] 在另一个实施例 (76) 中，本发明提供 (75) 所述的方法，其中所述招募进一步包括在其它设备上启动程序的执行以程序性地检查每个其它设备的资源和能力，从而确定每个设备是否具有参与实现特定的任务所需要的资源和能力。

[0309] 在另一个实施例 (77) 中，本发明提供 (76) 所述的方法，其中在每个特定的其它被招募的设备上，至少部分地通过存取存储在特定设备上或关于特定设备计算的设备专用的硬件抽象层信息，来执行所述检查。

[0310] 在另一个实施例 (78) 中，本发明提供 (77) 所述的方法，其中所述补充进一步包括发送和安装授权程序、数据、和 / 或内容，其中的程序、数据、和 / 或内容是使每个设备实现其特定任务的相应部分所需要的。

[0311] 在另一个实施例 (79) 中，本发明提供 (77) 所述的方法，进一步包括用于在初始设备和其它被招募的设备上临时地同步操作，所述同步包括生成和维护该任务事件队列。

[0312] 在另一个实施例 (80) 中，本发明提供 (73) 所述的方法，其中所述通信是这种通信和交互作用，即它既不属于客户机 - 服务器通信交互作用也不属于对等通信交互作用。

[0313] 在另一个实施例 (81) 中，本发明提供 (73) 所述的方法，其中招募执行特定的设备、服务，和资源发现以识别所需要的设备，接着使用事件向设备发送授权程序和信息；智能地和有效地形成设备组，并接着使用事件协调设备组，以实现最初运行在源初始设备上的 Dart 或应用程序或任务的目的。

[0314] 在另一个实施例 (82) 中，本发明提供 (73) 所述的方法，其中所述分布式信息处理系统包括存取和协调设备的一些或所有物理能力的使用。

[0315] 在另一个实施例 (83) 中，本发明提供 (82) 所述的方法，其中所述设备的物理能力是从下述组中选择的，该组可选地包括打印、传真、显示、播放音乐、播放视频、控制其它设备、在任何介质上存储数据 (无论是数字的还是模拟的)、制造产品、提供删除、拍摄照片的能力，或者可以被设备的处理能力存取、监视或控制的任何其它的物理能力。

[0316] 在另一个实施例 (84) 中，本发明提供 (1) 所述的方法，其中运行在多个设备上的软件应用程序至少部分地在两个或多个设备上执行互操作性操作，其中的设备具有最初为初始设备上的单个软件包的一部分的代码和 / 或数据和 / 或内容，从而享有比使用独立地开发和 / 或独立地分配的应用程序执行互操作性操作的情况下更可靠的互操作性。

[0317] 在另一个实施例 (85) 中，本发明提供一种与计算机系统或信息设备共同使用的计算机程序产品，所述计算机程序产品包括计算机可读的存储介质和嵌入在其中的计算机程序机制，所述计算机程序机制包括：引导计算机系统或信息设备以特定的方式起作用以招募被招募的设备组进行互操作的程序模块，所述程序模块包括用于下述的指令：通过至

少一个通信链接向不同于初始源设备的至少一个可到达的设备发送可操作的检查程序，以找到具有需要的资源或能力的设备，所述检查程序包括以与初始源设备和检查程序将达到的设备相同的可执行形式编码的检查程序指令；通过通信链接直接或间接地在初始设备上接收来自于每个可到达设备的回答响应；借助于在初始设备上执行的程序，分析从所有进行响应的可到达的设备接收的回答，以确定识别初始设备和进行响应的可到达的设备的能力和资源的组合的利用计划，以最佳地实现软件应用程序的目的；借助于在初始设备上执行的应用程序，将可执行代码、数据、内容，和/或 Dart 中的至少之一分配到每个可到达设备中的至少一个，根据标识的利用计划其中的可到达的设备被识别为具有所需要的资源或能力。

[0318] 在另一个实施例 (86) 中，本发明提供一种与计算机系统或信息设备共同使用的计算机程序产品，所述计算机程序产品包括计算机可读的存储介质和嵌入在其中的计算机程序机制，所述计算机程序机制包括：引导计算机系统或信息设备以特定的方式起作用以在多个不同的设备中形成集成的特定的飞行式分布式信息处理系统，从而作为单个的虚拟设备并参与到特定任务的实现中，所述程序模块包括用于下述的指令：通过初始设备启动分布式信息处理系统的形成，所述形成包括使用至少一个通信信道和协议广播消息，以识别和招募可能拥有参与特定任务的实现所需要的资源和能力的其它被招募的设备；按照需要将程序、数据和内容中的至少之一传输到每个被招募的设备上，从而每个被招募的设备都能够实现其相应的特定任务的部分。

[0319] 可以以许多不同的方式组合在这些示例性实施例中引用的特征和/或要素以及在详细的说明书或在附图中描述的示例性实施例的特征和/或要素，从而上述的实施例并不是对本发明的限制，并且具有特征和要素的任何不同组合或排列的其它的或可选的实施例也是本发明的实施例。

[0320] III. 再现适配和互操作性分段模型

[0321] 在本发明的另一个方面中，提供了用于再现的系统、装置、方法以及计算机程序。再现包括用于将互操作性应用程序分为一组离散的执行单元的方法，对于给定的环境和时间点选择该执行单元之一在给定的设备上运行。

[0322] 在一个特定的有益的实施例和实现方案中，Dart 应用程序框架 (DartFramework) 包括再现对象，其中再现对象作为被称为 Gizmos（参照图 11 所示）的处理单元分层结构的上部。为了试图理解什么是再现，可以将单个再现程序考虑为完整的传统的程序 - 尽管再现程序在下述方面与传统的程序并不相同：它可能是与相关的功能和内容一同生成并打包的多个再现程序之一，其中的功能和内容是实现互操作性操作所需要的。通常组中的再现程序都不能高效地和有效地独立实现应用程序的目的或再现程序包。互操作性应用程序，例如 Dart，是程序各部分的完整集合，可能包括知道如何将各部分组织在一起以动态地和静态地形成和管理组合的程序或再现程序的程序、数据、内容和/或资源（参照图 3 300、图 13、图 14 所示）。而且，互操作性应用程序或 Dart 包括用于智能地分配可能不同的再现程序或再现程序组的方法，一个再现程序组用于设备组中的每个设备，该再现程序组能够与其它设备组的其它再现程序作为一个组进行通信和协作，以高效地和有效地实现应用程序的目的。

[0323] 其中再现的优点包括在执行单元之间共享各部分，以限制互操作性应用程序的大

小。使用传统的编程技术，通常需要对一组应用程序进行打包，从而存在适于每个目标设备或环境的再现。如果没有再现，一般就会需要在包中的每个程序中存储多余的程序、数据、内容和资源。再现的另一个优点是可能存在与每个再现程序相关的选择程序（参照图 14 4000 所示），该选择程序被用在招募模型中以智能地选择在目标设备和环境上运行的正确的再现程序。再现的又一个优点是它能够将适配的数量限制到足够小的组，从而可能对互操作性应用程序进行完全地测试，同时由于存在至少一个适于在任何一个设备或环境上很好地运行的再现程序，而允许足够大的组。

[0324] 再现程序也可以被用于允许 Dart 工具（参考图 12200 中的 DartTools）自动地将各部分配置为发送到其它设备的专用的 Darts。例如，它们可以准备用于打印图像或文件。可以借助于 DartTools 静态地创建这种再现程序或再现程序集，或者通过在该文章其它地方描述的创建法的方法（图 720020）动态地创建，所述方法使用互操作性指令集，例如 DartInstructionSet，或者借助于两种方法的组合或者两种方法与其它技术相结合进行创建。

[0325] 再现程序的另一种用途是根据用户需求建立将被选择的一组语言或文化形式。例如，可以为法语、汉语等建立具有本质相同的信息的不同再现程序；并且不同的再现程序可以提供一些不同的信息，例如对于主要为日本观众的服装广告使用日本模特的照片，对于将被展示给意大利或欧洲观众的相同广告使用意大利模特的照片。

[0326] 再现程序的又一个用途是限制使用可能引起反感的、或不重要的，或另外地不适于特定人群的（或者任何其它的原因的）内容。再现程序的又一种用途是将内容的使用面向特定人群或设备，例如儿童或德克萨斯人。再现程序的又一种用途可能是面向不同的地点，例如包括与当地天气相关的信息的 Dart Renditions。

[0327] 在一方面，再现提供了一种用于紧密地将一组软件应用程序和相关数据、程序、内容以及选择标准程序的各部分打包为单个二进制图像的方法。再现还提供了各部分的数据库、特点、标识符和在二进制图像中的位置。再现还提供了将各部分映射为由程序、和 / 或数据和 / 或内容组成的多个独立的可执行的软件应用程序。它另外还提供了一种程序或一组程序，当该程序或该组程序被执行时，就确定对于给定的设备、用户偏好、通信特征，和 / 或环境执行一组软件应用程序和二进制图像中的相关数据、程序、内容中的哪一个。它还提供了一种工具集，允许根据一组源材料自动生成各部分并将各部分打包为单个的二进制图像（或者如果需要的话，打包为多个二进制图像或其它格式的图像）。再现的各个方面还提供了一种用于发现和存取给定二进制图像中各部分的数据库的机制。在图 8 和图 9 中表示了从 Dart 内部和借助于单个 Dart 使用分布式再现程序的例子。

[0328] 下面说明再现适配和互操作分段模型的一些特殊的示例性实施例。

[0329] 在一个实施例 (1) 中，本发明提供了一种用于将软件应用程序分为一组独立的可执行图像的方法，所述方法包括：根据将遇到的设备的实现应用程序目的的一个和多个方面的可能的资源和能力将遇到的设备分为各个等级；根据为实现应用程序目的的一个和多个方面的不同的再现需要或其它运行时间要求，将可能遇到的环境和操作要求分为各个等级；为需要的可执行图像指定需要的数据、代码、内容以及其它数字化表示的资源，以能够在每个等级的设备和每个环境和操作要求中实现应用程序目的的一个和多个方面；生成利用计划以选择哪一个设备和相应的独立分配的可执行图像在每个设备上运行，以在给定候

选设备、设备资源和能力，以及需要的环境或操作参数的列表的情况下实现应用程序的目的；以及指定所需要的数据、代码、内容和其它数字化表示的资源，以在每个等级的设备和每个环境或操作要求中执行和实现利用计划。

[0330] 在另一个实施例 (2) 中，本发明提供 (1) 所述的方法，其中所述软件应用程序将在一个或多个相同或不同的通信设备上运行。在另一个实施例 (3) 中，本发明提供 (1) 所述的方法，其中将选择一个正确的这种被选择的可执行图像，以在具有特定环境的给定设备上运行。

[0331] 在另一个实施例 (4) 中，本发明提供 (3) 所述的方法，其中关于每个设备的资源和能力以及环境和操作要求，根据应用程序的要求，选择一个正确的这种可执行图像并在每个设备上运行。

[0332] 在另一个实施例 (5) 中，本发明提供 (3) 所述的方法，其中至少一个设备是 Dart 设备。

[0333] 在另一个实施例 (6) 中，本发明提供 (1) 所述的方法，还包括执行生成的利用计划。

[0334] 在另一个实施例 (7) 中，本发明提供 (1) 所述的方法，其中所述软件应用程序被表示为 Dart。

[0335] 在另一个实施例 (8) 中，本发明提供 (1) 所述的方法，其中独立地执行的图像是再现程序。

[0336] 在另一个实施例 (9) 中，本发明提供 (7) 所述的方法，其中以与 DartFormat 一致的由 DartTool 打包的 Dart Renditions 的形式表示再现程序。

[0337] 在另一个实施例 (10) 中，本发明提供 (2) 所述的方法，其中当发送的程序在候选设备上运行时，完全地和部分地执行 (1) 中的利用计划，以确定特定的设备等级、资源和 / 或操作环境。

[0338] 在另一个实施例 (11) 中，本发明提供 (10) 所述的方法，其中所述程序是至少部分地由互操作性指令集中的指令组成的 DartProcedures。

[0339] 在另一个实施例 (12) 中，本发明提供 (11) 所述的方法，其中所述互操作性指令集是 DartInstructionSet。

[0340] 在另一个实施例 (13) 中，本发明提供 (11) 所述的方法，其中所述互操作性指令集可以在运行程序的一个和多个相同和不同的通信设备上执行。

[0341] 在另一个实施例 (14) 中，本发明提供 (13) 所述的方法，其中所述通信设备运行所说程序以确定特定的设备等级、资源和 / 或操作环境。

[0342] 在另一个实施例 (15) 中，本发明提供 (10) 所述的方法，其中所述程序被表示为作为应用程序的一部分的 Darts。

[0343] 在另一个实施例 (16) 中，本发明提供 (15) 所述的方法，其中所述应用程序被表示为 Dart，其中所述 Dart 包括用于在不同的通信设备上实现应用程序的目的的其它 Darts。

[0344] 在另一个实施例 (17) 中，本发明提供 (1) 所述的方法，其中 (1) 中的招募方法被用于发送和分配程序并分离可执行图像，以形成不同的或相同的设备组。

[0345] 在另一个实施例 (18) 中，本发明提供 (1) 所述的方法，其中各部分可以在不同的目标设备中的不同的独立执行的图像和处理环境之间共享，从而可以限制互操作应用程序

的大小。

[0346] 在另一个实施例 (19) 中，本发明提供 (1) 所述的方法，其中各部分可以在独立执行的图像之间共享，从而可以限制将被存储在软件中的数据量。

[0347] 在另一个实施例 (20) 中，本发明提供 (1) 所述的方法，其中各部分可以在独立执行的图像之间共享，从而可以限制将在设备之间传输的数据量。

[0348] 在另一个实施例 (21) 中，本发明提供 (18) 所述的方法，其中各部分是代码、数据、内容、程序、代码集、数据集、内容集、关于如何发现和组合各部分的元信息、描述性文本、图片、视频、图像、数据表，或能够以数字形式表示的任何其它信息单元或信息组的其中之一。

[0349] 在另一个实施例 (21) 中，本发明提供 (21) 所述的方法，其中各部分是 DartParts 和或元信息被表示为 Dart RenditionsTable 部分，和或 DartRenditionTable 部分，和或 DartPartTable，和或 DartTrailer。

[0350] 在另一个实施例 (22) 中，本发明提供了一种用于将软件应用程序分为一组独立的可执行图像的方法，所述方法包括：根据将遇到的设备的资源和能力将遇到的设备分为各个等级；将可能遇到的环境或操作要求分为各个等级；为可执行图像指定需要的数据、代码、内容以及其它数字化表示的资源，以能够在每个等级的设备和每个环境和操作要求中实现应用程序目的的一个和多个方面；生成利用计划以选择哪个设备和在每个设备上运行的可执行图像，以实现应用程序的目的。

[0351] 在另一个实施例 (23) 中，本发明提供了一种用于将软件应用程序分为一组独立的可执行图像的装置，所述装置包括：用于根据将遇到的设备的实现应用程序目的的一个和多个方面的可能的资源和能力将遇到的设备分为各个等级的装置；用于根据为实现应用程序目的的一个和多个方面的不同的再现需要或其它运行时间要求，将可能遇到的环境和操作要求分为各个等级的装置；用于为需要的可执行图像指定需要的数据、代码、内容以及其它数字化表示的资源，以能够在每个等级的设备和每个环境和操作要求中实现应用程序目的的一个和多个方面的装置；用于生成利用计划以选择设备和在每个设备上运行的相应的独立分配的可执行图像，以在给定候选设备、设备资源和能力，以及需要的环境或操作参数列表的情况下实现应用程序的目的的装置；以及用于指定所需要的数据、代码、内容和其它数字化表示的资源，以在每个等级的设备和每个环境或操作要求中执行和实现利用计划的装置。

[0352] 在另一个实施例 (24) 中，本发明提供了一种与计算机系统或信息设备一同使用的计算机程序产品，所述计算机程序产品包括计算机可读的存储介质和嵌入在其中的计算机程序机制，所述计算机程序机制包括：引导计算机系统和信息设备以特定的方式工作，来将计算机程序软件代码应用程序分为一组独立的可执行的计算机程序软件代码图像的程序模块，所述程序模块包括用于下述的指令：根据将遇到的设备的资源和能力将其分为各个等级；将可能遇到的环境和操作要求分为各个等级；为可执行图像指定需要的数据、代码、内容以及其它数字化表示的资源，以能够在每个等级的设备和每个环境和操作要求中实现应用程序目的的一个和多个方面；生成利用计划以选择设备和在每个设备上运行的可执行图像，以实现应用程序的目的。

[0353] 在另一个实施例 (25) 中，本发明提供了一种与计算机系统或信息设备一同使用的计算机程序产品，所述计算机程序产品包括计算机可读的存储介质和嵌入在其中的计算

机程序机制，所述计算机程序机制包括：引导计算机系统和信息设备以特定的方式工作，来将计算机程序软件代码应用程序分为一组独立的可执行的计算机程序软件代码图像的程序模块，所述程序模块包括用于下述的指令：根据将遇到的设备的实现应用程序目的的一个和多个方面的可能的资源和能力将遇到的设备分为各个等级；根据为实现应用程序目的的一个和多个方面的不同的再现需要或其它运行时间要求，将可能遇到的环境和操作要求分为各个等级；为需要的可执行图像指定需要的数据、代码、内容以及其它数字化表示的资源，以能够在每个等级的设备和每个环境和操作要求中实现应用程序目的的一个和多个方面；生成利用计划以选择哪个设备和在每个设备上运行的相应的独立分配的可执行图像，以在给定候选设备、设备资源和能力，以及需要的环境或操作参数列表的情况下实现应用程序的目的；以及指定所需要的数据、代码、内容和其它数字化表示的资源，以在每个等级的设备和每个环境或操作要求中执行和实现利用计划。

[0354] 可以以许多不同的方式组合在这些示例性实施例中引用的特征和/或要素以及在详细的说明书或在附图中描述的示例性实施例的特征和/或要素，从而上述的实施例并不是对本发明的限制，并且具有特征和要素的任何不同组合或排列的其它的或可选的实施例也是本发明的实施例。

[0355] IV.DartSource/互操作性资源再次想到DartSource提供了用于为打包的Dart互操作性应用程序指定所有需要的程序再现程序和代码、内容以及数据的结构和方法。DartSource将共同地用于面向特定设备指定单个可执行程序的语言结构扩展为这种语言，即也能够为设备组的智能招募和所需要的再现指定需要的程序，从而存在将被发送以在每个被招募的设备上运行的合适的再现程序，以实现该设备相应部分的被指定的应用程序的目的。

[0356] 在一个实施例(1)中，本发明提供了一种用于规定数字化编码的数据、代码以及内容，和所需要的数据、代码以及内容的形式的元信息的软件应用程序包，以在一个和多个连接的和间断连接的设备上实现目的的方法；所述方法包括以互操作性软件编程语言表示下述中的一个或者多个或者任何组合：(a)面向框架和或库的对象；(b)用于表示用于实现应用程序逻辑的主代码和数据的源代码，不管被表示为一个可执行的图像还是表示为完整的可执行图像组；(c)可数字化地表示的资源；以及(d)将应用程序的逻辑与设备(多个设备)本地的基本硬件和软件相连所需要的系统调用或指令调用。

[0357] 在另一个实施例(2)中，本发明提供(1)所述的方法，其中系统调用或指令调用用于将应用程序的逻辑连接到一个或多个下面的设备的本地的基本硬件和软件：(a)软件引擎；(b)软件操作系统；(c)通信子系统；(d)图像子系统；(e)加密子系统；(f)安全性子系统；(g)存储，音频播放，和或输入/输出子系统；(h)本地代码表示的算法或程序；(i)媒体压缩和或解压缩子系统；(j)数据处理或数据库处理；(j)通过应用程序接口表现的设备专用的独特功能和能力；(k)用于检索关于设备的资源、内容以及能力的信息的设备专用的轮廓信息；(l)事件队列和相关的事件队列管理子系统，以在设备上驱动应用程序的同步和异步操作；(m)用户接口、文本、音频、视频和其它自动编码或再现操作，和或一般的计算操作，和或一般的数据库操作；(n)电源管理、中止/恢复，和或从间断连接中恢复的应用程序级的错误恢复，通过使用事件来驱动在一个或多个协作的设备中或其之间共享和协作操作软件、数据、内容和状态；(o)动态地存储、配置、优化和或发送各部分，各部分的包、程

序、可执行图像、或可执行图像包到物理存储器或其它连接的设备或者从物理存储器或其它连接的设备发送的子系统；以及 (p) 以上的任何组合。

[0358] 在另一个实施例 (3) 中，本发明提供 (1) 所述的方法，其中所述框架或库和 / 或源代码包括类定义和对象执行，以实现、包括、存取，和 / 或组织 (a)-(p) 中列出的可被存取的一个或多个本地基础软件或硬件。

[0359] 在另一个实施例 (4) 中，本发明提供 (1) 所述的方法，其中所述框架或库和 / 或源代码包括类定义和对象执行过程，以为符合特定的运行时间惯例提供基础，其中的运行时间惯例用于在一个或多个协作的设备上或其之间的事件驱动的执行。

[0360] 在另一个实施例 (5) 中，本发明提供 (1) 所述的方法，其中所述框架或库和 / 或源代码包括类定义和对象执行，以确保设备之间的可执行的同步的操作中的事件的同步处理，这是通过在一组协作的设备上分配并运行的软件应用程序的各部分之间串行化事件的处理而实现的。

[0361] V. DartFramework/ 互操作性框架

[0362] 再次想到 DartFramework 是程序员在建立互操作性应用程序中所使用的 DartSource 的一部分，其中的互操作性应用程序包括利用 DartPlatform 的许多有益特点，而不要求程序员必须理解并实现 DartPlatform 的许多希望的互操作性特点。关于该说明书的线性任务分配部分还描述了包括更具体的 DartFramework 的各方面的互操作性框架的其它方面。

[0363] 在一个实施例 (1) 中，本发明提供了这种方法，该方法用于指定面向对象的具有一组面向对象的类定义，和被用作事件驱动的软件应用程序包的源规范的一部分的执行代码的互操作性框架，所述方法包括：(i) 使用面向对象的语言指定至少包括下述数据和代码成员的基本事件处理类：(a) 取事件数据结构的示例的参考或备份作为参数的处理成员；以及 (b) 具有相同的基类的其它事件处理对象的参考或示例的有序的列表成员；以及 (ii) 以源代码执行类规范的成员和方法。

[0364] 在另一个实施例 (2) 中，本发明提供 (1) 所述的方法，其中设计和执行所述软件应用程序包，以在一个或多个连接的或间断连接的设备上实现预期的目的。

[0365] 在另一个实施例 (3) 中，本发明提供 (2) 所述的方法，其中所述软件应用程序包符合 DartFormat。

[0366] 在另一个实施例 (4) 中，本发明提供 (1) 所述的方法，其中所述基类是 Dart Gizmo 类。

[0367] 在另一个实施例 (5) 中，本发明提供 (1) 所述的方法，其中还存在从基类继承下来的再现类，基类作为基类继承示例的树根，其中处理成员的参数可选地为空或 NULL 参考值。

[0368] 在另一个实施例 (6) 中，本发明提供 (5) 所述的方法，其中所述再现类用于表示执行独立的可执行图像的开始进入点，其中的可执行图像被包含在由利用面向对象的互操作性框架的源代码生成的输出包（或多个包）中。

[0369] VI DartTools/ 互操作性工具

[0370] 再次想到 DartTools 将 DartSource 应用程序规范处理为 Dart 应用程序包。

[0371] 在一个实施例 (1) 中，本发明提供了一种用于生成数字化编码的信息和需要的元

信息的互操作性软件应用程序包，以实现一个或多个连接的或间断连接设备的目的的方法；所述方法包括：通过互操作编译程序作用（软件产品）处理源材料，以创建对象文件；通过互操作性连接程序过程处理对象文件和可选的库，以创建库或互操作性软件应用程序包。

[0372] 在另一个实施例(2)中，本发明提供(1)所述的方法，其中数字化编码的信息包括数字化编码的数据、代码和/或内容，元信息包括数据、代码、和/或内容形式的元信息。

[0373] 在另一个实施例(3)中，本发明提供(1)所述的方法，其中所述互操作性编译程序过程被实现为编译程序计算机程序软件产品，连接程序过程被实现为连接程序计算机程序软件产品。

[0374] 在另一个实施例(4)中，本发明提供(1)所述的方法，其中根据互操作性源方法组合源材料。

[0375] VII DartFormat/互操作性格式

[0376] 再次想到DartFormat包括用于将Dart格式的包放在一起的结构和规则，其中的Dart格式的包包括互操作性应用程序所需要的所有代码、数据，和内容，加载所述的互操作性应用程序，互操作性应用程序并在包括运行的DartPlayer的DartDevice上运行。

[0377] 在一个实施例(1)中，本发明提供了一种用于存储与数字化编码的数据、代码和内容的互操作性格式一致的软件应用程序包，和以需要的数据、代码和内容形式的元信息，以实现一个或多个连接的或间断连接的设备的目的的方法，所述方法包括：形成至少一个线性连续的二进制编码的部分图像；形成由二进制编码的资源或程序要素的组合组成的任何需要的线性连续的部分图像，其中的二进制编码的资源或程序要素用于识别、加载、选择、执行或者被作为应用程序包的一部分进行处理；形成元信息；以及以一种形式将各部分和元信息打包在一起，其中的形式为可以确定地定位图像并识别、加载和执行独立的可执行的图像。

[0378] 在另一个实施例(2)中，本发明提供(1)所述的方法，其中所述至少一个线性连续的二进制编码的部分图像包括下述的至少之一：主要代码、主要数据、用于每个独立的可执行图像的记录表，以及用于属于特定的独立的可执行图像的每个部分的记录表。

[0379] 在另一个实施例(3)中，本发明提供(1)所述的方法，其中所述至少一个线性连续的二进制编码的部分图像包括下述中的每一个：主要代码、主要数据、用于每个独立的可执行图像的记录表，以及用于属于特定的独立的可执行图像的每个部分的记录表。

[0380] 在另一个实施例(4)中，本发明提供(1)所述的方法，其中所述需要的线性连续的部分图像可选地包括从由下述组成的组中选择的任何成员或组合：(a)程序、Darts、DartProcedures，或程序；(b)图片、视频、图像、音频、声音或能够被播放的任何其它媒体；(c)数据结构示例、列表、和或参数；(d)数据列表或数据表；以及(e)上述的任何组合。

[0381] 在另一个实施例(5)中，本发明提供(1)所述的方法，其中形成元信息的步骤包括形成至少下述之一的元信息：(a)给定标识符找到其各部分的表；(b)用于找到各部分的表的第一信息；(c)加载和执行线性连续的部分图像所需要的第二信息；以及(d)用于找到独立的可执行图像列表的第三信息。

[0382] 在另一个实施例(6)中，本发明提供(1)所述的方法，其中还形成一个或多个下述线性连续的二进制编码的部分图像：(i)程序、Darts、DartProcedures；(ii)包括从内容

项目组中选择的内容的内容，其中的组由图片、音频、视频，或其它多媒体内容或动画组成；(iii) 数据库；(iv) 索引；(v) 用于列表事项或表事项的参数；(vi) 虚指针数据；(vii) 应用程序堆积数据；以及 (viii) 可表示为连续的二进制数据图像的其它事项。

[0383] 在另一个实施例 (7) 中，本发明提供 (1) 所述的方法，其中还形成一个或多个下述的元信息：(i) 签名信息；(ii) 被存取以识别软件应用程序包的名称、类型、内容和或用途的关键字或其它信息；(iii) 用于列表项或表项的参数；(iv) 虚指针参数；(v) 安全校验和，和或签名，和或证书和或无用数据；以及 (vi) 唯一的标识符。

[0384] 在另一个实施例 (8) 中，本发明提供 (1) 所述的方法，其中通过包括在互操作性编译程序、互操作性连接程序和或互操作性主播放程序中的工具，执行所述步骤。

[0385] VIII. DartRuntime/ 互操作性运行时间

[0386] 在另一方面，本发明提供了一种软件运行时间模型（例如 DartRuntime 模型），对于所有的软件操作（无论是与 Dart 相关或与设备控制相关）该模型主要是事件驱动的，并且在一些实施例中完全是由事件驱动的。互操作设备组（图 17 660）中的每个进行互操作的设备上运行的一个事件队列提供应用程序和低级设备控制软件共用的一组事件语义、类型、结构和对事件的操作，其中所述的事件队列驱动同步应用程序事件处理和异步应用程序、设备、通信和互操作性操作的排队。同时还提供了用于管理事件的排队、解队和处理的指令集或系统调用。还提供了一种可选的稳定的设备互操作性通信运行时间模型，其中维护设备之间的通信，改正错误，并且需要时重新建立协作，但是只有少量的或者根本没有对有效地运行在互操作设备组上的 Darts 的中断（参照图 19 所示）。这些特点能够可选地扩展到分别生成的 Darts 和 / 或其它设备上。

[0387] 在该说明书的其它地方，包括下面的例子和示例性实施例，描述了 DartRuntime 的其它方面。在一个实施例 (1) 中，本发明提供了一种用于互操作性运行时间系统的方法，以实现事件驱动的软件应用程序包的执行，其中的软件应用程序包是在一个或多个连接的和间断连接的设备上实现指定目的（意图）所需要的数字编码的数据、代码和内容，以及数据、代码和内容形式的图元信息组成的软件应用程序包，所述方法包括：(a) 从给定的独立执行的图像包中选择并加载分离的可执行的图像；(b) 通过正在执行的图像将设备招募到组中；(c) 在招募的设备的组中分配代码、再现程序、数据，和 / 或内容中至少之一；(d) 以顺序的方式处理同步和异步的事件，以实现所述目的；(e) 在招募的设备组的各设备中和各设备之间同步和串行化事件处理；以及 (f) 在存储器中存储运行的单独可执行的图像包，包括存储数据和状态。

[0388] 在另一个实施例 (2) 中，本发明提供 (1) 所述的方法，其中 (a) 从给定的独立执行的图像包中选择并加载分别可执行的图像，是从给定的独立可执行的图像包中选择和加载正好一个分离的可执行的图像。

[0389] 在另一个实施例 (3) 中，本发明提供 (1) 所述的方法，其中所述运行时间是 DartRuntime。

[0390] 在另一个实施例 (4) 中，本发明提供 (1) 所述的方法，其中所述选择和加载、招募、分配、处理，以及同步和重新串行化是根据招募程序实现的。

[0391] IX. 线性任务分配

[0392] 在本发明的另一个方面，提供了用于线性任务分配的系统、装置、方法和计算机程

序。线性任务分配包括在处理单元之间实现处理器控制和设备资源的简单的确定的流动的方法。处理单元能够容易地被重新配置为单个体系结构，该体系结构包括被编译为应用程序的处理单元，分别地编译的应用程序，以及甚至运行在其它设备上的应用程序。

[0393] 因为多数互操作性应用程序旨在被单独的用户使用，因此线性任务分配模型的优点特别适合于互操作性应用程序。这里必须将控制传递到每个处理单元的运行时间的代价是不重要的，因为通常需要将人类的响应时间限制为三十分之一秒或更长。

[0394] 尽管存在许多发明的线性任务分配模型和方法的优点，但是这里仅说明其中的一些优点，其它的优点将是能够从本说明书的其它描述中明显得知的。

[0395] 第一个优点是由于将执行操作的顺序限制为很好控制的模式，因此用于所有编组设备的所有处理单元的处理顺序的确定的或接近确定的路径产生更简单的编程模型和极大减少的程序错误。

[0396] 第二，所有的处理单元能够完成复杂的操作顺序，而不会被及时的中断或被中断过程的动作影响。

[0397] 第三，可以容易地支持受控的关机，受控的休眠和恢复以及有益的垂直分层机制，可以例如用于如电源管理的应用。

[0398] 第四，随着操作模式和与用户的交互作用的变化，配置和重新配置体系结构至少是一个允许和禁止处理单元组的简单方法。

[0399] 第五，双亲处理单元可以以这种方式继承、改变和变形环境，即子女处理单元不需要包含知道它们的双亲处理单元如何使用它们的输出（如位图）或如何控制它们的环境（如时间感）的代码。

[0400] 第六，可以将分别生成的可执行表格容易地同化为运行的可执行表格的处理体系结构。在 DartPlatform 中，这种特点允许 Darts 在使用中将其它的 Darts 同化为其运行时间。例如，可以容易地构造前面描述的幻灯片放映 (SlideShow)Dart 应用程序，该应用程序能够收集并包含分别生成的 Darts，如幻灯片，如同其收集如幻灯片的静止 JPEG 图片一样容易。JPEG 图片可以被静止图片显示 Gizmo(Gizmo 是用于时间处理单元的基本调用）包括，而 Dart 可以被 Dart 容器 Gizmo 包括。

[0401] 在一个实施例中，在 Dart 平台（图 3 的 DartPlatform）的从 Dart 框架（图 11 的 DartFramework）到 Dart 运行时间（图 9、图 16、图 17 的 DartRuntime），到 Dart 引擎（图 22600 的 DartEngine）和 Dart 指令集（图 20、图 25 的 DartInstructionSet）的指令的整个执行中，有利地包括线性任务分配。其它的实施例可以结合可选的执行过程利用全部 Dart 基础设施来使用线性任务分配。

[0402] 在一个实施例中，发明的线性任务分配提供了一种用于确定地安排软件应用程序的多个处理单元的执行和运行时间环境的方法。软件或固件程序事件驱动的运行时间模型提供或生成这种环境，即以基于处理单元之间的连接的预定的顺序执行所有的处理单元。提供面向软件对象的框架，DartFramework（图 11），其中存在包括下述对象成员（图 11 115）中至少之一并且可能以任何组合包括所有下述对象成员（图 11 115）的基本处理单元类（图 11 115 的 Gizmo）：

[0403] (a) 双亲处理单元的可能为空的引用；

[0404] (b) 子女处理单元的可能空排序的线性列表引用；

[0405] (c) 相应于事件类型类的可选的二进制标记，其中从双亲和子女处理单元的链条继承的任何子女处理单元作用于该事件类型类，直到没有其它子女处理单元；

[0406] (d) 用于增加、删除，或重新排序双亲和子女处理单元的程序性方法；

[0407] 以及

[0408] (e) 用于对事件的处理进行排序的程序性方法。

[0409] 在至少一个实施例中，用于对事件的处理进行排序的程序性方法包括下述步骤（参照图 15 所示）：

[0410] (1) 根据事件类型、其值和参考值、处理单元的任务、事件指定的当前运行时间环境、双亲处理单元和设备的状态，执行需要的事件的任何前置－子女处理（参照图 15 8004 所示）；

[0411] (2) 设置相应于被子女处理单元链条作用的事件类型类的可选的标记，以显示没有时间类型被作用；

[0412] (3) 为每个子女处理建立任何环境变化，以引用子处理单元的列表的顺序调用每个子女处理（参照图 15 8005 所示）；

[0413] (4) 当每个调用返回逻辑 OR 时，每个刚被调用的子女处理需要处理处理类型的二进制标记，以根据所有子女处理和其继承处理的需要收集组合的事件处理类型；

[0414] (5) 根据事件类型、其值和参考值、处理单元的任务、事件指定的当前运行时间环境、双新处理单元和设备的状态，执行需要的事件的任何后－子处理（参照图 15 8004 所示）；

[0415] (6) 将来设置现在被该处理单元控制的事件类型的标记；

[0416] (7) 如果参考值非空则将控制返回到双亲处理单元（参照图 15 8006 所示）；

[0417] (8) 如果双亲处理单元为空则将控制返回到主处理循环（参照图 158008 所示）。

[0418] 由于所收集的二进制标记能够表示哪些事件类型是每个子女处理单元和其所有的继承处理单元进行处理所不需要的，因此该二进制标记能够可选地用于删除调用图表的分枝。如果相应于与标记相关联的事件类型分类的子女处理单元的标记值不为 1，则子处理单元和其继承处理单元就不需要处理该事件。

[0419] 现在描述线性任务分配的其它示例性实施例。在一个实施例 (1) 中，本发明提供了一种用于排序和管理软件应用程序包的多个事件处理单元的事件驱动的执行程序和运行时间环境的线性任务分配的方法，所述方法包括：提供面向软件对象的框架，框架包括基本事件处理单元类，直接或间接地从基本事件处理单元类继承的多个事件处理单元类或没有这种继承的事件处理单元类；以确保总是存在一个用于使事件通过链接形成的处理单元的图表的线性的确定的排序的方式来创建、保持、添加、删除或重新排序形成事件处理单元的图表或拓扑的链接；以及根据运行的应用程序的需要动态地改变处理单元的图表或拓扑。

[0420] 在另一个实施例 (2) 中，本发明提供了 (1) 所述的线性任务分配的方法，其中逻辑地扩展所述图表，以静态地通过一个或多个应用程序包中的引用，包括分别生成的应用程序包的图表，其中的应用程序包本身就是一个或多个其它的应用程序包的一部分。

[0421] 在另一个实施例 (3) 中，本发明提供了 (1) 所述的方法，其中逻辑地扩展所述图表，以动态地包括在运行应用程序包期间分别生成的应用程序包的图表。

[0422] 在另一个实施例 (4) 中，本发明提供了 (2) 所述的方法，其中处于分别生成的应用程序包图表的最上端节点的事件处理单元通过使参数进入事件处理的方法，来确定逻辑上它是扩展的图表的一部分，还是实际上没有被任何其它的应用程序包扩展的最上端应用程序包的开始。

[0423] 在另一个实施例 (5) 中，本发明提供了 (3) 所述的方法，其中处于分别生成的应用程序包图表的最上端节点的事件处理单元通过使参数进入事件处理的方法，来确定逻辑上它是扩展的图表的一部分，还是实际上没有被任何其它的应用程序包扩展的最上端应用程序包的开始。

[0424] 在另一个实施例 (6) 中，本发明提供了 (4) 所述的方法，其中如果并且仅当如果参数不指定将要处理的事件时，最上端节点将明确地试图从被其自身和按图表的线性顺序的任何其它的事件处理单元处理的事件队列中检索事件。

[0425] 在另一个实施例 (7) 中，本发明提供了 (5) 所述的方法，其中如果并且仅当如果参数不指定将要处理的事件时，最上端节点将明确地试图从被其自身和按图表的线性顺序的任何其它的事件处理单元处理的事件队列中检索事件。

[0426] X. 垂直分层

[0427] 在本发明的另一个方面中，提供了用于垂直分层的系统、装置、方法和计算机程序。垂直分层包括实现这种特征的高效和有效的实施方案，其中的特征本质上包括结束工具、应用程序、框架、引擎、再现程序以及指令集执行过程和操作中的紧密协作。

[0428] 本发明的垂直分层是所属领域当前状态中的传统的水平分层的替换方案，如果编程工具、应用程序、播放器、运行时间、操作系统，和 / 或低级函数能够完全共享数据结构和通信语义，则能够有利地采用垂直分层。

[0429] 在一个执行的实施例中，垂直分层被可选地但是有利地包括在 DartPlatform 的整个执行中 (图 3)。在 DartPlatform 中有利地采用垂直分层的两个例子是电源管理和应用程序级错误恢复 (图 19)。

[0430] 例如，在传统的设备中通常采用基于在最低水平检测的输入或输出活动的试探法来执行电源管理功能 ；然而，只有应用程序自身真正地知道其是否需要进行控制来返回到其处理中，以在三十分之一秒内译码视频的下一个帧，或者将不需要进一步的处理直到用户进行某种操作为止。

[0431] 在本发明的一个有益实施例中，当 Dart 运行时间 (DartRuntime) 每次通过被称为 Gizmos 的事件处理单元的单级线性任务分配时，Dart 平台 (DartPlatform) 收集处理器响应要求 (图 11、图 15)。在 DartPlatform 每次通过 DartRuntime 的设备外部分后 (图 15)，无论何时使用基于 Gizmo 的 Dart 框架 (DartFramework) 体系结构的处理单元构建 Dart (图 11)，DartPlatform 都将最低的同步的 Gizmo 树响应时间要求 (图 15 8010) 与设备引擎 (图 22 600 的 DartEngine) 的事件队列 (图 22660 的 EventQueue) 中的异步的事件时间要求相结合，以确定设备在将控制返回到 DartPlayer 之前能够断电或降低电源水平或消耗的准确的可靠的时间量。电源管理的方法，例如处理器逻辑时钟的减少，降低逻辑或电源电压水平，或切断较大电路部分的电源都是所属领域已知的。例如考虑这里前面描述过的互操作性幻灯片放映 Dart 应用程序。该运行在初始设备上的应用程序能够通过招募形成例如 5 个设备构成的组，其中虽然任意地进行比例调整或另外地修改以用于有效的传输

并且在每个连接和每个设备上显示，但是组中的每个设备显示与初始应用程序相同的幻灯片。如果具有无线连接的设备离开连接范围，或者便携式设备自动地关机以节省电池寿命的话，那么会发生什么情况呢？在传统的执行程序中，如果该设备突然移动回到连接范围内或被重新开机，则只有幻灯片放映（SlideShow）应用程序知道如何恢复（参照图 19 的例子）。在传统的所有级水平分层的设备中需要大量的程序，以能够无缝地恢复，特别当非初始设备被关机并丢失了整个幻灯片数据和状态时更是如此。这是因为传统的软件生态系统没有建立从检测失去连接的低级程序到应用程序的所有级所需要的水平抽象语义。没有用于信息从应用程序流向低级的通信系统并返回的直接机制，就难以编写应用程序以自动地将数据和状态信息重新发送到连接恢复的设备。

[0432] 在 DartPlatform 中采用垂直分层，当通过 DartFramework、DartRuntime，以及 DartEngine 将失去的连接建立到应用程序时，重新发送应用程序数据和程序以在设备组上恢复应用程序。因此使用 DartFramework 构建的并在 DartPlayer 上运行的任何应用程序不需要为稳定的多设备应用程序和数据同步恢复进行任何特别的编程处理。

[0433] 在 Dart Platform 的一个实施例中，直接通过直接在运行在 DartInstructionSet 上的 Dart 应用程序和控制通信的本地代码之间传递 DartEvents，传统的低级操作（例如通信）直接与传统的高级操作（例如，应用程序）进行互操作。

[0434] DartTools（图 3200、图 12200）、DartInstructionSet、DartRuntime（图 9、图 15、图 16、图 17）以及线性任务分配（图 18）都被设计为提供非常缺少抽象层的环境，其中应用程序形成并处理 DartEvents，与 DartEngine 内的通信函数具有完全相同的结构和语义。因此，应用程序能够容易地和有效地收集、同步和传输其所有需要和状态，通过传递和处理事件，作为 DartPlatform 的一部分的所有其它的软件操作被每级的互操作系统的几乎所有组件所了解，而不管这些组件是否是应用程序的一部分，执行应用程序的引擎，还是在整个设备组上分配的应用程序的互操作部分。

[0435] 在一个实施例中，垂直分层提供用于将软件应用程序的操作与设备控制软件密切结合以在应用程序和设备控制软件之间促进高级的协作功能的系统、方法、和计数机程序，其中的设备控制软件具有低复杂度、低处理要求、很少的应用程序接口，以及很少的协议接口的特点。它还提供了一种软件运行时间模型，不管是应用程序还是设备控制关联的，对于所有软件操作该模型主要是事件驱动的；并且提供了应用程序和低级设备控制软件共用的一组事件语义、类型、结构和事件的操作。还提供了驱动同步的应用程序事件处理和异步的应用程序、设备、通信和互操作性操作的排序的事件队列。垂直分层方法还提供了用于管理事件的排队、解队以及处理的指令集或系统调用。还可以提供可选的稳定的设备互操作性通信运行时间模型，其中保持设备之间的通信，纠正错误，并且必要时重新建立协作，但是具有对有效地运行在互操作设备组上的应用程序的少量的中断。用于串行化和同步通过协作的设备组的事件的系统被有利地用于保持异步操作的互操作系统的所有组件被紧密地连接，并因此被可靠地简单地执行，有效地和简单的使用。

[0436] 在另一个实施例中，本发明提供了用于直接将软件应用程序的操作与一个设备上或一组通信设备上的设备控制软件操作相连接的方法、计算机程序软件、设备和系统，以促进软件操作之间的协作功能的高效率和灵活的自由度，而不管是在一个设备上还是在一个或多个协作的设备上进行的。这可以包括源代码和资源、软件工具、预先打包的或预先指定

的软件框架或库、事件驱动的运行时间，以及指令集或系统调用组，以在每个设备上管理事件队列。在至少一个实施例中，为此目的在每个设备上管理正好一个事件队列。应用程序可以是 Dart，设备控制软件可以是 DartEngine 或 DartPlayer。在一些实施例中，设备组是或包括使用这里描述的招募或通过用于招募设备组的不同的方法建立的组。在一个实施例中，软件框架是 DartFramework。在一个实施例中，事件驱动的运行时间是 DartRuntime。在一个实施例中，源代码和资源是 DartSource 源代码并且软件工具是 DartTools。在一个实施例中，指令集和系统调用是被 DartInstructionSet 和 / 或可以作为部分内嵌指令类型的指令被调用的函数提供的，例如 Dart BUILTIN_INSTRUCTION(图 20 670、671、672、673) 和 / 或 OEM_BUILTIN_INSTRUCTION(图 20 674、680、681、682)。

[0437] 在至少一个实施例中，通过使具有通常理解的语义的单个数据结构在设备 (图 17 800)、应用程序，以及通信代码组成的系统中用于传输信息和 / 或内容和 / 或状态，至少部分地实现所说的效率。

[0438] 在至少一个实施例中，例如 DartEvents(图 17 800) 的事件被用作单个数据结构。

[0439] 在至少一个实施例中，设备控制软件操作包括通信、设备配置管理、设备发现、管理分配、解除分配以及访问存储器、物理存储器、物理显示装置、物理输入 / 输出设备、或设备的任何其它的物理的或软件虚拟的物理方面。

[0440] 在一个实施例中，软件工具采用编写的软件源代码和资源以使用预先指定的软件框架，其中通过访问用于管理事件的排队、处理和解队指令集和系统调用集，软件框架确保与支持的事件驱动的运行时间的执行模型一致。

[0441] 在至少一个实施例中，在图 15、图 9、图 16、图 17 中表示和描述应用程序的事件处理。在至少一个实施例中，被放入到设备的队列中的事件被串行化和同步。根据图 9 所示的串行化和同步方法执行串行化和同步操作。

[0442] 现在描述包括垂直分层的一个方面的特定的示例性实施例。在一个实施例 (1) 中，本发明提供了用于协调程序性 (软件) 组件之间的操作和数据移动的事件驱动的垂直分层系统，其中的程序性组件执行应用程序级操作、设备硬件控制级操作、通信级操作，或任何其它级或一个或多个编组的设备中或之间的操作的子集，以在程序性 (软件) 组件之间建立有效的和 / 或稳定的协作功能，所述系统包括 :(a) 在一个或多个设备上和运行在一个或多个设备中的程序性 (软件) 组件上，通常在所有事件生成和处理单元之间得知和了解其字段和字段定义的静态事件数据结构 ;(b) 在每个编组的设备上的队列，其中的设备存储、移除、管理，以及控制对事件数据结构示例的访问 ;(c) 用于管理队列中事件的放置、修改，以及移除的装置，其中的队列可以从所有协作的程序性 (软件) 组件进行访问 ;(d) 用于指定和保持事件类型的共同列表的装置，其中在协作的设备的队列之间串行化和同步事件类型 ;以及 (e) 用于确保在所有设备上使程序性 (软件) 组件以正好相同的顺序处理共用列表上的所有的任何类型的事件的装置，而不考虑是什么程序性 (软件) 组件启动事件，或者启动事件的程序性 (软件) 组件运行在哪个编组的设备上。

[0443] 在另一个实施例 (2) 中，本发明提供 (1) 所述的系统，其中存储、移除、管理，以及控制对事件数据结构示例的访问的每个编组的设备上的队列在每个编组的设备上包括正好一个队列。

[0444] 在另一个实施例 (3) 中，本发明提供 (1) 所述的系统，其中用于管理队列中事件的

放置、修改，以及移除的装置包括被实现为计算机程序的程序，该程序包括在互操作的设备的处理器逻辑中执行的多个程序指令，其中的队列可以从所有协作的程序性（软件）组件进行访问。

[0445] 在另一个实施例 (4) 中，本发明提供 (1) 所述的系统，其中用于指定和保持事件类型的共同列表的装置包括被实现为计算机程序的程序，该程序包括在互操作的设备的处理器逻辑中执行的多个程序指令，其中在协作的设备的队列之间串行化和同步事件类型。

[0446] 在另一个实施例 (5) 中，本发明提供 (1) 所述的系统，其中用于确保在所有设备上使程序性（软件）组件以正好相同的顺序处理共用列表上的所有的任何类型的事件的装置包括被实现为计算机程序的程序，该程序包括在互操作的设备的处理器逻辑中执行的多个程序指令，而不考虑是什么程序性（软件）组件启动事件，或者启动事件的程序性（软件）组件运行在哪个编组的设备上。

[0447] 在另一个实施例 (6) 中，本发明提供 (1) 所述的系统，其中下述中的一个或多个或任何组合为真：(1) 静态事件数据结构是 DartEvent；(2) 通过使用设备招募将编组的设备组合起来进行协作；(3) 通过使用互操作性指令集或 DartInstructionSet 使用用于管理事件的放置、修改，以及移除的方法；(4) 系统至少部分地通过互操作性运行时间或 DartRuntime 实现其事件驱动功能；(5) 以与所描述的在招募时串行化和同步事件相同的顺序处理用于指定共同列表并确保所有的任何类型的事件都在共同列表上的方法；以及 (6) 互操作性工具或 DartTools 用于生成系统的软件应用程序操作代码。

[0448] XI. 应用程序事件驱动的电源管理

[0449] 再次想到采用至少部分地包括在 DartFramework 中的线性任务分配和或垂直分层构建的 Dart 应用程序，总是能够跟踪其准确的响应时间需求，从而有效的电源管理技术，如使处理器速度减慢，能够延长电池的寿命，限制消耗掉的能量，或者限制设备上产生的热量。在所属领域的当前状态中，大多数应用程序不能跟踪其响应时间需求，并且如果这些应用程序能够跟踪的话，也不能通过现有的协议层将这些需求从应用程序传输到设备的硬件，其中现有的协议层符合这种规范，即不包括用于传输响应时间需求的接口的规范。

[0450] 现在描述应用程序事件驱动的电源管理的特定实施例。在各例子中说明其它的实施例。在一个实施例 (1) 中，本发明提供了一种用于设备能量和电源管理以及能量和电源消耗降低的方法，包括：在至少一个设备中建立一种事件驱动的运行时间环境，其中对于所述设备将实现能量和电源管理以及消耗降低；生成并保持至少一个事件队列，该事件队列确定所有同步和异步处理的事件以及相关的用于处理队列中事件的希望最小的处理事件；以及根据队列中希望最小的同步和异步事件处理时间，选择最终的最短的估计的响应时间。

[0451] 在另一个实施例 (2) 中，本发明提供了 (1) 所述的方法，进一步包括：在需要处理任何异步事件之前，在第一异步事件队列的所有事件中搜索驱动所有的异步处理并收集需要的最短时间的事件；在需要处理任何同步事件之前，在第二同步事件队列的所有事件中搜索驱动所有的同步处理并收集需要的最短时间的事件；以及通过选择异步事件所需要的最短时间和同步事件所需要的最短时间中的较短的时间，来确定最终的所需要的最短的响应时间。

[0452] 在另一个实施例 (3) 中，本发明提供了 (2) 所述的方法，进一步包括：在进一步处

理任何事件之前，使用最终的最短的响应时间值来执行电源管理或电源消耗降低的任务。

[0453] 在另一个实施例 (4) 中，本发明提供了 (2) 所述的方法，其中所述第一异步事件队列和所述第二异步事件队列是相同的一个统一的事件队列。

[0454] 在另一个实施例 (5) 中，本发明提供了 (2) 所述的方法，其中所述第一异步事件队列和所述第二异步事件队列是不同的事件队列。

[0455] 在另一个实施例 (6) 中，本发明提供了 (3) 所述的方法，其中所述事件是 DartEvents。XII. 互操作性应用程序事件驱动的错误恢复

[0456] 由于低容量电池电源引起的干扰、距离限制，以及突然断电，设备之间的无线通信连接通常是不可靠的。在传统的水平分层协议中，在设备的软件执行中，任何层中的重大错误将会导致不可恢复的错误，这是因为应用程序不具有标准的接口以容易地重新建立连接和连接环境，并且因为传统的应用程序不具有许多在运行在不同设备上的应用程序之间跟踪和重新建立共享的状态的基础设施，因此对于应用程序来说这种错误将难于恢复。DartFramework 跟踪共享的状态，并使用再现容易地在设备和垂直分层之间重新建立失去的状态，这就使得通信错误易于被传播到 Dart，并且 Dart 易于将恢复信息直接传播到通信处理单元。因此当恢复连接时，甚至当先前失去的设备已经丢失其所有应用程序状态时，运行在多个设备上的 Darts 能够无缝地从协作的设备之间的间断的完全失去的通信中恢复，并且恢复设备的共享状态。

[0457] 现在描述互操作性应用程序事件驱动的错误恢复的其它实施例。在一个实施例 (1) 中，本发明提供一种用于在失去通信或间断地进行通信的环境中适当地继续进行操作的方法，其中协作地运行在多个编组的设备上的事件驱动的互操作性应用程序包适当地继续进行其操作和 / 或部分地从目前失去与作为组的一部分的设备的通信中恢复，所述方法包括 :(1) 当失去或者中断从组成员设备到编组的设备的通信，并且在预定的或动态确定的时间段内不能重新建立该通信时，在组成员设备上生成通信会话丢失类型的事件示例，其中每个组成员和编组的设备实现其相应的应用程序包的目的 ;(2) 直接地或者通过驱动应用程序包的同步操作的事件队列，将通信会话丢失类型的事件发送到处理通信会话丢失事件的应用程序事件处理单元 ;(3) 在不需要被失去或中断通信的编组的设备的情况下就能继续进行其操作时，应用程序包的事件处理单元修改应用程序包的行为 ;(4) 当组成员和编组的设备的通信被恢复时，在组成员设备上生成通信会话恢复类型的事件示例 ;(5) 直接地或者通过驱动应用程序的同步操作的事件队列，将通信会话恢复类型的事件发送到处理恢复的通信会话的应用程序事件处理单元 ;(6) 接着，应用程序的事件处理单元使组成员设备发送需要的任何代码、数据，和 / 或内容，以使编组的设备与剩余的事件驱动的互操作性应用程序同步 ;以及 (7) 接着在实现应用程序的目的的过程中，应用程序的事件处理单元修改应用程序包的行为，以包括现在恢复的编组的设备。

[0458] 在另一个实施例 (2) 中，本发明提供了 (1) 所述的方法，其中下述中一个和多个在任何组合中为真 :(1) 事件驱动的互操作性应用程序包与互操作性格式一致或者是 DartFormat ;(2) 互操作性应用程序与该详细的说明书其它地方描述的一样，或者互操作性应用程序包为 Dart ;(3) 招募方法用于在多个编组的设备上建立协作运行 ;(4) 事件是或者包括 DartEvents ;(5) 事件处理单元是 Dart Gizmo 类示例，或者从 Dart Gizmo 类直接或间接继承的任何类的示例 ;(6) 由互操作性引擎或 DartEngine 在每个设备上实现事件的协

调和同步；以及 (7) 根据互操作性运行时间或 DartRuntime 在设备上或设备之间实现事件的协调和同步。

[0459] 在另一个实施例 (3) 中，本发明提供了 (1) 所述的方法，其中没有编组的设备也能继续进行其操作的应用程序的被修改的行为可选地从包括下述中一个和以任何组合的多个的一组修改中选择：(1) 失去通信的设备简单地显示允许应用程序在没有显示的情况下继续进行的状态；(2) 失去通信的设备执行的功能是冗余的，因此不是实现应用程序的目的所需要的；(3) 可以由设备组中的任何剩余的成员（或多个成员）执行失去通信的设备执行的功能；(4) 可以由设备组可以达到的另一个设备执行失去通信的设备执行的功能，该设备则可以被招募到设备组中；(5) 可以以降低的操作模式执行剩余编组的设备的功能；以及 (6) 上述的任何组合。

[0460] XIII. 互操作性指令集

[0461] 在另一方面，本发明提供了一种互操作性方法、软件，以及指令集。执行互操作性指令集 (IIS) 的软件或硬件是用于执行共同的程序性环境的有效方法，其优点为：(i) 用于编组和扩展或分配应用程序的招募模型；(ii) 允许未被修改的应用程序在其它不同的设备上运行；以及 (iii) 将独特的资源、独特的能力和 / 或独特的内容展示给其它应用程序和设备。

[0462] 在本发明的一个有益的实施例和执行过程中，互操作性指令集是包括在 Dart 引擎（图 22600 的 DartEngine) 中或与其兼容的 Dart 指令集 (DartInstructionSet)。

[0463] 应该知道，可以以硬件、固件，和 / 或软件实现计算机组件和信息系统和设备，通常存在关于对于特定执行过程的特定组件使用哪个硬件、固件，或软件的执行选择。因此对于 Dart 引擎和互操作性指令集也存在这种选择。因此应该知道，尽管可以根据一个硬件、固件或软件描述某个实施例的某个组件，但是可选的实施例可以使用硬件、固件或软件的不同组合来提供用于实现希望的功能或效果的装置。

[0464] 在一个示例性实施例中，尽管可以对组件和功能进行不同地分组，从而对于引擎或指令集的结构或操作来说，数量不是决定性的，但是用于实现互操作性指令集的硬件和 / 或软件包括 11 个组件（参照图 4 3010)。

[0465] 首先，处理器或中央处理单元 (CPU)、存储器，以及输入输出 (I/O) 的用于运行或执行程序、支持来自和到系统、设备、网络以及设备外面或外部的通信的能力。

[0466] 第二，存在存储器存取、计算、测试和分支、输入 / 输出指令，以至少执行传统的通用计算任务。

[0467] 第三，有利地提供互操作性性能增强的指令，用于将共同的二进制应用程序和再现程序的实际到达扩展到性能较低的设备上。

[0468] 第四，有利地提供互操作性指令，尽管如这里其它地方描述的对于所有的执行过程实施例并不需要所有的 Dart 组件，但是可以用于实现招募、再现、创造、垂直分层、线性任务分配、社会同步、社会安全、以及虚指针的 Dart 方法。

[0469] 第五，还有利地提供了某些独特的能力指令，它们可操作地将特定设备拥有的或可以从特定设备得到的任何特征、资源、能力，和 / 或功能展示给软件应用程序和其它设备。

[0470] 第六，有利地提供安全维护指令，以控制或另外地访问安全特征的设置，具有特定

的交叉设备访问权限的设备分组，和 / 或为应用程序设置对设备和 / 或资源的访问权限。

[0471] 第七，还有利地提供收集指令，例如 Dart 收集 (DartContainment) 指令，以允许 Darts 或 Dart 兼容的指令有效地将初始 Dart 的执行扩展到其它分别生成的 Darts 上，其中分别生成的 Darts 被作为初始 Dart 的操作的一部分收集、保持和运行。

[0472] 第八，为一个或多个译码，编码，压缩，解压缩，操作和再现图片、位图、声音、输入事件，文本再现，以及其它这种操作有利地提供共同的用户接口 (UI) 指令。

[0473] 第九，为例如，打开、关闭，以及保持会话和在会话中传输的数据有利地提供通信指令。

[0474] 第十，有利地提供存储指令，以控制和维持对存储器、存储设备、存储资源等的存取。

[0475] 第十一，有利地提供兼容性指令，以在不同格式或参数的数据、内容和 / 或代码之间进行转换或自动编码。

[0476] 互操作性指令集在其它共同的方法中例如使用虚拟机时的优点，是设计和优化互操作性指令集（特别是 Dart 互操作性指令集）以执行所有需要的互操作性操作，其中的互操作性指令集是作为被调度的这种函数的指令，即被编译和组合到设备处理器中的本地代码。在一个实施例中，包括了一些可选的特点和能力，互操作性指令集应该具有下述中的多数（即使不具有全部的话）：(i) 招募指令；(ii) 分析指令；(iii) 同步指令；(iv) 用户接口和图形指令；(v) 电源管理指令；(vi) 连接和会话管理指令；(vii) 存储指令；(Viii) 再现指令；(ix) 创造指令；(x) 应用程序部分管理指令；(xi) 加密大数数学 (large number math) 指令；(xii) 文本和 / 或符号解析指令；(xiii) 虚指针管理指令；以及 (xiv) 能够将设备的独特能力展示给 Darts 并从而展示给任何其它的 DartDevices 或 Dart 兼容的设备的指令。

[0477] 现在描述互操作指令集的其它特定实施例。在一个实施例 (1) 中，本发明提供了一种用于执行互操作指令集 (IIS) 的装置，所述装置包括：处理器；连接到所述处理器的存储器；以及同所述装置外部的实体支持与所述处理器通信的输入 / 输出 (I/O) 接口；用于实现包括下述中至少之一的方法的执行支持互操作装置：招募、再现、创造、垂直分层、线性任务分配、社会同步，以及社会安全；以及用于打开和维持通信会话和在通信会话期间在通信设备之间传输的程序、数据、内容或其它信息的通信互操作性指令。

[0478] 在另一个实施例 (2) 中，本发明提供了如 (2) 所述的装置，其中执行支持互操作装置包括用于实现所述方法的互操作性指令。

[0479] XIV 创造论

[0480] 在本发明的另一个方面，提供了用于创造论的系统、装置、方法和计算机程序。创造论包括使互操作性应用程序生成其它互操作性应用程序，而该被生成的互操作性应用程序又能够生成更多的互操作性应用程序的方法。这就允许以不同的形式在连接的和间断连接的设备上有效地动态地生成和分配应用程序、数据和内容。

[0481] 在一个有利的实施例和执行过程中，这是 Darts 的这种能力，即创建其它的 Darts（图 12700）或 Dart 程序 (DartProcedures 图 144000)，该被创建的 Darts 或 Dart 程序又能创建其它的 Darts 或 Dart 程序。创造论被包括在整个 Dart Platform(DartPlatform 图 3) 中。例如，Dart 工具 (DartTools 图 3200) 允许说明源代码中的各部分，并将源代码编

译为包括这些部分的DartMaster(图12230)。当DartMaster在MasterPlayer(图12203)上播放时,如果需要的话则能够收集更多的资源,并为用户接口提供要求任何需要的信息,关于该信息再现程序和各部分被包括在输出Dart或DartProcedure中。

[0482] 运行在任何Dart播放器上的任何Dart能够包括来自被DartEngine支持的DartInstructionSet的指令,以动态地形成各部分的再现程序并有效地将各部分打包成DartFormat文件。只要所创建的Darts和程序是由需要的数据、内容和程序形成的,则Dart方式创建定制的Darts的过程就能够无限地继续进行下去。

[0483] 创造论提供了一种使互操作性应用程序创建其它的互操作性应用程序,该被创建的互操作性应用程序又能够以递推的和或串行的和或扇出的方式,创建更多的互操作性应用程序的方法,相关程序和计算机程序代码和产品。提供了用于将源代码(图3 100)和可选的源数据和/或源资源编译为包括至少一个再现的二进制图像的工具。还利用了并可以提供可执行指令和/或应用程序编程接口(多个接口),以动态地将数字部分组合为包括一组再现(例如,Dart再现)的二进制图像,以及控制如何根据通信能力、设备特征和目标设备的环境组合再现的部分。

[0484] 工具可以包括Dart工具(图12 200)。二进制图像可以包括Dart格式(图3300、图13、图14),或由其组成,或者可以是不同格式的二进制图像或者是以非二进制形式编码的图像。数字部分可以是被表示为结构化的数字序列的任何类型或形式的程序、数据集,和/或内容。

[0485] 现在描述创造论的其它特定实施例。在一个实施例(1)中,本发明提供了一种用于使最初独立的可执行图像或独立的可执行图像包动态地生成至少一个其它目标的独立的可执行数据图像或独立的可执行数据图像包以实现最初可执行图像或独立的可执行图像包的目的的方法,所述方法包括:(1)收集关于设备的特征、内容、资源或能力中至少之一和或用于执行生成的可执行图像或图像包的其它环境的第一信息,该信息可以用于实现生成可执行图像或图像包的目的或部分目的;(2)确定如何组合下述中至少之一:(i)其自身图像的部分;(ii)被收集的信息;或(iii)编程产生的信息,以有效地利用目标设备的资源、能力和内容,或对其收集信息的环境;以及(3)按照需要,收集生成一个或多个其它独立生成的可执行数据图像或图像包所需要的第二信息,以不受限的顺序实现生成的目标可执行图像或图像包的目的。

[0486] XV. 互操作性引擎/DartEngine

[0487] 再次想到DartEngine是或者包括用于在设备上执行Darts指令并实现其目的的软件和或硬件。DartEngine和设备专用的DartPlayer(DartEngine被包括在其中),提供了共同的执行过程和DartRuntime环境,该环境允许招募和再现以建立高效的设备组,并尽可能地传播其代码、数据以及内容,以实现Darts的目的。

[0488] 现在描述互操作性引擎的其它特定实施例以及互操作性引擎DartEmgine的更加特定的实施例。在一个实施例(1)中,本发明提供一种使或辅助设备彼此进行互操作的互操作性引擎,所述引擎包括:(1)用于加载、运行,和实现包括代码的互操作性软件包的至少一部分目的的装置,其中至少一部分代码嵌入在与互操作指令集一致的指令序列中;(2)用于发现其它互操作性设备的装置;以及(3)用于直接地或间接地与其它互操作性设备进行双向通信的装置。

[0489] 在另一个实施例 (2) 中，本发明提供了如 (1) 所述的互操作性引擎，其中下述中的一个或多个在任何组合中为真：(1) 所述引擎包括 DartEngine；(2) 互操作性软件包包括互操作性软件包或者包括与 DartFormat 一致的 Dart；以及 (3) 用于发现其它互操作性设备的装置，所述装置是至少部分地使用招募程序或 Dart 招募程序实现的。

[0490] 在另一个实施例 (3) 中，本发明提供了如 (1) 所述的互操作性引擎，其中所述引擎包括事件队列，并执行指令以通过互操作性应用程序包支持使用事件队列。

[0491] XVI. 互操作性设备授权

[0492] 再次想到互操作性设备授权是通过将 DartEngine 移植为 DartPlayer 的一部分，将传统的设备变为具有高度互操作性的 DartDevice 的过程。此外，还需要存取设备专用信息、能力和内容所需要的硬件抽象层的执行过程。在具有 DartPlayer 的设备变为 DartDevice 之前，必须执行至少一个通信协议。

[0493] 现在描述互操作性授权的其它特定实施例。在一个实施例 (1) 中，本发明提供了一种用于使用互操作性引擎的共同软件源代码创建使设备成为互操作性设备的互操作性软件的方法，所述方法包括：(1) 创建互操作性引擎对象或示例；(2) 创建设备硬件抽象层 (HAL) 对象或示例；(3) 确定并填充 halBase 类或说明的所有预定义的要求的 halBase 成员功能的功能性；以及 (4) 创建充分连续地在执行线程上运行引擎的设备专用的播放器对象。

[0494] 在另一个实施例 (2) 中，本发明提供 (1) 所述的方法，其中创建充分连续地在执行线程上运行引擎的设备专用的播放器对象的步骤如下：(a) 调用引擎初始化函数；(b) 在循环中调用引擎处理函数，如果返回值表示出现了不可恢复的错误或将关闭引擎，则结束循环；(c) 调用引擎非 - 初始化函数；以及 (d) 停止执行的线程。

[0495] XVII. 互操作性安全模型 /DartSecurity

[0496] 再次想到 DartSecurity 是用于提供需要的基础设施以保护设备的完整性和设备内容不会被恶意的和意外的破坏的系统和方法。

[0497] 需要将稳定的安全基础设施用作保护设备、设备的内容或辅助设施不被滥用、破坏或另外地损害，或者保护其不以任何不希望的方式被使用。如图 29 所示，这种基础设施是 DartPlatform 的组件以优选的方式实现的发明的 DartSecurity 系统。

[0498] 在优选实施例中，当 DartDevice 第一次运行 DartPlayer 时，DartEngine 的部分初始化过程执行下述：

[0499] 1. 收集适用于生成公共和私人加密密钥对和唯一的 ID(UID) 的足够的信息熵，以统计学地保证生成的密钥对和唯一的 ID 是真正不同于那些以相似的方式生成的但是使用了不同的收集的信息熵的密钥对和唯一的 ID。

[0500] 通过数字化地采样设备外世界的各方面，能够可靠地收集信息熵，其中不管怎样设备都没有与运行采样硬件的时钟同步。用于与其它设备通话的通信机制通常是信息熵的很好的来源，这是因为电磁干扰和与控制采样的处理器的时钟无关的量子噪声影响了包的准确定时，数据的超时和变化。信息熵的另一个容易的来源是设备请求来自人类用户的输入并无论何时用户提供任何输入时都以细粒时钟进行采样。包括定时和数据采样的信息熵能够被散列为一组用于为随机数生成器提供种子的数据值。尽管实际上很难知道何时收集了足够的信息熵，但是可以进行估计，并采用过采样来确保收集了足够的信息熵并将其保

存为散列的数据值。

[0501] 2. 使用传统的随机数生成算法和传统的加密操作以生成通用的唯一的公共 / 私人密钥对和用于在任何时候唯一地识别设备的通用的唯一的DartDevice UID。在优选的实施例中，所有UID的长度都是128比特。

[0502] 3. 以所属领域状态已知的任何传统方式在设备上存储私人密钥和信息熵散列的数据值的组，以遮挡和或隐藏设备上的数据，但是实际上仍然由引擎使用数据。

[0503] 4. 根据一组二进制标记限制对设备的数据、内容或工具资源的访问。在优选实施例中，如果阻挡对用于设备工具的相应数据的访问，则标记为1。例如，对于下面的资源可以有单独的标记：

[0504] a 阻挡对标记为私人的数据或内容的访问；

[0505] b 阻挡对到另一个设备的公开的通信会话的访问；

[0506] c 阻挡对本地存储设备的访问；以及

[0507] d 阻挡对用户接口元件，例如屏幕、键盘、鼠标以及发声器的访问。

[0508] 5. 将被加载以在DartEngine上运行的每个Dart或DartProcedure具有明确的或暗示的标记值的组，该标记值控制其对DartDevice资源的访问。每个DartDevice具有明确的或暗示的访问标记的组，该标记对应安全级别。如果应用程序或发送应用程序的设备不具有用于资源的所有标记，则将不会加载Dart应用程序或DartProcedure，除非设备先从具有可接受的授权或标记的用户或其它设备得到适当的授权，其中的授权或标记授权Dart或DartProcedure在特定设备上运行。可以在目标DartDevice上永久地，或者持续特定的时间段，或者在Dart或DartProcedure执行期间或者在设备间的通信会话持续期间，保存与Darts、DartProcedures或DartDevices的UID＝捆绑在一起的访问标记。

[0509] 6. 如果Dart应用程序试图使DartEngine代表其访问任何资源，其中对于所述资源应用程序的环境标记被设置为1，则DartEngine将立即停止执行应用程序的指令并关闭应用程序。

[0510] 在一个优选执行过程中，必须与用于进行访问的标记组一起数字化地标记Dart。设备之间的通信会话的一部分或内容的加密 / 解密也是DartSecurity系统的一部分。

[0511] 图25表示具有一些安全系统组件的DartDevice400。DartEngine具有用于支持加密150、信息熵收集153、随机数生成152、以及使用DartSecure协议保护设备之间的所有通信的基于指令的方法，以确保能够自动地加密任何两个DartDevices之间的任何通信，从而DartEngine的基本通信不需要依赖于任何外界的协议来保护数据的通信。可以使用安全套接字层(SSL)协议或在设备之间建立安全的通信链接的任何其它的协议来实现DartSecure协议153。在优选的执行过程中，因为SSL中许多选项都是不需要的，因此执行SSL协议标准中的功能子集以减小大小和复杂度。所有需要的大数数学加密基元150和151都被表示为DartEngine的一部分。为了实现更好的性能，理想的是以引擎的而不是Dart内的本地代码实现加密数学基元。

[0512] 图25还表示DartEngine维护相应于安全等级的UID列表155和156。图25还表示存在将级别数字映射为访问权限标记组的表159。设备和作为特定权限级别的应用程序ID的列表被DartEngine维持，并且当允许从设备向设备这样传播时以优选的操作模式对其进行过渡的传播，从而列表中的任何设备能够访问新的设备而不需要进一步收集权

限。在该文章的其它地方进一步描述了本发明的设备的过渡编组和特定安全级别的即社会安全的应用程序。

[0513] 为了确保甚至在 Dart 应用程序或 DartProcedure 出示其标记、符合要求，并被加载后，也不会故意地或无意地访问范围之外的资源，引擎使用环境数据结构 160 和 170 检查所有这种试图的访问，其中数据结构带有最初被 Dart 或 DartProcedure 出示的所有允许标记。

[0514] 注意到如该文章其它地方进一步描述的，撤销列表 155 可以被过渡性地保存为社会同步，从而之前与特定安全等级的其它设备共同编组的任何设备能够使其访问权限被逻辑地撤销，从而与传递地从组中的任何其它设备得知撤销的任何设备进行互操作。在一个优选的执行过程中，如果被撤销的设备得知撤销后与编组的设备连接，则 DartEngine 将使得被撤销的设备不进行操作或者部分地不能进行操作。在具有共同的访问权限的可互操作的设备的组上撤销访问权限并不是理想的，但是在许多情况下，在可能间断连接的设备的组上撤销丢失的、行为失常的或有不良影响的设备的访问权限的系统是实际存在的。

[0515] 传播访问权限或访问权限的撤销的方法非常类似于告知其朋友对他人造成伤害的人的个人，并且类似于传播流言的方式将信息在人们之间传播，从而大多数的人都能够得知并能够远离对他人造成伤害的人，甚至当这些人没有直接地与最先知道该对他人造成伤害的人交谈时也是如此。

[0516] 除了访问标记外 DartEngine 环境还包括引擎通过运行的 Dart 或 DartProcedure 用于限制对不是运行的 Dart 的一部分的任何内存或存储器或代码进行访问的信息。在所属领域的状态中这种访问限制被称为“沙箱”。DartEngine 在所有 Darts 和 DartProcedures 上支持这种沙箱，从而 Darts 不能被用于访问设备外的功能、内存和存储器部分，其中明确地提供这些部分用于被沙箱中运行的 Darts 和 DartProcedures 使用。

[0517] Darts

[0518] 在本发明的一个优选执行过程中，在传统地被认为是代码、数据、操作系统、图形或用户输入子系统，以及线程之间，或就此而言内容、程序，或者甚至设备之间，几乎没有限定。DartPlatform(图 3) 能够智能地混和和匹配软件程序、代码、数据、内容，以及设备电路，使之成为集成的虚拟系统，该系统实现 Dart 设计者设计的以及 DartSource 中指定的应用程序的目的 (图 3 100)。

[0519] DartTools 创建的 Darts 能够自优化、重组、发送和同步其版本或扩展，以有效地在不同设备之间或不同设备中共享程序、控制、硬件资源和内容。

[0520] Darts 被创建为具有建立集成的工作应用程序所需要的所有数据、内容和程序，其中的应用程序能够在任何数量的不同设备上，和同时地在许多类似或不同的设备上很好地运行。

[0521] 此外，单个 Dart 的运行时间环境也能够动态地或静态地包括作为任何 Dart 的双亲 Dart 或子女 Dart 的其它 Darts。因此 DartRuntime 环境在其到达的未知设备和未知的独立产生的 Darts 上是唯一的。

[0522] 不同于目前采用的互操作性技术，尽管 Dart 技术不限于这种任务，但是关于人类确定大小的数据以及如幻灯片放映、约定日历、联系、控制面板和消息对该 Dart 技术进行优化。系统的响应时间快得足以在五分之一秒中放映运动视频，响应按压按钮或请求在个

人联系列表中找到特定的联系人。对于人类来说，这种响应时间几乎不能与即时响应相区分。在多数应用程序环境的基础上，Dart 的响应时间要求与大多数实时操作系统相比是非常低的，这是因为通常希望实时操作系统运行如数据服务器的应用程序，其中的数据服务器必须每秒处理几百个操作。因为 Darts 仅需要在人类的反应时间内进行响应，因此使用有益的简单的和高度稳定和灵活的分级事件处理机制，以代替在所属领域的当前状态的互操作性操作系统和其相关联的运行时间环境中采用的标准的占先的或协作的线程系统。

[0523] 考虑幻灯片放映 Dart 应用程序的例子。在一个执行过程中，尽管可以使用任何编程语言，但是使用 DartFramework（图 11 102）以 C++ 程序代码语言编写幻灯片放映 Dart，还以 C++ 编写并使用 DartTools（图 12200）编译和链接。C++ 源代码还包括 Dart 专用的 C++ 程序语句，程序语句将能够被创建的应用程序扩展为包括所有的 DartProcedures、再现程序，和其它 Dart 应用程序需要的互操作性扩展程序，以发现和检查其它设备、根据所述检查发送优化的备份或部分、使该优化的备份或部分在选择的发现的设备上运行，并接着通过所有包括的设备上的事件队列进行互操作。这种设备可以是事件驱动的，其中事件自动地串行化和 / 或同步，从而以高度协作和稳定的方式操作应用程序的所有组件，以将互操作性应用程序的目的表示为封装在 Dart 中（图 9）。在优选的执行过程中，直接从 DartTools（图 12 200）生成的 DartMaster（图 12 230）将包括单个 MasterRendition（图 11 113）导出的对象，它提供了作为执行的开始点的主要方法，该执行继续创建 MasterRendition 对象维护的列表，其中 MasterRendition 对象包含对其它再现程序导出的对象（图 11 114）的参考值。

[0524] 通常由 C++ 或其它高级语言程序员使用指向 DartInstructionSet 的工具和框架创建 DartMasters。为了涵盖 Dart 可能发现其所运行在的设备的全部范围，一般地 MasterDart 将包括 MasterPlayer 所使用的内容和程序，以在一个和五个不同再现程序之间生成。

[0525] 逻辑上，再现程序可以被看作分离的程序，或可执行的图形，其中在优选的执行过程中，在任何一个时刻在任何一个设备上仅选择一个程序进行运行。

[0526] 物理上，再现程序通常有利地共享其数据、代码和内容组件的大多数，从而在实际的 DartFormat 二进制图像或文件中有利地存在数量极大降低的备份。

[0527] Dart 的 DartSourece（图 3100）还应该包括在设备上运行的程序，以确定哪个再现程序最佳地运行在该设备上。

[0528] 例如，包括图片集的幻灯放映 Dart 可能具有下述三种再现程序：(i) 用于设备的简单的文本再现程序，仅具有单个线条或几个线条的文本显示，该再现程序卷动幻灯片放映的名称和包括的幻灯片名称的列表，因为该程序不能显示图像本身；(ii) 小屏幕的再现程序，例如可能适用于具有小屏幕尺寸和有限的 CPU 电源的蜂窝电话和 PDA；以及 (iii) 高端的大屏幕再现程序，以多个显示选项、指标和过渡效果显示大的图片，可以适用于运行在更大屏幕的个人计算机 (PC) 或家庭娱乐系统中。

[0529] 在一个特别有益的操作模式中，当包括多个再现程序的 Dart 首先开始在一个设备上运行时，Dart 程序使用 DartInstructionSet 的轮廓轮廓指令以在该设备上检查被构建为 DartEngine 的设备配置，从而确定该设备是否具有运行最先进的再现程序所需要的所有特征。如果具有的话，则将运行最先进的再现程序。如果不具有的话，则对每个次先进

的再现程序程序性地检查设备配置，直到发现一个能够在设备上有效地运行的再现程序为止。

[0530] 注意到不具有嵌入的DartEngine的目标设备能够被具有DartEngine的任何设备或通过具有DartEngine的任何设备访问，通过对目标设备的深入了解和通过到目标设备的连接代理和虚拟用于该设备的DartEngine的操作来构建该设备。一个例子是使用不具有DartEngine的打印机，但是该打印机的确能够被想要在其上进行打印的设备通过个人计算机达到，其中的个人计算机自身运行DartEngine，通过PROFILE_INSTRUCTION展示其打印资源。只要在PC上运行的DartEngine展示打印能力，则具有DartEngine的任何其它初始设备将能够使用个人计算机可用的打印机，并且通过个人计算机上的DartEngine的硬件抽象层(HAL)对象的配置和打印方法进行访问。

[0531] 当请求Dart将其自身发送到另一个设备上时，多数Darts中一个用户的选项将发送的再现程序集限制为那些能够运行在目标设备上运行的程序。这样，用户能够限制到目标设备的传输时间和对目标设备的内存要求。当然新的设备不会具有更高级的再现程序以重新发送到能力更高的设备上。

[0532] 由于大量的内容以及创建和编辑内容的应用程序是基于PC或互联网的，因此Dart内容能够输入和输出现有软件系统本地生成的内容就变得重要了。

[0533] Darts可以有利地包括用于JPEG图片的标准格式的数据，可能遇到的任何其它形式的数据或内容，等等，并且菜单选项可以被构造为能够输入和输出图片、视频、音频、文本和将不同组件联系在一起的XML文档的Dart内容。

[0534] DartInstructionSet可选地但是有利地包括JPEG和其它标准媒体格式的解压缩和回放指令，从而可以有效地执行CPU密集的解压缩任务，作为DartEngine的一部分将其编译为DartDevice的CPU的本地指令集。这些指令还限制必须包括在Darts中的代码数量，这是因为DartEngine进行了作为执行DartInstructionSet一部分的许多解压缩和显示的操作。类似地，存在XML和其它文本解析指令以限制应用程序代码数量，并当解析文本、RTF、XML和其它基于文本的文档时提供本地代码速度的优点。

[0535] 此外，PC应用程序能够容易地被其制造商修改以直接输入和输出Dart内容。Darts能够包括内容存取APIs和控制APIs，以允许其它应用程序和Darts程序性地计数和提取内容部分，例如JPEG图片，以及音频剪辑和文本名称和部分的描述。

[0536] 也能够计数和访问Dart控制APIs，和API函数的文本描述，以从其它Darts或设备控制Darts自身的操作，允许对Darts和其操作的设备进行远程控制。

[0537] 将Dart Engine移植到设备上的程序的实施例现在描述用于将DartEngine移植到新的设备上的程序。该例子假设Dart Engine是C++代码编写的Dart Engine，但是并不脱离程序的一般性。

[0538] 首先，通过从halBase对象继承或以任何其它方式，如直接创建的方式，来创建你自己的硬件抽象层(图4 3020、图22 650、图27 650)对象。

[0539] 第二，将包括下述的功能填入halBase成员功能的功能性中：分配给定大小的单个连续的内存块，在几毫秒内返回时间，移动三个标准的内部格式之一的位图，或者在给定位置将时间变量编译到屏幕。HalBase类还包括用于得到配置信息字的虚函数，即必须提供配置信息字以允许Darts在DartEngine上运行，确定实际设备的CPU、内存、屏幕、通信、声

音、打印，以及其它特征。特殊的优点是通过使用 halBase 对象的 OEM_Function 方法设计和创建程序性接口，以展示设备的任何独特的能力、内容或功能，这些能力、内容或功能不会另外地通过基本的 halBase 类的真正的虚拟方法展示。

[0540] 第三，通过从 playerBsae 对象继承，来创建设备专用的 DartEngine 对象（图 22600），或者不利用继承直接创建。

[0541] 第四，构建设备的可执行的采用 DartEngine 对象的 DartPlayer（图 22600），包括在循环中调用 DartEngine 的 Process() 成员函数（例如，playerBase::Process() 成员函数（图 23 4003、图 25 611）的直到返回满足预定条件（例如，非零值）的循环。DartPlayer 的所有同步（图 15 8010）和异步操作（图 16 在虚线框内表示的异步事件处理），包括通信，都是被该简单的循环的执行线程驱动的，因此不需要多线程的操作系统。

[0542] 设备互操作性的 Dart 方案有利地将适配和测试复杂度方程从 N- 平方（图 1 和图 2）变为 N 的一阶。代替必须考虑新的设备如何与每个其它类型的不同设备共享内容和控制并执行各自的方案，在具有 Darts 时仅需要将 DartEngine 移植到设备专用的 DartPlayer 上（图 10）。

[0543] 需要执行了解设备具有的所有通信信道的功能，并构建程序以报告 CPU 速度、屏幕尺寸等，但是不必考虑设备如何共享内容和控制。Dart 内容执行该项操作而不是设备内嵌的软件执行该项操作。

[0544] 应该知道这种移植提供了可管理的 N 的一阶方案的 Dart，而不是 N- 平方级方案的 Dart，并且对于每个新的设备只进行一次 DartPlayer 的移植，并且对于每个新的应用程序仅需要开发 Dart 一次，从而每个新的设备或应用程序仅要求增加一个工作单元。

[0545] DartPlayer 的实施例

[0546] 下面描述 DartPlayer（图 22500）的一个实施例，例如用 C++ 语言实现，并使用传统的编程工具编译为 DartDevice 处理器的本地指令集，或者在一些实施例中编译为中央处理单元 (CPU) 的本地指令集。

[0547] PlayerBase 类 (DartPlayer 类从其继承）执行作为一系列具有参数的操作码的 Dart 程序。基本上，DartPlayer 可以被模拟为微处理器，DartInstructionSet 被模拟为一组机器指令；然而，在为 Darts 的情况下，大多数指令比一般的微处理器中的指令更高级。DartPlayer 本地的指令可以例如，包括将 JPEG 图片解压缩到缓存中，将缓存中的图片以正确的格式移动到物理屏幕上的特定位置上，并保存运行的 Dart 和其所有代码和数据完整状态的单条指令。此外，可以有利地将完整的 2D 图形基元集、先进的用户输入处理，和音频解压缩处理指令构造为 DartInstructionSet。

[0548] 当 Dart 想要：(i) 将其自身发送；(ii) 发送其优化版本；(iii) 请求控制面板；(iv) 请求 Dart 程序；或者 (v) 从另一个设备请求数据时，它使用 EnumerationInstruction 或使用 BuiltinInstruction 将 EnumerationEvent 放入 EventQueue 中。Enumeration Instruction 或 Enumeration Event 使播放器调用 halBase 计数成员函数，该函数得到播放器可以连接到的每个 Dart 设备的名称和描述。可选地，可以将 DartProcedure 发送到每个这种设备，这种设备运行程序，并自己返回在初始设备上运行的程序。因此与一般的处理、数学、以及测试和控制指令相关的 EnumerationInstructions 可以被用于有效地检查任何连接的设备，以检查其是否能够实现大多数任何需要的函数集，或者包括使用中的或感兴

趣的代码、数据或内容。

[0549] 由于被发送到其它设备和接收回来的 DartProcedures 或 Darts 的函数能够包括进行大多数处理的代码，因此该功能性有利地是广范围的设备间协作任务所需要的所有功能。在多数情况下，应该由人或程序员将播放器移植到设备中，以决定当连接到其它设备时，采取什么类型的物理连接、协议、安全度量，等设计选择。在一个特别有益的执行过程中，DartPlayer 假设接收的任何通信模块都被纠正了错误，并已经通过移植程序员构建在 halBase 虚函数中的任何安全要求。有利地，除了在 HAL 的设备专用的函数中低级地发送和接收这种模块外，建立安全的通信会话、发送具有相关数据、代码和内容的事件，自动地请求和处理在传输中丢失的模块，自动地恢复设备之间临时失去的会话的功能都是以 DartEngine 的可移植的代码部分执行的。这就极大地降低了在 DartDevice 上支持不同通信协议所需要的工作量，并极大地提高了在设备之间执行的可靠度，这是因为使用相同的源代码基来移植 DartEngine 的可移植部分。

[0550] 对于物理发现使用 Dart 指令集的实施例

[0551] DartInstructionSet 的实施例有利地主要地以非常高级的功能性，处理设备、服务，以及资源发现和设备间通信。在一些实施例中，DartInstructionSet 专门地以非常高级的功能性，处理设备、服务，以及资源发现和设备间通信。尽管运行的 Dart 能够检查设备配置以确定实际的通信特征，例如通信速度和等待时间，但是有利地通常运行的 Dart 不关心或者不需要知道设备之间的实际通信是否是 HTTP，TCP IP，USB，802.11b，或一些其它协议，或共享的内存卡，物理传输的内存卡或任何其它的物理介质或协议。

[0552] 类似地，通过人指定和执行移植，可以将其它设备、服务，或资源或鉴别和安全的物理发现构建在 halBase 越权函数中。

[0553] 构建这种移植的一个有益的方式是将用户可编辑的“友好的”设备列表创建为基于 playerBase 的 DartPlayer 的一部分。在这种方式中，用户能够指定需要的互联网协议 (IP) 地址和 / 或网络名称和 / 或口令，以访问其希望访问的所有设备，以允许设备通过其它方式访问另外的未被发现的、难以发现的或者难以在其它许多联网的设备中通过其它装置准确地定位的设备。

[0554] 可以有利地实现本发明的实施例，从而所有的 DartPlatform 字符都是 32 位字，以有利地适应任何字符表示系统，例如统一代码或多字节标准，而不需要由 Dart 程序进行特别的处理。由 halBase 导出的 HAL 执行对象来回转化这些 32 位字符，以与特定设备的本地能力相匹配。然而，本发明不限于任何特定位大小的字，可以使用更大或更小的位大小的字。

[0555] 希望进行设备、资源和 / 或服务发现的设备应该遵循适当的标准。Dart 计数指令接口或 SignalEvent BuiltinInstruction(图 20670) 处理有利地隐藏用于 Dart 应用程序的不同设备发现标准之间的差别，同时提供帮助函数以在硬件抽象层中使用，以帮助建立兼容的对 IP、红外 (IR)、蓝牙、802.11x，和其它连接的或可连接的设备或系统的支持。在至少一个示例性 DartPlatform 上，Dart 有利地执行几乎所有的内容、数据和程序的适配，而不是将该适配嵌入在引擎中。正是 Dart 自己决定哪个再现程序将最佳地运行在目标设备上。并且正是 Dart 中的程序对连接的 DartDevices 改变内容回放。

[0556] 有利地，由于存在 DartPlatform，就不需要对每个应用程序和新的设备需要与

其一同工作的其它设备进行计划，这是因为设备之间的协议是非常简单的，但是也是非常有效的。一个 DartDevice 能够将 DartProcedure 发送到任何其它的 DartDevice 上，该 DartProcedure 自动地在目标 DartDevice 上运行，并接着返回自动地在初始设备上运行或处理的事件或程序。DartDevices 不需要知道他们正在运行的程序或 Darts 是否在传输数据、应用程序、控制面板或仅仅检查能力。

[0557] 在至少一个特别有利的实施例中，DartPlatform 不采用客户机 / 服务器或对等模式的设备间互操作性。相反，它有利地采用招募模式。

[0558] 招募和招募模型和方法允许 Dart 应用程序自动地将其到达或连接范围延伸到多个不同的连接和设备上。招募根本上不同于客户机 / 服务器或对等连接或通信并且赋予 DartPlatform 独特的和非常希望的特性。

[0559] 招募模型的实施例允许 Dart 应用程序以与人类形成组以完成工作相同的方式形成关于应用程序的设备组。如果某个人是希望建造房屋的施工承包者，则他为木匠和木材提供商确定施工所需要的技术和资源。接着他与因此被招募的提供商和木匠作为一个组进行工作，以使木匠得到木材并根据开始建造该建筑物的承包者的目的来协调该建筑物的建造。类似地，Dart 应用程序能够通过任何现有的通信介质通过发送能够自动地在目标设备上允许的程序，来达到和检验其能够进行通信的所有 DartDevice 的资源。运行的 Dart 自身找到 DartDevices，对其进行授权并形成 DartDevices 的组，将任何需要的内容和代码发送到组中的每个 DartDevice，以为用户实现应用程序。不要求设备用户的参与。

[0560] Darts 将其操作延伸到招募的 DartDevices 上，并有效地同时在所有 DartDevices 上运行。其结果是系统有利地不需要如客户机 / 服务器配置或操作中的任何中央控制设备。有利地，与对等配置和操作相比，Dart 系统不需要与初始 Dart 的丢失相关的程序存在于除初始设备外的任何设备上。并且设备用户永远不必考虑媒体格式、通信协议，也不必考虑加载驱动程序。无论何时另一个 DartDevice 被招募到组中时，招募模型还允许 Dart 应用程序和数据将其自身从 Dart 设备向 Dart 设备传播。因此有利地，通过使用或不需要任何明确地加载或保存应用程序或其相关数据，可以进行分配 Darts 和其封装的内容、程序和数据。安全性可能通常是 DartPlatform 的必要部分，以防止恶意的或出现故障的 Darts 或其它代码和数据的传播。该安全性部分地是以要求可能被招募的设备接受或通过不允许通信或简单地拒绝运行来自其它设备的程序或 Darts 的拒绝招募的形式出现的。

[0561] 在一个特别有利的执行过程中，招募模型和方法是基于 DartInstructionSet 的。使用 DartInstructionSet 来表示程序，其中对于设备互操作性、多媒体用户接口、以及招募模型和方法的使用来优化 DartInstructionSet。

[0562] 有利地，大多数高级语言可以被编译为面向 DartInstructionSet。此时在一个特别有利的实施例中，采用的高级语言是 C++，通过编译指示工具来增加扩展。这些优点都来自于 C++ 编程语言的当前的广泛使用，但是本发明不限于任何特定的语言，并且以现在已经存在的或将来开发的其它语言实现也能起到相同或更好的效果，其中包括 C++ 编程语言的改进、增强，和 / 或扩展。

[0563] 现在描述互操作性安全模型的其它特定实施例或更特定的实施例，DartSecurity 模型。在一个实施例 (1) 中，本发明提供了一种用于限制访问和或得知互操作性应用程序包的资源或能力，和或互操作性设备的资源或能力的方法，所述方法包括 :(1) 使用下述

步骤中的至少多个步骤形成安全性的基础：(a) 自动地收集与设备相关联的信息熵状态度量；(b) 生成密钥对；(c) 生成设备 ID；以及 (d) 在设备上存储密钥对、设备 ID，和信息熵状态度量，以当设备有效时继续使用；(2) 形成允许或阻止限制访问的规则；以及 (3) 使用形成的基础和形成的规则来至少为设备提供安全性任务。

[0564] 在另一个实施例 (2) 中，本发明提供了一种用于限制访问和或得知互操作性应用程序包的资源或能力，和或互操作性设备的资源或能力的方法，所述方法包括：(1) 当设备首先开始工作时在至少一个下述步骤中形成安全性基础：(a) 自动地收集信息熵；(b) 生成公共 / 私人密钥对；(c) 生成唯一的设备 ID；以及 (d) 在设备上存储公共和私人密钥对、设备唯一的 ID，以及用于随机数生成器的信息熵状态，以无论何时设备有效时继续进行使用；(2) 形成允许或阻止限制访问的规则；(3) 对下述安全任务中的一个或多个使用形成的基础：(a) 对于访问设备、应用程序和资源执行所述规则；(b) 保证规则自身的安全性，从而它们不会被未被授权的用户或代理修改；(c) 保证公共和私人密钥对、设备唯一的 ID，以及用于随机数生成器的信息熵状态的存储的安全性；(d) 保证操作在应用程序和设备之间共享的操作参数或数据的安全性；(e) 保证通信信道的安全性；(f) 对资源进行加密，从而它们只能被特定的设备或设备组，和 / 或特定的应用程序或应用程序集知道或使用，和 / 或当使用共享的密钥访问时被知道或使用；以及 (g) 生成统一的唯一的 ID 并使用其在所有时刻在所有设备上和或应用程序和或数据集或项上，识别有效的设备、应用程序、数据格式、集合、记录、单独的媒体文件，或甚至单独的数据项。

[0565] 在另一个实施例 (3) 中，本发明提供了 (2) 所述的方法，进一步包括：撤销或修改用于限制访问的规则。

[0566] 在另一个实施例 (4) 中，本发明提供了 (2) 所述的方法，其中限制访问资源和或知道资源包括下述中的一个或以任何组合的多个：(1) 执行允许设备彼此进行通信和或为何允许设备进行通信的规则；(2) 对在设备之间流动的数据进行加密，从而可能访问数据的其它设备不能得知或使用该数据；(3) 对数据、内容，代码或其它资源包签名，以确保自从签名以来不曾对这些包进行修改；以及 (4) 管理数字权限以关于下述动作中的一个或多个执行一组用于处理数据、代码内容，或任何其它可数字化表示的资源的规则集：共享、备份、打印、显示、执行、分配、回放、再现、运行、修改，或其任何组合。

[0567] XVIII. 社会同步互操作性方法 /Dart 社会同步

[0568] 通常以传统的方法实现多个设备上的数据集或操作的同步，其中在传统的方法中假设存在一个主设备，所有其它的设备直接与该主设备同步其数据集。这些方法在公司和政府的通用计算机网络中提供了高度稳定的同步系统，其中通用计算机网络中的计算机总是以可靠的高速网络彼此连接并由受过训练的专业人员进行管理和维护。现在使用相同的直接到一个主设备的方法，实现如移动电话、PDA、和数字照相机的个人设备的多数现代同步，其中主设备是个人计算机或通过互联网连接的服务器；然而，对于间断连接的通常为移动的设备来说要求同步到一个主设备是非常受限的，其原因包括：

[0569] 1 所有的设备必须与主设备共享至少一个通信协议。

[0570] 2 一些移动设备只能当他们与主设备紧密相邻时才能进行同步，因为它们仅有适于与主设备同步数据集的有限距离的有线或无线的直接连接。

[0571] 3 主设备为所有设备提供一个故障源。

[0572] 4 没有经过训练的各个用户必须在主设备上下载设备专用的同步软件并配置和维护该软件。

[0573] 5 对于移动设备来说没有使彼此直接同步的简单方式，甚至当它们彼此紧密相邻并共享共同的通信协议时也是如此。

[0574] 本发明的社会同步是一种特殊的方法子集，它能够用于同步 DartDevices，与所属领域当前状态中通常采用的传统的主从同步方法相比，提供了许多优点。再次想到社会同步是一种有效的并易于管理的方法，用于在任何数量的设备和协议上同步特定的数据组和或操作，而不需要每个设备都与主设备相连接，也不需要任何设备充当主设备。设备和内容的社会同步类似于人类共享信息和任务的方式，是所属领域的当前状态中通常采用的主从同步技术的有益替换。

[0575] 与所属领域当前状态中通常采用的传统的主从同步方法相比，本发明系统和方法的一些优点和好处包括：

[0576] 1 可以对于设备组进行同步，而不需要所有的设备能够直接与一个主设备进行通信；只要在设备之间存在通过任何数量设备的任何通信路径就能实现同步，并且不需要同时地建立这种路径。

[0577] 2 移动设备不需要直接与主设备相连接；而是，它们仅需要能够直接连接到为同步建立的设备组中的任何一个移动设备上。

[0578] 3 不存在主设备并因此没有单一的故障源。发生故障的编组的设备能够被新的设备代替，该新的设备能够容易地与组中的任何一个设备同步。

[0579] 4 不需要在任何一个设备上安装设备专用的软件，或者有效地维护或配置这种软件；而是同步组中的任何设备能够直接地将任何其它设备添加到设备组中。

[0580] 5 设备能够直接与组中的任何其它设备同步，其中通过设备之间的任何连接顺序存在到任何其它设备的任何连接路径，即使当这种连接不是同时可用时也是如此。

[0581] 社会同步的工作有些类似于与某人遇到的其他个人共享信息的人类团体，其中该其他的个人又与另外的其他个人共享相同的信息。人类模拟也能够用于说明主从同步的限制。设想一个拥有大量雇员的公司，其中雇员除了与一个上司谈话外不能共享关于他们同事的所有活动和最初想法的信息。存在许多需要分配所有需要的信息交互活动，从而所有需要通过一个上司，甚至通过固定的管理体系结构，共享的知识都是可用的。当各个雇员彼此相遇时，他们需要彼此直接共享共同关心的信息，并且希望该信息将被散布到需要通过中介了解的其他人。实际上，尽管社会同步的这些方法不是最理想的，但是它们是在一群或一组人或设备中散布信息的非常有效的方法，其中在人之间或在设备之间存在许多进行中的间断的与可能不规律的连接。

[0582] 图 30 表示如何在一个优选实施例中实现社会同步的例子。该例子表示到简单的联系列表的简单变化的同步，但是应该知道对于复杂的数据库和或不仅数据而且 DartDevices 和或 Darts 的组的操作及操作结果也可以进行同步。

[0583] 在图 30 的例子中，存在初始设备 A，具有最初包括两个输入的名字的联系列表 Dart 应用程序。其它两个设备，B 和 C 最初不了解 A 上运行的联系列表 Dart 的存在或特征。在一个优选实施例中，运行在设备 A 上的联系列表 Dart 计数其可以与其共享联系列表 Dart 的所有设备，其中联系列表 Dart 包括和控制联系列表。联系列表 Dart 采用 Dart 招募的方

法并可选地采用再现方法。设备A或设备B上联系列表Dart进行的招募是由用户通过选择“到其它设备的组”菜单项启动的。接着向用户显示当前可到达的设备和适于联系列表的编组的设备列表，该列表是由被发送以在可到达的设备上运行的一个或多个DartProcedures或Darts确定的，接着返回关于其成为被招募的设备的适合性的信息。用户从列表中选择联系列表Dart最终得出的设备B，保存包括一个或多个或所有再现程序和构成内容列表元素的数据的Dart。作为RunDart类型事件的一部分，该被保存的Dart被发送到设备B。当Dart在设备B上运行时，Dart进行自身保存，从而即使在设备B断开与设备A的连接或者断电后，也能在设备B上使用Dart。现在运行在两个设备上的Dart在每个设备的社会同步UID列表中进行输入（如果一个也不存在的话）。输入包括将有两个名字被共享的联系列表的UID，它是当首先创建该特定的联系人名称列表时由Dart生成的。使用该文章其它地方描述的DartSecurity基础设施生成UID。该输入还包括联系列表Dart的UID。DartTools自动地将UID放入生成的每个Dart示例中。

[0584] 类似地，在设备A完成其与设备B的永久的编组操作后的某个时刻，设备B被用于永久地编组到社会同步ObjectUID/DartHandlerUId列表，输入确定联系列表的特定编组逻辑示例的128位UIDA。该输入还包括存储在相同的设备上的Dart的UID，其中该设备能够用于实现同步该列表的联系。注意到通常地，处理程序Dart能够实现可以被表示为Dart的任何同步决定或操作或规则，并且在该例中不限于简单的数据同步操作。

[0585] 现在这三个设备都包含包括了UID和初始联系列表的两个名称的社会同步输入，即使设备C从没有直接与设备A通信也是如此。

[0586] 在该例中，在设备被编组后，关闭连接并且由运行ContactHandlerDart Z的用户将新的名字添加到设备C上的列表中。三个模块图表示在将第三个名字添加到设备C后的三个设备的状态。

[0587] 接着ContactHandlerDart Z或者运行在设备C上的任何其它Dart启动到设备A的连接。当建立连接时，设备C上的DartEngine自动地比较两个设备上的社会同步UID，并确定它们共享128位UIDA UID。设备C上的DartEngine自动地启动ContactHandlerDart Z，并且将协调数据的同步，而不管该数据是存储在Darts中或者被分离地存储。现在设备A和C包括了其联系列表中的所有三个名字。

[0588] 在设备B连接到设备A或设备C后，将类似地运行ContactHandlerDart Y，将实现设备B和设备B连接到的设备之间的名字的同步。注意到即使在同步之前A从来没有与C通信，但是它们知道需要进行同步并知道如何进行同步，这是因为在各个编组期间Dart和数据通过了中间设备B。

[0589] 可以从逻辑上得出这样的结果，即任何数量的设备能够被已经被编组的任何设备容易地过渡地添加到组中，即使新的设备直到被编组才了解联系列表/ContactHandler Dart或初始Dart也是如此。只要在设备组之间有足够的的专门连接，则所有的设备自动地保持共享的联系列表的高度同步，甚至当没有任何主设备时也是如此。

[0590] 本发明系统、方法，和计算机程序以及计算机程序产品还提供了串行化和同步的平台。图15、图16、图18中表示的设备内的DartRuntime和图17中表示的设备间的DartRuntime共同创建了单个事件驱动的DartRuntime，其中在所有被编组的设备的所有处理单元上有利地细致地串行化和同步所有操作。图9表示如何在包括在Dart

Recruitment、DartRenditioning、DartSource、DartTools、DartFramework、DartRuntime 以及 DartEngine 中的 DartPlatform 中实现该串行化和同步基础设施。这些系统、方法以及装置共同提供了一种用于确保 Dart 互操作应用程序的稳定的互操作性的基础设施，对部分 Dart 程序员而言只需要很少的工作来建立或管理复杂的交互作用和对在多个设备上或设备之间协作的处理单元的操作进行排序，甚至当出现通信错误和其它错误时也是如此。

[0591] 使用 DartPlatform 特别地使用 DartFramework 编写 Dart 应用程序自动地赋予应用程序在所有 DartDevices 上的二进制可移植性、稳定的电源管理、稳定的应用程序级错误恢复、简单并有效的设备编组能力，智能发现并使用设备上的资源，以及 DartPlatform 的许多其它的优点，对部分 Dart 程序员来说所有这些都需要相对较少的工作。如图 9 的例子所示，通过很大程度上自动的串行化和同步事件，用于产生这些优点的关键要素包括在 DartRuntime 中。

[0592] DartRuntime 协调进行互操作和通信的初始设备和被招募的设备之间的 Dart 事件和相关的文件的传递和交换。在任何数量的编组设备的事件队列中自动地备份和 / 或同步 Dart 事件。直接在所有设备的所有事件队列中分配被指定为同步的事件的事件，其中的事件队列共享同步的事件类型列表。如图 9 中的例子所示，以如下的方式实施：

[0593] 1 初始应用程序，Dart 700 的再现程序 701a 用例子为特定的未指定的设备间协作功能表示初始设备 400-1 上的连接管理器对象示例。连接管理器是在 DartPlarform（图 3）中提供的作为 DartFramework 的一部分（图 3102）的连接管理器类的示例。

[0594] 2 初始再现程序 701a 使用招募和再现招募设备组来创建设备组，共享的复制的事件类型列表在所有协作的设备上串行化和同步。该列表被每个编组的设备 (400-1，400-2，400-3，400-N) 上的连接管理器 420 的示例维护。

[0595] 3 无论何时将任何事件放入处于共享的列表中的任何编组的设备上的队列中，都首先将事件放入所有直接连接的编组的设备的所有队列 (660) 中，接着推迟事件在初始设备队列中的放置，直到接收到事件被成功地放入到所有直接被编组的设备队列中的确认信息。由于所有的其它设备将进行相同的操作，则初始事件不会被放入初始设备队列中，直到所有的其它编组设备在其队列中放入事件为止，甚至那些没有直接连接到初始设备的设备也是如此。这就确保如果存在任何阻止将事件放入在任何编组设备的队列中的错误，事件就不会被任何编组设备处理。通过顺序地将初始事件的应答与表示事件正在返回的状态的错误标记以及以事件结构示例的返回值字表示的错误码传播到发送方，来报告错误。

[0596] 4 为了防止一个设备生成的事件混乱地到达任何直接连接的设备，通过直到接收到前面发送的事件已经被成功地放置在队列中的确认信息，才允许任何新的事件放置在编组设备的队列中，来串行化将被放置在所有直接连接的编组设备的事件队列上的所有事件。需要该操作的情况的例子是第一事件是用于将新的幻灯片添加到共享幻灯片放映中的 ADD_SLIDE 类型的事件的情况，以及第二事件是显示将被表示的新的幻灯片的放映幻灯片 (SHOW_SLIDE) 类型事件的情况。由于其大小要小得多，因此非常可能是第二事件将在第一事件和应用程序试图放映幻灯片之前到达设备上，其中第一事件和应用程序试图是在第二事件到达设备之前试图放映幻灯片的。这就是为什么需要在所有直接连接设备之间串行化事件的原因。

[0597] 将串行化的事件发送到所有直接连接的设备上确保将以正确的顺序在所有编组

的设备上处理从单独的设备发送的所有事件；然而，当两个或多个不同的设备独立地以信号方式发送被标记为同步的事件时，还需要根据所有编组设备上表示的共享的同步的事件列表 430 同步事件的正确顺序。

[0598] 一种用于确保在所有设备上以相同的顺序处理所有同步的事件（即使当同步的事件需要被两个或多个设备独立地以信号方式发送时）的优选方法是通过使用保留的用于被连接管理器使用的 MasterSendEvent 事件类型。每个连接管理器将通过直接的连接招募它的设备看作其逻辑上的主机。在图 9 中设备 400-N 的逻辑主机是设备 400-3，设备 400-3 的逻辑主机是设备 400-2，设备 400-2 的逻辑主机是设备 400-1。因为设备 400-1 执行设备组的初始招募，因此设备 400-1 没有逻辑主机。这就使得设备 400-1 成为设备组的主机。具有逻辑主机的连接管理器的任何设备不会将事件放入其队列中，除非该事件被其逻辑主机放置，相反地，连接管理器将包括需要的所有信息以将事件以信号方式发送到 MasterSendEvent 事件类型的事件，并放置在所有其直接连接的编组设备中。所有这种 MasterSendEvent 事件最终将传播到设备组的主机。当将事件放置在其队列中时，知道其设备组主机的连接管理器将试图将重建的初始包括的事件放置在其队列中。如果包括的事件类型处于连接管理器的同步列表中，则以串行的方式将其传播到所有编组的设备上，就如同该事件类型是由设备组主机生成和以信号方式发出的。

[0599] 实际上，设备组主机是唯一被允许发送将被放置在其它编组设备的队列中的同步事件的设备。所有的其它设备需要请求主机为其发送任何同步的事件。由于所有的同步事件是由设备组主机通过串行信道发送的，因此设备组中的所有设备将以正好相同的顺序在其队列上接受所有事件，按照需要在所有编组设备上保持同步操作的完整性。

[0600] 注意到，在一个优选实施例中，同步事件的串行化能够被用于确保所有设备上的连接管理器保持正好相同的将被同步的事件类型列表。同时串行化系统能够被用于发送串行化和同步事件类型，该类型表示新的设备将被当作设备组的主机。由于串行化和同步宣布新的主机事件，因此所有设备将以正好相同的顺序接收这种事件，确保多个发送这种宣布不同新的主机的事件的设备将最终决定所有设备得到相同的宣布新的主机的最后的事件，从而最后处理的这种事件将会胜出。

[0601] 现在描述本发明的社会同步互操作性方法或更特别的 Dart 社会同步的选定的特定实施例。在一个实施例 (1) 中，本发明提供了一种用于在一个或多个动态产生的相同和 / 或不同设备的组中维持资源的同步的方法，其中的设备可能是间断地直接地连接和 / 或通过一系列直接连接间接地连接，这些直接连接可以是在其它编组设备之间建立的独立的间断的连接，所述方法包括：(1) 在互操作性设备上运行一个或多个独立可执行图像的初始互操作性应用程序包，该应用程序包逻辑地或物理地封装将被同步的资源和 / 或包括收集和管理将被同步的资源所需要的所有能力；以及 (2) 通过初始互操作性应用程序来对其它设备进行编组，其中对其它设备进行可能间断的连接，初始应用程序将互操作性信息传播到新的被编组的设备上。

[0602] 在另一个实施例 (2) 中，本发明提供 (1) 所述的方法，其中所述方法进一步可选地包括：(3) 重复对其它设备进行编组，其中运行在任何一个或多个编组设备上的互操作性应用程序包作为初始互操作性应用程序包起作用，从而设备组能够继续以有限的或可能无限的方式增长。

[0603] 在另一个实施例 (3) 中，本发明提供 (1) 所述的方法，其中所述方法进一步可选地包括 ：(4) 在设备中和在设备之间同步资源。

[0604] 在另一个实施例 (4) 中，本发明提供 (1) 所述的方法，其中所述方法进一步可选地包括 ：(3) 重复对其它设备进行编组，其中运行在任何一个或多个编组设备上的互操作性应用程序包作为初始互操作性应用程序包起作用，从而设备组能够继续以有限的或可能无限的方式增长 ；以及 (4) 在设备中和在设备之间同步资源。

[0605] 在另一个实施例 (5) 中，本发明提供 (3) 所述的方法，其中所述同步方法进一步可选地包括 ：(a) 在任何两个编组的设备之间启动通信会话 ；(b) 将一个设备上的同步唯一 ID 与另一个设备上的同步唯一 ID 相比较 ；(c) 运行与每个匹配的唯一 ID 相关的互操作性应用程序 ；(d) 在两个设备上传播互操作性应用程序的执行 ；以及 (e) 通过使其操作在两个设备上传播的互操作性应用程序，执行实现互操作性同步所需要的任何同步方法。

[0606] 在另一个实施例 (6) 中，本发明提供 (4) 所述的方法，其中所述同步方法进一步可选地包括 ：(a) 在任何两个编组的设备之间启动通信会话 ；(b) 将一个设备上的同步唯一 ID 与另一个设备上的同步唯一 ID 相比较 ；(c) 运行与每个匹配的唯一 ID 相关的互操作性应用程序 ；(d) 在两个设备上传播互操作性应用程序的执行 ；以及 (e) 通过使其操作在两个设备上传播的互操作性应用程序，执行实现互操作性同步所需要的任何同步方法。

[0607] 在另一个实施例 (7) 中，本发明提供 (1) 所述的方法，其中所述互操作性信息包括 ：(i) 被用于表示为特殊的同步目的编组设备的一个或多个互操作性通用的唯一 ID ；以及 (ii) 与通用的唯一 ID 相关联的一个或多个互操作性应用程序包，通用的唯一 ID 可以被执行以实现同步的目的。

[0608] 在另一个实施例 (8) 中，本发明提供 (6) 所述的方法，其中所述互操作性信息包括 (i) 被用于表示为特殊的同步目的编组设备的一个或多个互操作性通用的唯一 ID ；以及 (ii) 与通用的唯一 ID 相关联的一个或多个互操作性应用程序包，通用的唯一 ID 可以被执行以实现同步的目的。

[0609] 在另一个实施例 (9) 中，本发明提供 (1) 所述的方法，其中下述中一个或以任何组合的多个为真 ：(1) 至少部分地使用设备招募程序或 Dart 招募形成设备组 ；(2) 一个或多个互操作性设备是 DartDevice ；(3) 互操作性应用程序包符合互操作性格式或者符合 DartFormat 和或为 Dart ；(4) 独立的可执行图像是再现程序或 Dart 再现程序 ；(5) 资源是所描述的互操作性资源类型中的一种或多种 ；(6) 使用社会安全性程序基础创建互操作性通用的唯一 ID ；以及 (7) 使用招募方法或创造方法中的一个或两个生成一个或多个互操作性应用程序包。在另一个实施例 (10) 中，本发明提供 (1) 所述的方法，其中同步包括下述中一个或以任何组合的多个 ：(1) 以这种方式，即以维护下述一个或多个所需要的任何方式，不依赖于单独的示例进行变化，来维护独立的示例和 / 或共同确定的数据库示例的语义和 / 或可以被数据库或包括在数据库中的元素保存的任何东西 ：(a) 包括与共同的身份相关的指定的数据和目的的数据库的完整性 ；(b) 数据库的元素 ；(c) 数据库的定义和 / 或数据库的要素之间的相互关系 ；以及 (d) 数据库示例的共同身份 ；(2) 跟踪独立存储的和或在独立的设备上修改的项目列表上元素的添加、删除或修改 ；(3) 设备组收集和或处理和或比较信息 ；(4) 维护控制和限制对设备资源和能力的访问所需要的唯一的 ID、共享的密钥和安全设置的列表 ；(5) 管理设备间的设置，实现设备或一组设备的操作 ；(6) 在多个设备上

配置管理；(7) 在多个设备上安装软件或内容；(8) 获得设备和或设备资源或设备可达到的资源或一个或多个设备为其保存和或维护信息的资源的详细目录；(9) 在多个设备上分配或更新软件或内容；以及 (10) 上述的任何组合。

[0610] XIX. 社会安全互操作性模型 /Dart 社会安全

[0611] 再次想到社会安全是一种特别便于管理的方法，用于在可能进行间断连接的设备之间形成安全的网络。社会安全的工作方式类似于人类如何彼此信任的方式。社会安全的基础是使用 SocialSynchronization 传播利用 DartSecurity 系统生成的唯一的 ID 以及过渡性地从一个设备传播到另一个设备的允许的存取权限。从来没有直接地进行通信的设备通常会发现它们是设备组的一部分，其中允许所述的设备以一定的存取权限进行互操作而不需要进行其它的获得允许的操作。

[0612] 在安全性问题上，所属领域当前状态中的一个最大的问题是终端用户难于得知和管理安全方法，并最终根本不能使用安全模型。在使用的安全方法力度和管理的复杂度之间存在直接的联系。在传统的企业和政府的计算机网络中，认为高力度被认为是必须的并且必须由经过训练的全职专业人员管理。很少重新配置这种网络的事实使得对于专业人员来说管理的工作量是合理的。这些相同的传统的安全方法通常用于开发和产生个人使用的移动设备的网络，其中的个人没有经过训练并且没有时间或耐心进行确保设备网络安全的全职管理，而由于移动设备来回移动因此需要经常的管理。尽管本发明的社会安全方法不象企业和政府网络中采用的传统方法有效，但是能产生很好的安全水平，同时对于设备来说易于管理，当人们通常不使用或避开传统的方法时，大部分人实际上是采用社会安全方法的。因此在不另外地采用安全方法的情况下，社会安全有利地产生很好的安全水平。

[0613] 电子邮件是一个很好的例子。可以使用传统的安全电子邮件协议，并自从人们开始使用电子邮件开始将其建立在多数已有的电子邮件用户的客户机软件中；然而，很少有电子邮件的终端用户利用这些安全电子邮件协议。一个原因是得到、保护和使用证书并确保他们希望发送电子邮件到达的每个人也执行了得到、保护和使用证书所需要的管理，所需要的管理太复杂并且令人厌烦。另一个例子是难于在无线接入端口建立安全性，从而许多家庭无线网络都是不安全的。

[0614] 社会安全类似于保护互相活动的安全性的人类模型。一个人学习信任了解公司的专有信息的公司的新雇员，这是因为该新雇员是被公司的其它人介绍的。很可能老雇员从来没有与最初了解该新雇员的任何人进行直接交谈，就开始信任该新雇员是公司的真实雇员。不需要任何中心协调或跟踪就能建立这种信任网。如蜂窝电话、打印机、PDA、膝上型电脑、数字照相机以及便携式音乐播放器的设备通常共享媒体、信息或控制，但是它们从来没有一次共同连接在一起或没有连接到任何一个相同的中心设备源。但是希望使用安全方法来保护设备上的信息的完整性不会被其它设备或软件错误地使用或者破坏。社会安全方法是本发明的互操作性安全或本发明的 DartPlatform 的 DartSecurity 方法的特定子集，其中将访问权限从设备传递到设备，自动建立信任的设备、软件应用程序或 Dart 的网络，特定的一组访问权限具有设备的用户所需要的最少管理或训练。

[0615] 在该文章的其它地方描述了建立在 DartSecurity 方法之上的社会安全方法。DartSecurity(图 29) 确保每个 DartDevice 和 Dart 应用程序具有通用的唯一标识符，UID。每个 DartDevice 还维持其它 DartDevice 的 UID 列表，其中承认 DartDevices 的二进制标

记集所示的访问权限。对每个二进制访问权限标记集分配一个等级。

[0616] 社会安全是一种用于形成和保持设备组并在所有编组的设备上分配和保持具有共同的访问权限的设备列表的递推方法。图 31 表示左侧具有设备安全列表，被撤销的列表，以及社会安全的设备 UID 元素的 DartDevice。表示的社会同步列表是该文章中其它地方描述的社会同步方法的一部分，在优选的执行过程中被用于至少部分地在设备上递推地传播和同步 DartLists。

[0617] 图 31 的右侧表示编组前和编组后旧的 DartDevices 和新的 DartDevices 的状态的例子，以帮助说明社会安全方法。在该例子中，在编组前，旧的 DartDevice UIDA 具有应用程序和设备的 UID 列表，其中的应用程序和设备将被允许在相应与列表等级的标记的访问权限内进行互操作。在编组前，新的 DartDevice UIDZ 的唯一的设备安全列表允许设备自己在等级 9 与自己进行互操作。通常从生产商处与设备一起运送的一些 Dart 应用程序需要高级的访问权限，以配置设备和或设备的安全性方面，因此设备将对其上的应用程序授予高级的访问权限，如通过使设备自身的 UID 处于级别 9 列表中所示的。当新设备上的 Dart 开始设备与旧的设备的初次连接，并试图发送和运行由其自身的再现程序生成的 Dart，以在旧的设备上运行时，检查新的设备和 Dart 的访问权限。当然旧的设备根本不了解新的设备，并且在该例子中，也不拥有关于 Dart 应用程序的访问权限的任何信息，其中的 Dart 应用程序试图将其执行扩展到旧的设备上。在该例子中，根据 Dart 中的标记传输的访问需要，将通过每个设备上的用户接口请求用户允许这些设备进行互操作。图 31 的右侧下部表示在允许设备在级别 7 进行互操作后设备的状态，相应于准予 Dart 中指定的所有访问权限被传递到旧的设备的访问标记集。在通过用户接口得到用户的允许后，旧的和新的 DartDevices 现在都包含了相同的设备和 Darts 的 UID 的级别 7 的列表，其中允许设备和 Darts 进行互操作而不需要请求允许。注意到，如果新的 DartDevice 将试图向其新的级别 7 列表上具有相同安全访问标记要求的任何其它设备发送 Dart，则允许运行该新的 DartDevice，而不需要询问用户。实际上，设备为其了解的所有其它设备提供担保。如果设备的 UID 列表已经通过连接（即使与设备组中的其它设备间断地连接）在设备上分配，则信任的设备组将彼此信任，即使以前从来没有直接进行联系也是如此。在优选执行过程中，使用该文章其它地方描述的社会同步方法中的一些部分同步 UID 列表。

[0618] 应该注意到，需要用户参与和了解以形成和维护设备组的唯一的管理，是能够直接回答这种问题的，即在设备能够以需要的方式开始互操作前必须授予什么类型的访问。采用设备和 Darts 的简单任务促进访问权限的收集和分配，只有少量的易于了解的来自用户的允许请求。

[0619] 也可以采用社会安全来撤销设备的访问权限，以传播撤销的设备和 Darts 的 UID 的列表。在优选的执行过程中，不会允许在设备之间进行互操作的任何尝试，其中的设备是具有要求的权限的设备被撤销列表中之一，并且将删除用于适当的安全级别的被撤销的设备组列表中的 UID。

[0620] 现在描述本发明的社会安全互操作性模型的其它特定实施例以及 Dart 社会安全模型的更加特殊的形式。在一个实施例 (1) 中，本发明提供了一种用于自动地和递推地在互操作性设备之间传播访问权限和证书的方法，所述方法包括 :(1) 向每个互操作性设备分配唯一的 ID ;(2) 向每个互操作性设备分配一组初始的访问权限 ;(3) 向每个互操作性软

件包分配一个或多个唯一的 ID，并在包中嵌入所有和或每个独立可执行图像需要的唯一的 ID 集和相关的访问权限，其中的独立可执行图像是应用程序包的一部分；以及 (4) 当两个互操作性设备开放通信信道时，用于与唯一的 ID 相关的设备组或应用程序的任何现有的访问权限与其它设备上的访问权限同步，其中所述唯一的 ID 的限制性不比任何设备上的两个设备的互操作性的现有的访问权限的限制性大。

[0621] 在另一个实施例 (2) 中，本发明提供 (1) 所述的方法，其中下述中一个或多个为真：(1) 互操作性设备包括 Dart 设备；(2) 唯一的 ID 的基础是互操作性安全程序；(3) 访问权限是在互操作性安全程序中描述的；(4) 程序包是互操作性软件包和 / 或 Dart；(5) 作为设备招募程序的一部分开始通信信道的开放；以及 (6) 采用社会同步程序同步访问权限。

[0622] 在另一个实施例 (3) 中，本发明提供 (1) 所述的方法，其中以下述步骤实现所述方法：(1) 启动从运行在初始互操作设备上的互操作性软件包到目标互操作性设备的通信会话，以实现特定的互操作性目的；(2) 确定在设备之间和设备中是否存在用于指定目的的适当的访问权限；以及 (3) 如果存在适当的访问权限，则允许应用程序软件包通过向目标设备发送独立地可执行的图像并在具有需要的访问权限的环境中运行所说图像，将其执行延伸到目标设备上。

[0623] XX. 互操作性设备 /DartDevice

[0624] 再次想到 DartDevice 是能够进行高度的互操作的设备，这是由于 DartDevice 运行包含 DartEngine 的 DartPlayer，并且 DartDevice 将用于进行连接的至少一个通信协议传输到其它 DartDevice 上。在图 3(300) 和图 4(3000) 中描述互操作性设备和更特定的 DartDevice 的各个方面。

[0625] 在一个实施例 (1) 中，本发明提供了一种互操作性设备，包括：(1) 基于物理的图灵机 (Turing) 完整指令集的处理器与能够执行通用计算和输入 / 输出操作的物理存储器连接；(2) 至少一个用于与其它设备进行双向通信的装置；以及 (3) 在处理器上运行的互操作性引擎，互操作性引擎被封装在互操作性播放器中，以可执行的形式实现，可以在设备上加载和执行。

[0626] 在另一个实施例 (2) 中，本发明提供 (1) 所述的互操作性设备，进一步包括互操作性播放器。

[0627] 在另一个实施例 (3) 中，本发明提供 (1) 所述的互操作性设备，其中下述中独立的一个或以任何组合的多个为真：(1) 互操作性设备是 DartDevice；(2) 互操作性引擎包括 DartEngine；以及 (3) 互操作性播放器包括 DartPlayer。

[0628] 在另一个实施例 (4) 中，本发明提供用于以与其它相似或不同设备进行互操作的模式操作设备的方法，所述方法包括：(1) 提供可以在与物理内存相连的处理器中执行的，并且能够执行一般的计算和输入 / 输出操作的物理的图灵机 (Turing) 全部的指令集；(2) 至少在设备和另一个设备之间发送和接收双向通信；以及 (3) 操作处理器上的互操作性引擎，互操作性引擎被封装在互操作性播放器中，以可执行的形式实现，可以在设备上加载和执行。

[0629] XXI. 互操作性平台 /DartPlatform

[0630] 再次想到 DartPlatform 可以是 Dart 方法的任何组，能够实现 DartDevice 的指定、生成、智能编组，并且有助于在一个或多个 DartDevice 上传播并运行 Dart 互操作性应

用程序。在图 3 中和在该详细描述的其它部分中描述了互操作性平台，以及一般的互操作性平台的更特别的 DartPlatform 的各个方面。

[0631] 现在描述互操作性平台的其它实施例，其中 Dart 平台是一个特殊的例子。在一个实施例 (1) 中，本发明提供了一种用于在多个可能不同的设备上，以安全、可靠、有效和稳定的方式指定、建立、分配，以及实现独立地可执行的图像的互操作性软件包的目的的系统，所述系统包括：(1) 用于指定互操作性计算机程序软件包的互操作源；(2) 用于建立程序指令的互操作性工具；(3) 用于至少打包程序指令的互操作性格式；(4) 用于表示互操作性工具生成的程序指令的的互操作性指令集；(5) 用于在所有互操作性设备上运行互操作性软件包并提供共同的互操作性基础设施的互操作性引擎；以及 (6) 用于形成、分配，以及维护互操作性设备组的设备招募装置。

[0632] 在另一个实施例 (2) 中，本发明提供 (1) 所述的系统，其中所述设备招募装置进一步包括：用于通过至少一个通信链接，向不同于初始源设备的至少一个可到达的设备发送可操作的寻找具有需要的资源或能力的设备的检验程序的装置，所述检验程序包括以初始源设备和检验程序将到达的设备共用的可执行的形式编码的检验程序指令；用于直接地或间接地通过通信链接，在初始设备上从每个可到达的设备接收返回的响应的装置；用于通过在初始设备上执行的程序，分析从所有发出响应的可到达的设备接收的应答，来确定识别初始源设备和发出响应的可到达的设备的能力和资源的组合的利用计划，以最佳地实现软件应用程序的目的的装置；以及用于通过在初始设备上执行的应用程序将可执行代码、数据、内容，和 / 或 Dart 中至少之一分配到每个可到达设备中至少之一的装置，其中根据确定的利用计划可到达的设备被确定为具有需要的资源或能力。

[0633] 在另一个实施例 (3) 中，本发明提供 (1) 所述的系统，其中在所述系统中包括或使用下述可选组件中的一个或以任何组合的多个：(1) 互操作性框架；(2) 线性任务分配装置；(3) 垂直分层装置；(4) 应用程序驱动的电源管理装置；(5) 应用程序驱动的错误恢复装置；(6) 互操作性运行时间；(7) 互操作性应用程序驱动的运行时间；(8) 创造装置；(9) 虚指针；(10) 互操作性安全模型装置；(11) 社会同步装置；(12) 社会安全装置；以及 (13) 互操作性设备授权装置。

[0634] 在另一个实施例 (4) 中，本发明提供 (1) 所述的系统，其中下述中一个或以任何组合的多个为真：(1) 所述系统包括 DartPlatform；(2) 互操作性软件包符合互操作性格式或者为符合 DartFormat 的 Dart；(3) 独立的可执行的图像为再现程序；(4) 互操作性源为 DartSource；(5) 互操作性工具为 DartTools；(6) 互操作性格式为 DartFormat；(7) 互操作性指令集为 DartInstructionSet；以及 (8) 招募方法是 Dart 招募方法。

[0635] XXII. 虚指针

[0636] 传统的程序大多通常面向传统的处理器被编译和链接，传统的处理器可以直接寻址一个线性数据地址空间。处理器通常提供虚拟内存，以允许程序和操作系统通过执行虚拟内存的系统，直接地寻址比物理内存大的主内存，其中有限数量的真实内存块，或内存页被自动地交换到更大的但是更慢的存储设备内或外。这就有利地将程序员从必须关注直接管理在直接可寻址的快速主内存和慢速但是更大的存储设备之间的交换所需要的逻辑中解放出来。

[0637] 但是程序员必须经常关注内存管理逻辑，以使单独的数据结构不会彼此冲突地寻

址，因为这些数据结构在执行程序的过程中动态地改变大小。动态地分割和管理共同使用的单个地址空间通常是复杂的并易于出现故障，甚至当使用虚拟内存技术产生更大的有效的数据地址空间时也是如此。

[0638] 传统的虚拟内存技术的另一个限制是处理器仅支持固定大小的真实内存页，而不考虑正在运行的程序的特征，或者按照要求真实内存大小是变化的方式用于多个程序，其中的大小并不是最优地匹配所有同时加载的程序或程序的部分的需要。

[0639] 在本发明的一个执行过程中，在硬件处理器中或软件指令集执行引擎中采用多个复杂的寻址逻辑，以提供多个独立的地址控制，被每个程序使用，并且允许真实的内存页的数量和其大小有利地根据可用的真实的主内存的物理大小、速度和性能特征，以及希望的特定程序到特定数据集的访问方式，进行变化。在源代码、编译器和链接器中支持本发明的编程语言扩展，以提供对本发明的虚指针的应用。采用虚指针具有下述优点：

[0640] 1 多个大的独立的地址空间。可以在其自己的不同的虚指针地址空间中保存每个动态地重新规定大小的数据结构，从而不会出现与在其它地址空间中的数据结构的冲突。

[0641] 2 对于每个独立的地址空间的真实的内存页的数量的应用程序专用的控制。可以根据希望的访问方式来设置优选的真实的内存页的数量，以限制要求的真实的内存的数量，和或限制需要在进行读写的慢速内存块的数量，以在真实的内存和存储器之间交换数据。例如，如果总是在零点处访问大量的数据，并以线性顺序继续，则多个内存页将是多余的。可以将内存页的大小设置为实际存储器的最优的访问内存块大小，以确保良好的性能、速度，或者限制访问存储块的数量，从而不会耗尽设备。

[0642] 3 对真正的内存页大小的设备专用的控制，以匹配存储设备性能特征或提高存储设备的寿命。通常，设备采用闪存进行存储，在设备开始出现故障之前，闪存具有已知的有限数量的保证的存储块访问。硬盘通常在特定大小的存储块中存储或缓存数据。

[0643] 4 程序员不需要预测或管理数据结构或列表需要的内存数量，因为虚指针自动地逻辑地包括地址空间的值，这种地址空间的值大于使用这种数据结构或列表访问的希望的最大地址空间。

[0644] 5 以独立于内存页大小或内存页数量的形式自动地保存数据。可以使用 DartSource 设置每个虚指针变量的参数，该虚指针变量表示采用 DartInstructionSet SAVE_INSTRUCTION 的 Dart 的保存是否自动地保存地址控制的完整的数据状态。在一个优选的执行过程中，将被保存的每个虚指针的值保存为具有值范围的解析列表的 DartPart，而不考虑目前被运行的 Dart 使用的内存页大小或内存页数量。这就使得可能在其它设备上－运行保存的 Dart，其中在其它的设备上需要改变内存页大小和或内存页数量。

[0645] 6 有效地缓存来自更大的并且可能更慢的数据存储器的数据，最小数量的宝贵的主内存 RAM 允许应用程序运行，就如同它们具有比实际上大得多的 RAM 内存一样运行。

[0646] 7 用于索引数据库操作的简单的和有效的基础设施，其中在不同的虚拟地址空间中保存数据和索引。

[0647] 在优选的执行过程中，DartSource、DartTools，以及 DartEngine 支持虚指针，以提供上述列出的优点。

[0648] 在 DartSource 中，使用 C 或 C++ 语言的扩展，#pragma 虚指针（参数），来指定每个虚指针，其中参数包括：

[0649] 1 当程序开始执行时，保持虚拟地址空间的起始地址的指针变量的名称。

[0650] 2 建议使用的真正的内存页的数量。

[0651] 3 表示地址空间中的值是否随应用程序自动地保存的二进制标记。

[0652] 4 建议的真正的内存页的大小。

[0653] DartTools 处理 DartSource #pragma 语句，以建立将加载指针地址值的 Dart，该地址值有效地作为分离的地址空间的起始偏差。该起始偏差实际上是多字段地址的一部分，多字段地址由 DartEngine 解释以确定特定的虚指针。

[0654] 在图 32 中表示 Dart 虚指针的例子。在优选的执行过程中，Dart 的地址空间的单位是 32 位字，而不是更普遍的多数传统的处理器的 8 位字节，传统的处理器最大的可直接寻址的空间是 2^328- 位字节。DartPlatform 使用 32 位地址的两个最高有效位作为表示将被使用的地址空间的字段。由于 DartEngine 执行的 DartInstructionSet 的可寻址单位是 32 位字，因此在可编址内存的大小中不存在损失，为 2^30 32 位字。图 32 的例子表示地址空间类型字段表示这是虚指针地址空间的地址。后面 5 位字段表示使用虚指针 1，在该执行过程中意味能够存在多达 2^5 = 32 个不同的虚指针地址空间。剩余的 25 位是虚指针 1 的地址空间中的数据的偏移或地址。

[0655] 图 32 的例子表示对于具体的地址，0x50001，在特定设备上的 DartEngine 如何管理虚指针 1 的存取的具体示例。由于块（页）大小是 64K = 2^16 个字，因此 25 位偏移量的前 9 位表示哪个 64K 块包括被存取的数据值。在所示的例子中，两个真实的内存页目前都不保存该块。同时用于虚指针 1 的闪存块文件都不保存需要的数据块。这些文件是过去存取的结果，要求用于块 100 和 101 的真实的内存页在过去被其它页代替。在代替真实的内存页之前，DartEngine 自动地在两个闪存文件中保存这些块的数据值。对于该次存取实际上当前的数据值仍然存在于虚指针 1 部分的运行的 Dart 文件中。当使用数据值最后保存运行的 Dart，接着保存在其虚指针 1 的地址空间时，就产生了该部分。在该例中当块 101 的数据值从这部分的第三范围读入两个真实的内存页之一时，就完成该次存取。如果将被代替的真实的内存页具有被改变的或被污染的数据集时，则将写出新的闪存文件以在其值被代替之前在真实的内存页中保存当前值。当返回指向现在处于真实内存中的 0x50001 值的指针，以由存取 Dart 指令处理时，就完成了该次存取。

[0656] 现在描述虚指针的其它方面和实施例，以及其相对于本发明的其它方面的使用。在一个实施例 (1) 中，本发明提供了一种用于在软件应用程序中提供多个大的独立的可存取的数据存取地址空间的方法，其中的地址空间有效地利用物理内存和或物理存储设备，所述方法包括：(1) 指定一个或多个独立的地址空间的性质；(2) 将计算机程序代码源语句处理为适于在特定的软件引擎和或支持虚指针功能的硬件处理器上运行的可执行图像；以及 (3) 在特定的软件引擎或执行指令集的硬件处理器上运行可执行图像，该指令集通过采用快速存取限制大小的主内存和较慢存取但是更大的二级存储器，以有效的方式决定对地址空间引用的数据进行存取。

[0657] 在另一个实施例 (2) 中，本发明提供了 (1) 所述的方法，其中指定一个或多个独立的地址空间的性质是采用计算机编程语言实现的，所述计算机编程语言支持遵循语法和语义的语句。

[0658] 在另一个实施例 (3) 中，本发明提供了 (1) 所述的方法，其中所述处理包括运行软

件工具，其中所述软件工具将源语句处理为适于在特定的软件引擎和或硬件处理器上运行的可执行的图像。

[0659] 在另一个实施例 (4) 中，本发明提供了 (1) 所述的方法，其中至少部分地通过下述步骤实现有效地利用和有效的方式：(1) 对于每个独立的地址空间，建立和维护许多真实的内存页，它们均为相同的特定大小；(2) 当在独立的地址空间中逻辑地进行数据存取时，决定存取并返回一个值或指向包括这个值的主内存的指针到进行存取的指令或程序，采用了下述程序：(a) 确定数据是否处于主内存页之一中；(b) 如果数据没有处于一个主内存页中，则执行下述程序：(i) 确定数据是否处于二级存储器中；(ii) 如果数据没有处于二级存储器中，则执行下述程序中的一个或多个：(1) 返回默认值；(2) 返回指向包括默认值的主内存数据元素的指针的指针；以及 (3) 返回存取未知数据代码或其它指示，即不存在用于数据的已知值；(c) 如果数据处于二级存储器中，则执行下述程序：(i) 选择真实的内存页，并开始对包括需要的数据值的数据块寻址；(ii) 如果所选择的真实的内存页被标记为被污染，则写出或另外地使所选择的真实的内存页中的值被写入到二级存储器中；(iii) 读取或另外地加载连续范围的可编址数据到真实的内存页中，它与所选择的真实的内存页具有相同的大小和相同的新的起始地址；(iv) 将该真实的内存页中的数据标记为未被污染；以及 (v) 返回现在处于所选择的主内存页中的数据值或者指向现在处于所选择的主内存页中的数据的指针。(d) 如果数据处于一个主内存页中，则执行下述程序：(i) 如果已知存取表示这个值的使用者可能改变数据值，则将主内存页标记为被污染；以及 (ii) 返回处于主内存页中的数据值或者指向主内存页中的数据的指针，其中发现该主内存页包括该数据。

[0660] 在另一个实施例 (5) 中，本发明提供了 (4) 所述的方法，其中如果数据没有处于真实的内存页中并且没有处于二级存储器中，则如同数据处于二级存储器中来进行处理，除了不用从二级存储器加载所选择的带有数据的真实内存页外，还将选择的真实内存页填充默认值。

[0661] XXIII. 示例性应用程序和应用程序环境

[0662] A. 中微子检测器应用程序例子

[0663] 参照图 21 所示，描述了本发明实施例的例子，表示通过采用 DartInstructionSet 的 OEM_BUILTIN_INSTRUCTION，DartPlatform 如何被用于产生新型 DartDevice，家庭中微子检测器 3500 的本地功能，甚至当移动电话 DartDevice3600 最初不了解中微子检测器 3500 的存在或特征时，也能发现和使用中微子检测器 3500。

[0664] 家庭中微子检测器 3500 目前仅在理论上存在，但是如果真正存在的话，应该知道没有与这种设备的互操作的现有的标准。然而 DartPlatform 提供了用于使中微子检测器 (ND) 与其它 DartDevices（在该例中为移动电话 3600）能够进行互操作的方法，即使移动电话 DartDevice3600 的制造商不了解现有的标准或这种设备的功能也能够进行互操作。

[0665] ND 的制造商是 Mirabella Electronics，它被分配唯一的 ID，MIRABELLA_ELECTRONICS_ID，从而使 Mirabella Electronics 制造和使用的任何唯一的 OEM 功能与所有其它制造商制造和使用的功能相区别。Mirabella 希望其设备能够与 DartDevice 进行互操作，将 DartEngine 作为运行在 ND 上的 DartPlayer 的一部分进行移植。为了表现设备在五秒的时间段内采样通过检测器的中微子的数量的独特能力，从 HAL 的 OEM(<oemld>，

<subfunction>) 方法调用本地嵌入的函数 CollectSamples，所说函数产生收集和返回通过检测器的中微子的数量。接着 MirabellaElectronics 采用 DartSource 和 DartTools 创建控制面板 Dart，该控制面板 Dart 包括再现程序 R1 和再现程序 R0，R1 显示用于控制面板的用户接口，R0 处理被保留为表示将进行采样的 Dart 专用的类型的事件。

[0666] 具有用户接口能力的每个 DartDevice 通常将被与存在于其上的 GetControlPanels Dart 一同运送，以用于发现所有实现为可到达的 DartDevices 上的 Darts 的控制面板，该 DartDevices 愿意于其它 DartDevics 共享其控制面板 Darts。当 GetControlPanels Dart 在移动电话 3600 上启动时，发送 GetControlPanelList DartProcedure，以通过任何通信协议在所有可到达的设备上运行。在这种情况下，移动电话通过两个设备共用的蓝牙协议发现并建立通信会话，接着将 DartProcedure 作为 RunProcedure 类型事件的一部分发送到运行 DartProcedure 的中微子检测器。当 DartProcedure 在 ND 上执行时，通过使用作为每个 DartDevice 的 DartInstructionSet 的一部分的嵌入的指令，收集 ControlPanel Dart 的名称、描述以及唯一的 ID 的列表。存在于 ND 上的可用控制面板 Dart 的名称和描述被发送回移动电话，并被 GetControlPanels Dart 显示以由用户选择。在从移动电话上的列表中选择控制面板后，发送事件，请求通过唯一 ID 确定的控制面板 Dart 开始在 ND 上执行，并接着通过建立的通信会话招募移动电话，这就自动地将电话上的 GetControlPanel Dart 替换为运行的 R1 再现程序，其中 R1 再现程序是由进行招募的 ND 的并且运行 R0 再现程序的 ControlPanel Dart 发送的。

[0667] 再现程序 R1 则在移动电话屏幕上显示大的绿色采样按钮。当用户选择该按钮时，R1 产生标记为同步的事件，请求进行采样。DartRuntime 自动地将事件发送到编组的 ND 的事件队列中，这就使得通过使用 OEM_BUILTIN_INSTRUCTION 处理事件，来通过使 DartEngine 调用 halOEM() 的方法实现采样，其参数被 Mirabella Electronics 分配，以生成 CollectSamples() 本地设备函数和将采样数返回到 halOEM() 的调用者，接着被引擎返回到执行的 R0 再现程序。R0 接着调用以将标记为同步的 DisplayNumberOfSamples 类型事件放置在其队列中，这就使得将 DisplayNumberOfSamples 类型事件放置在编组的移动电话的队列中。运行在移动电话上的 R1 再现程序接着处理事件和在屏幕上显示检测到的中微子数量，它被包含在 DisplayNumberOfSamples 类型事件的参数中。

[0668] 因此图 21 的例子说明 DartPlatform 能够被有利地用于将甚至非常独特的设备的能力和控制展示和延伸到 DartDevices 上，而事先并不了解该独特设备的存在、特征或能力。

[0669] 上述例子和描述中的一些要点包括：

[0670] 首先，其独特的目的不能被标准委员会考虑的设备能够成为 DartDevice400，需要几乎与使任何其它设备成为 DartDevice400 相同的开发工作量。

[0671] 第二，该新的设备能够容易地将其独特的硬件能力展示给在任何这种类型的设备存在之前就已经存在的其它设备。已有的 DartDevice400 能够与新的设备进行互操作，控制对于任何设备来说是新的、独特的和未知的活动。

[0672] 第三，使中微子监测器 3500 成为 DartDevice400 所需要的开发工作类似于使任何设备成为 DartDevice400 所需要的开发工作，对于任何其它设备不需要任何其它工作，即使这些设备在出现该技术前就已经存在时也是如此。即，当采用传统的互操作性方法时以

N- 平方的方式出现的开发工作量，在采用 DartPlatform 中包括的方法实现时，仅是 N 的一阶的工作量。因为 DartApplication 的再现程序于来自相同的初始 Darts 的其它再现程序进行通信，因此与传统的互操作能力相比可以获得高度的可靠性，传统的互操作能力可能需要分别的执行和分配更可能会不兼容的互操作性组件。

[0673] B. 幻灯片放映应用程序例子 - 开发到使用的步骤

[0674] 现在我们更加详细地以例子描述示例性的系统、方法、计算机程序和组件，该例子采用幻灯片放映 Dart 应用程序作为基本的示例性应用程序，描述开发到使用的步骤。该幻灯片放映的例子仅是为了说明目的的，应该知道对于实际上任何软件应用程序（包括与硬件交互作用或利用或控制硬件的软件应用程序，实际上都需要在任何设备或设备组上运行，不管是顺序地每次在一个设备上运行、同时或并行地在多个设备上运行）可以有利地采用相同的技术。

[0675] 在该例子中，设计 DartSource（图 3100）文件 (C++) 和 DartFramework（图 3 102）类定义和源代码（图 3 101）作为表示构建互操作性应用程序需要的算法和代码的可选基础。

[0676] 参照图 3 所示，示例性幻灯片放映 Dart 应用程序作为 DartSource 文件开始（图 3100）。应用程序代码 101 通常符合标准的 C++ 语言。幻灯片放映 DartSource100 还包括标准的 C++ 源文件，它表示 DartFramework102。DartFramework102 是一组类定义和代码，被设计为作为表示构建互操作性应用程序需要的算法和代码的可选基础，该应用程序将被再现为 DartFormat300 并符合 DartRuntime8000（参照图 15 所示）。

[0677] 通过作为 DartInstructionSet 一部分的 BUILTIN_INSTRUCTION（图 20670），DartSource100 调用 Dart 引擎函数。通过编译器 201 生成的 BUILTIN_INSTRUCTION 670（参照图 20 所示）（它是设备独立的 DartEngine 600 执行的 DartInstructionSet 的一部分），C++ 源代码调用引擎函数，其中 DartEngine 600 是运行在 DartDevice 400 上的设备专用的 DartPlayer 500 的一部分。

[0678] 对 C++ 语言进行有利的扩展，从而可以从应用程序代码 101 和 DartFramework 102 引用许多类型的资源 103。这是通过特定名称的嵌入函数、特定名称的数据结构，以及具有 Dart 编译器 201 认可的关键字的 #pragma 扩展实现的。

[0679] 资源可以包括文件内任何地方的，或动态生成的和通过文件系统（参照图 22612、图 27 5100、图 28 5000、图 26）或任何形式的网络访问、无线访问，或任何形式的通信可由 DartTools 200 到达的任何类型的数据，或代码，或对数据或代码的引用。拿幻灯片放映应用程序举例来说，资源 103 可以例如包括 JPEG 格式的背景图像文件、PCM 格式的音效文件，和 / 或任何图片幻灯片或图像、视频、文本、字符，或者甚至作为默认的演示幻灯片包括的完整的 Dart 文件 (700)。C++ 语言的 Pragma 扩展还可以用于确定 DartTools 200，其函数被自动地加入到 DartProcedures 4000（参照图 14 所示）或部分 303（参照图 13 所示），而不是被包括在 MainCode2103 和 MainData 2102（参照图 13 所示）部分（图 3303）中。C++ 语言的扩展 partnumber() 嵌入函数可以被用在源代码中，以使编译器 201 生成该部分 ID 的 32 位数值，当用作函数或嵌入指令 (DartInstructionSet 的一部分）的参数时，用于引用该部分。

[0680] DartFramework 102（参照图 3 和图 11）有利地包括类说明和实现的体系结构。一

个关键的基类是 Gizmo 115 类。Gizmo 115 可以被用于封装一个处理的函数单元。Gizmo 115 导出的类包含 Setup() 和 Process() 方法，以及对双亲 Gizmo 和子女 Gizmo 的列表的引用。这些引用可选地可以为空引用。

[0681] 再现程序 114 类是从 Gizmo 类继承的，并被设计为用作一部分应用程序的起始点，其中的一部分应用程序能够独立于应用程序中的其它再现程序 114 被加载和运行。

[0682] MasterRendition 113 类是从再现程序 114 类继承的。这是 MasterDart230 中的唯一有效再现程序（参见图 12）。MasterDart 230 是二进制 DartFormat 300 文件或从 DartTools 200 输出的图像（参照图 3 和图 12 所示），优选地包括一个 MasterRendition 示例。MasterDart 230 可以有利地包括在源代码中引用的所有的代码、数据、内容、程序，以及资源，并且当在 MasterPlayer 203 上播放时被自动地加载。在加载 MasterDart 230 后，MasterPlayer 203 执行 C++ 构造方法，C++ 构造方法应该在启动 main() 输入点之前运行。MasterRendition 113 的 main() 方法用作 MasterDart 230 的逻辑开始点。

[0683] 图 12 表示将 DartSource 100 输入到 DartTools 200 中的实施例。编译器 201 将 DartSource 转换为 DartInstructionSet，一次将一个文件转换为对象 220，并将这些对象收集到库中。可选地，可以由库工具创建这些库。对象和 / 或库 220 也可以用作其它 DartMasters 和 Dart 应用程序的 DartSource 100 的资源 103。除了某些大的差别外，编译器 201 的操作、可选的库工具，以及链接器 202 通常类似于现在使用的传统的 C++ 编译器、库以及链接器。这些差别中的某一些包括：

[0684] 首先，目标指令集来自于 DartInstructionSet。第二，输出的可执行指令是将在 MasterPlayer 203 上执行的 MasterDart 230，它不是面向实际终端目标设备；并且，MasterDart 和 MasterPlayer 表示本发明特有的组件。第三，许多在使用后被丢弃的传统的编译器、库和链接器的操作中生成的类定义和链接结构（使用后通常被丢弃）被保存在 MasterDart 中，以被 MasterPlayer 使用。被保存的类定义和链接结构也不同于可以暂时存在于传统的编译器、库和链接器中的类定义和链接结构。第四，DartTools 200 处理源语言的扩展：(i) 指定或自动地收集执行过程中需要的运行时间资源要求；(ii) 作为部分 303 包括在资源 103 的 MasterDart 中（图 3、图 13）；以及 partnumber() 源代码函数提供用作被 DartTools 200 分配的资源的标识符的部分数量，以被应用程序使用。其它的发明性在于最终的 DartTools 输出的 MasterDarts 和 Darts 可以包括任何数量的再现程序和信息，这些再现程序和信息是将 Dart 或 MasterDart 图像中可能被共享的部分放在一起以按需要构造这些再现程序所需要的。另外的发明性还在于 DartTools 输出的 Darts 和 MasterDarts 包括实现招募所需要的程序、数据、代码和内容，以智能地建立专用的设备组，并根据 DartRuntime 实现应用程序的目的。

[0685] 编译器 202（图 12）组合对象 / 库 202 并将其打包为符合 DartFormat300（图 3）的单个二进制图像。该图像为 DartMaster 230。DartMaster 230 将在 MasterPlayer 203 上运行，DartMaster 230 包括 DartEngine 500（参照图 22 所示）。该 DartEngine 500 包括一些 BuiltInInstruction（参照图 25 和图 20 所示），它们支持调用利用符号表部分 2100（参照图 13 和图 14 所示）的 DartEngine 系统函数，以及编译器 201 和链接器 202 生成的其它图元数据部分。这些函数可以被 MasterDart 中的代码用于帮助收集例如，资源 103，初始化数据，以及将部分 2100 和部分 2200 组合为 Dart 700 或 Darts 组 700。通过将被生成的

Dart 或 Darts 中需要的部分 2100 和数据，这些函数也有助于减少代码的大小，或增加代码的处理效率。

[0686] MasterDart 符合 DartFormat 300（参照图 3 所示），但是可能限于仅有一个 RenditionTable 2104，RenditionTable 2104 被 RenditionTable 2101 中的 PartId 引用。PartId 是用作 Dart 700 中的部分（图 12303）的参考值引用的整数值标识符。在一个实施例中，PartId 是 32 位，但是是具有不同计数位的标识符，例如 16 位、24 位、40 位、48 位、64 位或可以使用的任何其它位数。在一个实施例中，在 DartSource 中仅允许一个 MasterRendition 对象示例。DartTools 200 利用源代码中的信息用于初始化 MasterRendition 113 对象示例，利用其引用的数据结构初始化 RenditionsTable 2101 记录中的参数。相应于 MasterRendition 113 示例的 RenditionsTable 2101 记录中的一些参数可以被 DartTools 200 自动地填充。这些 RenditionsTable 参数包括事件和文件的数量，当再现程序 114 对象示例首先被加载并准备由 DartPlayer 500 运行时，应该最初分配堆内存的数量。在任何 Dart 700 运行在任何 DartPlayer 500 上之前，运行在 DartDevice 400 之上的 DartPlayer 500 的 DartEngine 600 应该有利地首先选择并加载再现程序。

[0687] DartPlayer 500 在 DartEngine 600 周围提供设备专用的应用程序可执行壳（shell），DartPlayer 500 能够指定运行的再现程序 PartId，或能够指定零（“0”），这在一个实施例中表示其为无效的 PartId。如果指定零，则 DartEngine 将加载具有 RenditionsTable 2101 的第一记录的 PartId 的再现程序 114。

[0688] 为了发现 RenditionsTable，DartEngine 600 应该首先加载跟踪器 305（参照图 3 所示），它在一个实施例中应为 Dart 700 文件中的最后四个 32 位字。跟踪器 305 包括自 Dart 700 文件开始的 PartTable 304 的偏移。PartTableRecored 314（参照图 14 所示）中的每条记录，在 PartTable 304 中包括 PartId，例如 32 位 PartId，以及自 Dart 700 文件开头的 startOffset 和 endOffset，其中在 Dart 700 文件中找到部分 303 的连续图像。

[0689] 通常在 DartFormat 300 文件中仅允许一个 RenditionsTable 2101（图 13），它具有被永久分配的 PartId 值，该值被引擎用于寻找相应于 RenditionsTable 2101 部分的 PartTableRecord 314。该 PartTableRecord 314 中的偏移用于寻找 RenditionsTable 2101，RenditionsTable 记录包括关于 MainData 2102 和 MainCode 2103 部分中的线性分段的信息，MainData2102 和 MainCode 2103 是该记录中 PartId 引用的再现程序所需要的。RenditionsTable 2101 记录还包括开始再现程序 114 示例的执行的开始代码图像地址。最佳地 RenditionTable 2104 记录仅是一个 32 位字，它保存属于再现程序 114 示例的部分的 PartId。

[0690] 在本发明的一个实施例中，有几个与在 DartPlayer 203 上加载 Dart230 相关联的步骤。使用相同的步骤在 MasterPlayer 上加载 MasterDart，这些步骤分别是 Dart 和 DartPlayer 的特定示例。这些步骤可以包括下述步骤，包括可选的但是有利执行的步骤：

[0691] 步骤 1 应用程序开发设备 900 上的 DartPlayer 203（参照图 12 所示）调用 DartEngine.Init() 方法（图 246000、图 234002），具有规定 RenditionPartId 从 0 值开始的参数。

[0692] 步骤 2DartEngine 600 接着从 Dart 230 文件的末端读取跟踪器，并取出 PartTable 304 的偏移。

[0693] 步骤 3 读取 PartTable 304 的每条记录，直到对于所有 RenditionsTables 的 Partld 发现具有预先分配的固定值的记录为止。在任何 DartFormat 图像中只允许一个具有该固定值的 Partld 的 RenditionsTable。

[0694] 步 骤 4DartEngine 600 取 出 RenditionsTable 2101 的 startOffset 和 endOffset，并使用 startOffset 和 endOffset 读出第一 RenditionsTable 记录。

[0695] 步骤 5 读出第一 RenditionsTable 记录，提取初始运行时间参数，例如为其分配内存的文件和事件的数量，为堆和堆栈分配的空间数量。还提取出 RenditionsTable 2104Partld。

[0696] 步骤 6RenditionsTable Partld 用于从用于该 Partld 的 PartTable 304 中寻找 RenditionsTable 2104 的 startOffset 和 endOffset。

[0697] 步骤 7 读出 RenditionsTable 2104 中的每条记录。在一个实施例中，每条记录仅是一个 32 位字，具有属于再现程序的部分的 Partld。每个 PartTableRecord 314 包括管理在加载时发生的事件的标记和参数。例如，如果根据 contentType、参数 0、参数 1 和 / 或参数 2 的值，或者根据一些其它的参数或条件，以某种方式处理这一部分，则可能存在标记。MainData 和 MainCode 部分是这种部分的例子，即其必须在加载时被处理，以在再现程序开始执行之前将需要的代码和数据放入内存中。

[0698] 步骤 8 当发现需要的 MainCode 2103 PartTableRecord 314 时，contentType 将保存被分配给 MainCode 2103contentType 的固定值，并且设置标记参数字中的自动加载的标记位，参数 0 和参数 1 将保存包括在 MainCode 2103 部分中的第一和最后 DartInstructionSet 代码图像地址。尽管从广泛可用的 C++ 工具输出的多数可执行图像将产生总是在相同的固定偏移开始的代码图像，但是对于 Darts 却不总是如此，因为对于再现程序仅需要的线性连续的区域，它被作为运行 MasterDart 230 的结果输出或者保存 Dart 的一组再现程序，以进行备份或发送到另一个编组的设备。换句话，Dart 代码图像不总是在相同的固定偏移开始的，仅可以正常地显示再现程序需要的线性连续的区域，再现程序是通过使用创造法或 SAVE_INSTRUCTION 或保存的 BUILTIN_INSTRUCITON，从 MasterDart230 或 Dart 输出的。有利地形成其它 Darts 的任何 Dart 700 通常仅包括 DartTools 200 输出的源代码或数据的线性连续寻址的部分，这些部分是实际在生成的 Dart 中的再现程序所需要的。例如，一旦作为加载过程的一部分运行 MasterDart 230 的静态构造代码，则当运行 MasterDart 230 时通常不需要构造代码占用生成的 Darts 中的空间。这是因为生成的 Darts 其自身通常不包括 MasterDart 230。对于通过执行 MasterDart 230 或其它 Dart 产生的 Dart 文件的 MainData 2102 部分中的数据来说也是如此。步骤 9 对环境结构或环境对象示例分配内存，以跟踪与被加载的 Dart 相关联的内存、访问权限、状态。生成唯一的 contextld 值以用作可以在相同的 DartEngine 600 示例上运行的 Dart 代码或其它 Darts 调用的全局引用参考值。环境标记将限制 Dart 代码对明确分配为特定的运行的 Dart 的数据、DartHeap，或 DartStack 的内存的访问。同时试图访问不是该 Dart 或其 DartEngine 环境一部分或者对其没有建立访问权限（如环境的标记值表示的）的函数、内存或文件，在一个实施例中这将产生来自 DartEngine 程序调用 4300 的负值的错误返回代码，这将使得在可能发生非法访问之前，结束 Dart 的执行。

[0699] 步骤 10 提取以 PartTableRecords 314 的参数和 RenditionsTable 的记录表示

的环境和初始内存要求。如果需要为任何引擎维护的 EventInfo 结构、FileInfo 结构、或 CommSessionInfo 结构分配和重新分配内存，或者存在将被加载的 Dart 的预期执行需要的其它资源，则进行分配、重新分配、初始化等，以准备在相同的 DartEngine 示例上运行的 DartContexts、DartProcedureContexts 和 DartIdleContexts 共享的 DartContexts 和 DartEngine 环境。

[0700] 步骤 11 优选地打开和读出子文件，例如 SubFile 698（见图 26），以处理或执行 DartFormat 300 图像文件的部分，如同它们是独立的文件，不需要将数据备份到独立的文件中。

[0701] DartPlatform50 有利地采用唯一的文件系统。执行该文件系统 612（参照图 26 所示），以产生连接到文件标识符（字段）值的文件抽象，它可以被用于引用 Dart 代码中的文件或当读出、传递、写入或共享数据时引用文件。

[0702] 几个 BuiltinInstruction 670（参照图 20 所示）操作代码值可以被用于读出、写入、打开、关闭、重命名，以及定位操作。以一对一的方式与每个字段相关联的 FileInfo 结构跟踪存储器的类型和形式、特性和访问权限、以及当前存储器位置、文件中当前指针位置等等。

[0703] 有利地，设备专用的 halBase 对象映射 DartEngine 600 执行过程的可移植部分与在 DartDevice 400 上操作所需要的小组函数，设备专用的 halBase 对象非常小并且简单，因为多数不与实际的独立物理存储的文件相关的操作被 DartEngine 中的可移植代码虚拟化，如图 26 所示。

[0704] 对于文件的处理，仅仅需要执行 HAL 的读出、写入、打开、关闭、定位、得到位置、重命名以及删除操作，以将 DartEngine 600 移植到新的 DartDevice 400 中。DartEngine 600 执行过程的可移植部分为保存在分配的内存 697（参照图 26 所示）中的文件，或者子文件或其它文件生成文件存取抽象。一旦子文件被打开就能象任何其它文件那样被读出，但是数据源和边界范围实际上是以前打开的另一个文件的适当的子集。

[0705] 有利地，即使源文件在子文件之前被关闭，但是子文件仍然保持运行。使用子文件读出用于执行的 Dart 的代码允许适当地执行代码，这就使得不需要将代码从文件加载到内存中。当 Dart 处于 ROM 中或者 DartDevice 的文件存储是以一些其它快速可读存储设备实现而不是以慢速的硬盘驱动等实现时，这种方法特别有利。

[0706] 如果希望提高执行速度，与将被运行的再现程序相关的 Dart 的代码部分，可以被加载到为保存和运行 DartContext 的代码分配的内存中，并且内存文件（图 26697）或其等效函数可以被用于执行实际的 DartCode。

[0707] 步骤 12 在这里描述的特别有利的实施例和执行过程中，DartTools200 为 MasterDart 230 或任何其它的 Dart 700（仍然可以包括初始构造器）生成的 Dart 代码，可以在初始构造器程序位置开始执行，在调用 main() 函数前调用该程序。开始执行的代码地址在 RenditionsTable 记录的一个 32 位字中，其中的 RenditionsTable 记录相应于将运行的再现程序示例。

[0708] 步骤 13DartTools 200 自动地将 BuiltinInstruction 659 放在初始构造器的最后，初始构造器具有子函数值，这使得一旦第一次调用 DartEngine.Process 方法 4003，则 DartEngine 设置起始执行地址和用于 MasterDart 230 指令的连续执行的对象示例指针。

[0709] 根据本发明方法和计算机程序的特定实施例，这是创建和加载 Dart 的最后步骤（步骤 13）。

[0710] Dart 和 Dart 程序的动态生成 - 创造法

[0711] 由 Darts 和其衍生的 Darts 连续地动态地生成 Darts 和 / 或 DartProcedures 的能力是基于 DartPlatform 50 和 Dart 技术的系统和方法的更有效的特性之一。在整个系统中设计和构造 DartPlatform 50 的实施例，以支持由以前的 Darts 连续地有效地生成 Darts。用于实现上述的方法被称为创造法。

[0712] 例如，通过使用 C++ 扩展，DartSource 100 可以包括在 Darts 中的再现程序的生成中被组合、匹配和共享的部分的说明，所述 Darts 是由初始 MasterDart 230 生成或直接地由其它 DartTools 生成的。

[0713] DartFramework 102 的实施例固有地提供了用于指定再现程序，和将被包括在每个再现程序中的代码、数据和部分的机制和方法。这就允许 DartTools 200 通过编译器 201 和链接器 202 生成的对象的依赖树进行解析，以形成动态地生成 Darts 和 DartProcedures 4000 所需要的代码、数据和资源的有效数据库。以类似的方式，MasterPlayer 203 能够通过在运行时间可到达的数据、代码、方法和功能跟踪任何开始点的组合，从而有利地形成需要的信息，以生成仅包括需要的数据、代码和资源的 Darts。根据包括的再现程序开始点和数据值，自动地从 DartFormat 图像中排除不可到达的数据、代码和内容。

[0714] DartPlatform 50 的其它机制可以被有利地用于限制需要的代码和数据量，当生成其它 Darts 700 或 DartProcedures 4000 时，自动地保存运行的 Dart 的全部或选择的数据状态，并作为上述的 DartEngine 的 Dart 加载步骤的一部分，自动地重新加载全部或选择的数据状态。因为运行的 Darts 的数据值和内容是 DartFormat 图像本身的一部分，因此它们在运行时易于被保存和存储，因此通常不需要生成用于最初构造和建立代码的 Darts，所说最初构造和建立代码通常构成许多类型的用户为中心的应用程序，其中对该应用程序优化和设计 DartPlatform 50。例如，当 MasterDart230 最初被加载时，在设置在 main() 函数开始执行前调用许多构造器。对于多数 C++ 生成的程序和本发明可以使用的其它不同的软件或编程语言生成的程序来说也是如此。有利地，MasterDart 230 可以利用 BuiltinInstruction 659 的嵌入函数，在 main() 函数中保存最小的指令集，以访问 DartEngine 的 Save()。由于默认地保存当前数据值的完整图像，并且当加载保存的 Dart 700 时进行恢复，因此不需要在保存的 Dart 700 中运行构造代码。因此，MasterDart 230 能够引导保存过程，从而执行已经运行的最初构造的连续线性范围的代码不包括在生成的 Dart 中的任何再现程序中。以这种方式，生成的 Dart 可能比生成它的 MasterDart 230 明显较小。它还可能比类似功能的传统的应用程序可执行图像小很多，其中的可执行图像保存所有需要的静态构造器和建立代码，以当无论何时开始应用程序时，重新建立数据状态。

[0715] 不需要包括初始构造代码正是用于优化生成的 Darts 的大小、复杂度，和计算要求的许多其它机制和方法之一。与传统的机制和方法相比，这些优化通常产生更小的大小和更低的复杂度和计算要求。

[0716] 当符合 DartFormat 300，在 MasterPlayer 203 上运行的 MasterDart 230 或任何 Dart 700 生成 Dart 或 Darts 集时，执行数据成员消除法。通过 MasterPlayer 203 的用户接口或在 MasterPlayer 203 上播放的 Dart 的执行，可以选择任何适当的再现程序对象子

集，以被包括在将被生成的 Dart 700 中。

[0717] 通常，仅将所选择的再现程序需要的代码和数据的一些部分放入生成的 Dart(s) 中。这是由 DartTools 200 和 MasterPlayer 203 实现的，为此要通过解析来自编译器 201 和链接器 202 生成的图元数据的对象的关系，并且通过跟踪运行时间数据成员、程序方法调用和实际的指针值，来发现所有数据成员、代码成员，和每个类的方法，然后动态地从对象示例排除并访问代码所有的引用参考值和数据成员、结构成员和方法成员的空间，而这些是不能通过运行从开始的 Process() 成员函数中选择的再现程序达到的。因此，一旦对于将被生成的 Dart(s)700 确定所有的开始位置，则根据运行在 MasterPlayer 203 上的 Dart 的实际运行时间设置，有效地优化所有类信息。同时有利地，从再现程序的起始方法不能到达的所有设置代码不包括在生成的 Darts 中。

[0718] 通常以用户为中心的应用程序的重要部分，例如主要为其设计 DartPlatform 的部分，包括用于为用户接口图像设置初始屏幕位置、初始化数据结构、构建连接对象的图形、生成表等的代码和数据。DartFramework 102 在多数类定义中包括 Setup() 方法，Setup() 方法将仅被 MasterRendition 或来自其它 Setup() 调用的嵌套调用所调用。有利地，如果 MasterRendition 不被选择为包括在生成的 Dart(s)700 中，则 Setup() 方法和其它方法和函数，以及不会另外地从所选择的再现程序达到的所有类的数据成员或静态数据元素，都不会被包括在生成的 Dart(s)700 中。有利地，进行优化的 Setup() 调用不是特殊的，因为不能从所选择的运行时间起始点到达的任何代码、数据、方法等不会被 DartTools 200 包括在生成的 Dart(s)700 中。

[0719] 收集资源程序

[0720] 收集资源程序或方法是将被用在 DartFramework 102 中以消除生成的 Dart(s)700 中的不必要的代码和数据的另一个程序。来自对该 MasterRendition 方法的调用的 Setup() 方法或调用树可以调用与 GatherResouces() 相关的方法和函数的体系结构，以当 MasterDart 230 在 MasterPlayer 203 上运行时，提供用户接口或动态收集数据、内容或其它资源。

[0721] 还应该注意的是，结合这里的描述，DartTools 200 可以在 MasterDart230 中包括许多部分，以保存符号表、类定义、MasterPlayer 203 执行刚刚描述的优化所需要的其它图元数据。如果这些部分没有被包括在任何所选择的再现程序中，则它们将有利地不会被包括在生成的 Dart(s)700 中。

[0722] 一些选择的 MasterDarts 的其它用途

[0723] 对于 MasterDarts 230，包括大小和复杂度比将被生成的 Dart 应用程序 700 大得多的 DartPlatform 50 代码和数据是通常的（但不是必须的）。这种额外的代码用于执行存取、请求、初始化、用户接口、表的预计算、组合各个部分，等等。因此 MasterDart(s)230 一旦被编译器 201 编译并被链接器 202 链接，就能被程序员或非编程人员使用和重新使用，以利用新的参数、资源、或可能随时间变化或根据其它情况（例如库存报价）变化的其它数据生成定制的 Dart(s)700，以在被链接器 202 链接后，通过在 MasterPlayer 203 上运行 MasterDart 230 或者自动地或者应用户的请求，生成非常宽变化范围的 Dart 应用程序。

[0724] 动态地重新配置、生成、组合，以及优化各部分

[0725] 现在应该知道 MasterDarts 230 和由其它 Darts 生成的 Darts 能够动态地重新配

置、生成、组合，以及优化各部分的选择和各部分中的部分，以生成适于新的功能和目标传输和环境的继承的子女 Darts。

[0726] 根据 contextType 参数加载各部分

[0727] 一旦已经使用 RenditionsTable 加载 Dart 230 的再现程序，其中再现程序引用 RenditionsTables，其中对于每一个再现程序每个 RenditionsTable 引用被包括的部分，根据 contextType 参数加载需要被加载的所有部分。DartInstructionSet 的 BuiltinInstruction 实现的 DartInstructionSet 和扩展有利于由任何 Dart 700 或 Dart 程序 4000 有效地重新配置和生成高度优化的 Darts 700 或 DartProcedures 4000。在一个优选执行过程中，仅对于那些在加载中具有要求的 contentType 专用的处理的部分，设置 PartTableEntry 记录的标记参数中的标记位。

[0728] MasterDart 通常可能包括用于所有代码的完整的代码图像，其中所有的代码可以在开始 MainCode 部分中的 MasterRendition 对象示例时得到。类似地，MasterDart 通常可能包括用于所有数据的完整的数据图像，其中所有的数据可以在开始 MasterRendition 对象示例时得到。这些中的许多将被从生成的 DartFormat 图像中删除。

[0729] DartFramwork、DartRuntime，以及 DartEngine

[0730] 现在将更完整地描述 DartFramwork 102、DartRuntime 8000，以及 DartEngine 600。尽管应该知道使用多数任何运行时间能够成功地构建和运行大多数任何的 C++ 程序，其中的 C++ 程序可能执行，或者在任何标准的 C++ 工具集上实现，但是 DartFramwork 有效地利用被构造为 DartEngine 的功能，某种 DartRuntime 操作。DartFramwork 的一个主要的能力主要是来自于其紧密的集成度和使用作为 DartEngine 的一部分实现的指令、机制、程序和函数。例如，由压缩的 JPEG 文件或部分解压缩和构建位图图像，可以通过编译来自 JPEG 标准例子（在互联网上可以得到）的 C++ 源代码实现，但是这种代码的执行可能比嵌入函数的简单使用慢得多，其中的嵌入函数采用 BuiltInInstruction（在该文章的其它地方也被称为 BUILTIN_INSTRUCTION）来调用 DartEngine 方法，以当为 DartDevice 的实际处理器编译 DartEngine 源代码时，利用生成的本地代码执行 JPEG 解压缩操作。

[0731] 类似地，DartFramwork 102 在 DartPlatform 50 的整个执行中利用嵌入函数的生态系统或环境，例如，在一些实施例中包括但不限于：实际的 DartInstructionSet、DartRuntime 8000 系统、通信子系统、事件处理系统、文件系统（参考图 26 中的文件系统 612）、图形和输入系统、电源管理系统、Dart 组合压缩和加密机制，以及解压缩、存取、保护、数字权限管理和生成机制。

[0732] 应该知道，使以用户为中心的函数的很好定义的广集和集合是用被包括在任何或每个设备中的为本地处理器生成的代码表示的，这样的集合是有利的，并且在一些实施例中，是生成能够将其自身扩展到其它设备上的可移植的应用程序所需要的函数和适配的关键。

[0733] 特别地，应该知道，尽管在理论上，任何图灵机兼容的指令集能够被用于生成任何其它代码或实现任何算法，但是实际上该方法永远不会如提供相对完整的平台的紧密集成的平台那样进行执行、存储、提供能量，或成本有效，并且以平台专用的方式执行必须的计算或存储饥饿功能，如同 DartInstructionSet 的 DartEngine 处理。

[0734] 通常，为了将设备看作 Dart 设备 (DartDevice)400，应该在该设备上安装 Dart 播

放器，例如DartPlayer 500，并且在授权的设备上应该支持至少一个通信机制。DartPlayer 500是设备专用的可执行单元，用于提供启动、初始化、给予处理器控制并关闭DartEngine 600所需要的环境和可选的用户接口。DartEngine 600被构建为两个互相连接的类（便携式playerBase类和设备专用的halBase类）的示例。当DartEngine 600被移植到新的设备上时，必须构建设备专用的DartPlayer 500应用程序，以封装DartEngine的执行，并且必须构建设备专用的halBase导出类，以提供从独立于DartEngine部分的平台到实际的设备操作系统和/或硬件的桥梁。

[0735] 在图23中表示DartPlayer 500主循环流程。首先调用DartEngine.Init()6000函数，如果存在参数的话，则加载识别Dart 700的参数。接着在循环中调用DartEngine.Process()（图234002、图246000），直到返回负值，使得调用DartPlayer.Uninit()，并且关闭DartPlayer 500应用程序。正的返回值使得在返回主循环之前，执行值专用的DartPlayer处理（参照图23 4000所示）。

[0736] 这种专用的DartPlayer处理包括释放对可能在设备上运行的其它应用程序或处理的控制，根据电源管理值中止PlayerEngine线程，收集如鼠标移动和键盘按压的输入并将其转换为对将事件800添加到DartEngine的EventQueue的调用（参照图15 8002和图16 5000所示）。如果正的返回代码值是被保留为表示Dart完成处理的值之一，则将调用DartProcess.Uninit()并且中断Dart处理。例如，返回DONE正的返回代码值是通常的方法，其中运行的Dart将以信号表示其自己的结束。

[0737] 在图24中表示DartEngine.Init()的处理，用于调用包括引用被加载的Dart 700的DartEngine.Init()。当环境不被激活并且没有被加载的Dart 700时，在调用DartEngine 600的情况下，DartEngine内部的IdleProcedure将被加载到IdleContext中，而不是Dart 700被加载。在IdleContext中运行的IdleProcedure由DartInstructionSet指令的循环构成，这些指令执行保持DartEngineEvents（图234003）处理有效所需要的处理。在一个优选执行过程中，IdleProcedure简单地执行实现对EngineEventProcessing函数的调用的指令（图12和图15 8002）。在一个优选执行过程中，DartPlayer 500的一个示例通常能够以多个方式中的任何一个，处理任何数量的执行的Darts 700。一种方式是通过改变主循环400，在相同的运行的本地处理器线程中调用单独的循环，或者通过生成任何数量的独立的DartEngine 600示例，或者通过使用独立的线程，或上述的任何组合。一种可选的方案是Dart被明确地加载，并作为subDart或运行的双亲Dart的子女Dart运行。这里subDart一旦作为被编译和被链接的双亲Dart的一部分被加载，则通过事件处理单元的事件处理将被传输到subDart的事件处理单元。在优选的执行过程中，这些事件处理单元被称为Gizmos，并且是DartFramework类Gizmo的示例。

[0738] Gizmos

[0739] Darts 700自身也能够以串行的，或并行的，或高度协作的方式，在其它Darts 700上运行，其中Darts作为其它Darts的处理体系结构的一部分运行。DartFramework 102主要由Gizmo 115导出的对象组成。根据子女Gizmo和双亲Gizmo指针的引用，以严格定义的线性顺序有利地配置这些Gizmos（参照图18 10000所示）。该顺序用于将Gizmos集中到互相连接的树状图形中，所述树状图形确定执行顺序和在访问权限和屏幕视野、时间感，以及其它环境特征方面的关系。例如，双亲Gizmo能够为子女Gizmos和其派生Gizmos改

变时间和表观时间变化速率，以控制其处理和速度。在图 18 的例子中，通过严格地坚持特定的处理顺序，所有这些都变得更加易于以稳定的方式实现。

[0740] 图 18 表示 Gizmo 115 导出的对象的体系结构的例子，其中 Gizmo 115 导出的对象用于运行幻灯片放映 Dart 700 应用程序。重要的在于明白幻灯片放映 Dart 700 应用程序利用 DartEngine 600 实现组成幻灯片放映 Dart 的代码的 DartInstructionSet 指令。特别地，注意到使用面向 DartDevice400 的本地处理器和操作系统的标准的构建工具编译、链接，加载和运行 DartPlayer 500 和 DartEngine 600。正是 DartEngine 600 执行由 DartInstructionSet 的指令构成的幻灯片放映 Dart 700 的代码，其中 DartInstructionSet 由 DartTools 200 生成。作为调用 DartPlayer.Process() 的调用的结果，当基于 DartInstructionSet 的 Menu.Process() 方法开始在引擎上执行时，首先控制图 18 所示的体系结构顶层的菜单 Gizmo，其中 DartPlayer.Process() 的调用使得 DartEngine 600 开始处理 Dart 700 的指令。注意到在优选的执行过程中，体系结构顶层的 Gizmo 是从图 11 中的基 Gizmo 类继承的再现程序对象类。

[0741] 在图 18 的例子中，每个圆角框表示 Gizmo 导出的对象，用于处理幻灯片放映应用程序的用户接口单元。用于选择不同功能的处理用户接口的菜单包括添加幻灯片、开始幻灯片的自动排序、选择幻灯片过渡效果，等等。可以从菜单中选择的一个选项是观看幻灯片放映。在该例子中观看可以是例如包括超链接索引观看，其中用户能够点击幻灯片的名称和描述；可以是缩图索引观看，其中用户能够点击幻灯片的小的缩图；可以是幻灯片放映观看，其中幻灯片占用大部分屏幕，并且在屏幕上箭头、物理按钮或鼠标或画笔被用于向前和向后移动通过幻灯片的结构。菜单仅有一个子女 Gizmo 导出的对象，但是当用户从菜单中选择视图时，指向子女 Gizmo 导出的对象的指针被变换到支持所选择的视图的 Gizmo。因此，有利地，对用户如何观看和导航幻灯片进行改变所需要的操作是改变菜单 Gizmo 中的一个指针。

[0742] 在该例子中，每个视图具有指向三个子女 Gizmo 115 导出的对象的指针的列表，在图 18 中被表示在它们下面的一行中。子女 Gizmo 都是简单的静止图片或幻灯片，以三个视图的每个子女指针列表中指针的从左到右的顺序表示。Gizmo 导出的对象的子女指针列表被标记为“3”，以从左到右的顺序指向三个子女 Gizmo。被标记为“4”的对象表示 Gizmo115 导出的对象，Gizmo 115 导出的对象封装了独立生成的运行的 subDart，作为一个幻灯片包括在幻灯片放映中，但是实际上能够包括其自己的幻灯片体系结构，其中每个幻灯片自身能够包括任何图片、视图或 Darts 700 和所有功能。被标记为“7”和“9”的框包括 Gizmo 115 导出的对象，这个 Gizmo 115 导出的对象显示实时回放的视频 / 音频幻灯片。应该注意到，对于该幻灯片，所有 Gizmo 115 导出的对象进行的事件处理和被包括的 Dart 回放 Gizmos 115 的 Darts 700 以指向子女和双亲的指针产生的图形的正确的顺序接收处理控制。在图 18 的框内表示的数字值的顺序中明确地表示了该顺序。

[0743] 由于几乎 Gizmo 115 导出的对象的所有的处理是通过对其 Process() 方法的调用，指针指向作为参数进行处理的事件，因此任何双亲类可以生成或改变其子女类可见的事件的参数或类型，或者能够决定是否将被处理的事件传递到子女类。因此实际上，任何双亲类能够为其所有子女类设置环境。例如，包括 Gizmo 对象的 Dart 不需要知道它仅被给予其双亲类可用屏幕的一部分。类似地，当 Gizmo 115 导出的对象访问 GetTime() 方法时，如

果双亲类表示它是子类的时间源，则它将自动地从其双亲类请求时间，而不是从嵌入的对引擎的调用中请求。双亲类能够控制其子女类的运行时间环境，这是 DartFramework 200 和 DartRuntime 8000 的非常有效的特征。这就使得易于构造例如，幻灯片放映的放映，收集刚刚收集 Darts 的 Darts，允许用户选择将运行的 Darts，能够从打印机或其它 Dart 激活的设备得到 Dart 700 形式的控制面板，并在嵌入提取的 Dart 的方形中运行。

[0744] DartEngine 体系结构 - 传递控制

[0745] DartEngine 600 的体系结构有助于将控制从 Dart 容器 Gizmo 115 导出的对象传递到包含的 Dart 体系结构的最顶层 Gizmo 115，这是通过包括 BUILTIN_INSTRUCTION 函数以允许包含的 Gizmo 115 导出的对象为包含的 Dart 700 创建 DartContext，使 Dart 700 被加载到该环境中，并接着通过 DartEngine 的 BUILTIN_INSTRUCTION 调用体系结构的最顶层 Gizmo 115 的包含的 Dart 的 Process()，来调用包含的 Dart 700 的 Process() 实现的。

[0746] 与调用一起，管理和传递处理到包含的 Darts 700，DartEngine 600 也使得从容器 Dart 700 的代码到被包含的 Dart 的代码的转换更有效，这是通过允许包括指向事件 800 的指针的 EventProcessing 结构示例被包含的 Dart 处理，并直接通过使用指向有效地在所有 DartContexts 之间共享的内存的特定指针（参照图 32 指针类型 11），处于可存取的内存中。这就允许在 Darts 700 之间有效地传递事件 800 和其它参数，其中每个 Dart 700 具有另外地分离的地址空间。

[0747] 当从一个 Dart 700 调用另一个 Dart 700 时，DartEngine 记忆调用 Dart 的 DartContext，并接着开始在被调用的 Dart 的 DartContext 外进行操作，直到当引擎堆栈处于与最初调用相同的位置时，调用的 Dart 的 DartContext 执行 DoneInsturction 为止。因此 DartEngine 600 使得 Darts 700 能够容易地和有效地运行运行 Darts 700 的 Darts 700（作为 Gizmo 115 导出的体系结构的部分），如同它们作为一个 Dart 700 由 DartTools 生成。包含 Darts 能够提供其想为被包含的 Darts 700 提供的任何环境，正如同其能够为子女 Gizmo 115 导出的对象提供一样。

[0748] 可能假设，将控制简单地向下传递通过整个体系结构将浪费调用 Gizmo 115 导出的对象的处理器时间，该导出的对象不需要进行处理，例如表示静止图片的 Gizmo 115 导出的对象不需要被重新显示。在优选的实施例中，用于删除 Gizmo 115 导出的对象调用的树的系统可能会对这种问题产生不良影响。每个 EventType 801 的值是由正好设置一位的类型标记字段和将被属于该类型处理的用于特定的事件 800 的事件子类型字段组成。在一个实施例中，存在相应于 EventType 801 的类型标记的两组标记。如果 Gizmo 导出的对象本身需要控制该类型的事件，则设置一个标记，如果其任何继承对象需要看到事件 801 的特定类型则设置另一个标记。在一个实施例中，在每次通过体系结构时，Gizmo 115 导出的对象为其自身设置标记，并收集其调用的所有子女对象设置的标记的逻辑 OR 的集合。因此根据将被处理的事件 800 的类型，Gizmo 总是知道其子女 Gizmo 是否需要被调用或不被调用。EventsTypes 的集合定义类型以有效地删除不必要的调用。例如，事件 800 需要处理用户的输入，不需要向下传递通过 Gizmo 115 导出的对象的树，该树不包括需要处理用户的输入的任何 Gizmo 115 导出的对象。DartEngine.Process() 处理

[0749] 在图 23、图 15 和图 16 的实施例中表示 DartEngine.Process() 处理的例子。这就驱动了 DartDartInstructionSet 指令 611 的主要执行。执行主循环以从 Dart 指令流

得到下一条指令，解码指令的操作码和源字段和目的字段，检查被寻址的源参数和目的参数没有处于 Dart 的 DartContext 的数据区域之外，接着跳到执行指令的操作字段调用的函数的 DartPlayer600 方法和函数的设备处理器本地代码。如果指令的源字段或目的字段指定处于被允许存取范围之外的参数，则根据操作码不进行调度，并且正好在 DartPlayer.Process() 调用后返回控制，特定的负返回值表示特定的违反规定的存取。这就使得 DartPlayer 500 调用 DartPlayer.Uninit() 并以错误信息或其它可选的指示结束 Dart 的执行。

[0750] 图 22 表示 DartPlayer 500、DartEngine 600 和硬件抽象层 (HAL)650 的关系和其一些功能。HAL 650 是播放器的一部分，它被实现为从 halBase 类导出的类，用于封装所需要的设备专用存取函数。有利地，将被要求执行的 halBase 函数的数量和复杂度降低到最小，从而能够快速和容易地移植到新的 DartDevice 400。halBase 设计的最低要求方法还有助于保证分别执行的 DartEngine 600 能够获得高度的兼容性，因为它们共享了多数执行源代码函数。因此将尽可能多的函数移动到 DartPlayer 500 的可移植设备的独立部分是非常有益的。便于移植以及高度的兼容性是 DartPlatform 50 实施例的重要特征，通过在任何可能的时候使用可移植代码和在每个设备上使用相同的源代码，可以部分地实现上述特征。因此尽管对每个不同类型的设备来说，通常需要重新编译和链接，但是大多数函数将从与其它设备相同的源代码编译，希望其它设备能够与设备协作地工作。

[0751] 体系结构的电源管理特征

[0752] 电源管理尽管是可选的，但是它是在 DartPlatform 50 体系结构的许多部分中运行的另一个特征。当 DartPlayer 500 调用 DartEngine.Process4003 方法时，通过将保存需要的毫秒单位（或任何其它单位）的响应时间的变量设置为可以表示的最大值，来开始 Gizmo 导出的对象体系结构的执行。由于事件是通过排序的 Gizmo 115 导出的对象的树处理的，因此每个对象采用 BUILTIN_INSTRUCTION 来告诉 DartEngine600 直到它需要再次调用 Process() 方法，还需要等待多长时间。

[0753] 例如，对于显示运动视频的对象，如果是视频的帧速率的话，可能是三十分之一秒。DartEngine 600 收集 BUILTIN_INSTRUCTION 报告的最低响应时间要求，BUILTIN_INSTRUCTION 实现收集作为参数传递的最小响应时间要求。在完整地通过 Gizmo 树形的体系结构后，当控制返回到 DartPlayer 500 时，DartPlayer 500 或 DartEngine 600 能够利用该信息阻挡其线程或者另外地用信号表示至少在所示的时间内不需要电源。通常 DartPlayer 500 将利用等于收集的最小响应时间的超时来阻挡其线程。如果新的输入表示需要被 Dart 700 处理，则该阻挡也将结束。应该知道在许多情况下，运行在处理器上的应用程序是在所有时间都知道真正的响应时间要求的唯一实体，可以以这种方式，即易于使 DartPlayer 500 接收该信息并利用该信息极大地降低设备的电源和能量消耗要求，而不会极大地降级动态应用程序的性能，来有利地设计和生成 Dart 应用程序。

[0754] 事件处理排队机制

[0755] 在 EngineEventProcessing（图 158002、图 165000）处理期间执行的事件处理排队机制也可以跟踪或监视异步处理（图 16 虚线框的内容）所需要的响应时间，其中的异步处理不是 Gizmo.Process(EventInfo*) 调用体系结构的一部分。这样，有效的 Darts 报告的最小的响应时间要求反映了 Gizmo 树的线性任务分配执行的主流同步的 Dart 应用程序处

理，和 DartEngine 600 的事件处理部分控制的异步处理。

[0756] 安全性和完整性 - 在指令解码过程中执行检查

[0757] 通过使用在解码 DartInstructionSet 的指令时执行的检查，至少部分地保证 DartPlatform 50 的安全性、完整性和稳定性。数据和指令的源字段的地址被引擎检测，以保证它们指向正在执行的 Darts 范围内的数据，根据所运行的 DartContext，所述范围是有效的可存取的数据区域。类似地，Dart 应用程序和组成应用程序的 DartInstructionSet 指令限于仅存取属于运行的 Dart 的物理存储器。这是为了防止恶意的或无效的 Dart 有能力存取私人数据或不应被运行的 Dart 存取的存储资源。

[0758] 硬件抽象层 (HAL) 文件系统命名规定保证 Dart 应用程序不能指定其合法范围之外的文件（但是可看到可选的研究过的例外）。当为例如打开或删除的操作指定文件时，Dart 应用程序不能指定完整的文件路径，而仅是 locationId 和数字，或者 locationId 和终端名称字符串，如图 28 5010 所示。这样，在 HAL 中可存取的物理存储器限于图 28 5020 中表示的位置。仅根据指定文件的这些参数，由 HAL 决定执行打开、关闭、读出、写入、定位、重命名以及得到位置操作。在 HAL 执行的沙箱外不进行存取的一个有意的但是可选的例外是在 HAL 中可以可选地支持相应于 USER_SPECIFIED_LOCATION 的 locationId，以提供对 Darts 环境限制之外的文件的存取。由 HAL 决定何时打开、删除或重命名文件，以明确地提示用户选择文件。虽然仅有用户的参与所给予不明确的同意，但是这允许应用程序输入和输出文件。

[0759] 尽管已经参考一些特定实施例描述了本发明的结构、方法、装置、计算机程序，以及计算机程序产品，但是该描述用于解释本发明的，而不应看作是对本发明的限制。所属领域技术人员可以进行各种修改，而不脱离随附的权利要求限定的本发明的真正精神和范围。这里引用的参考文献被包含在此以作参考。

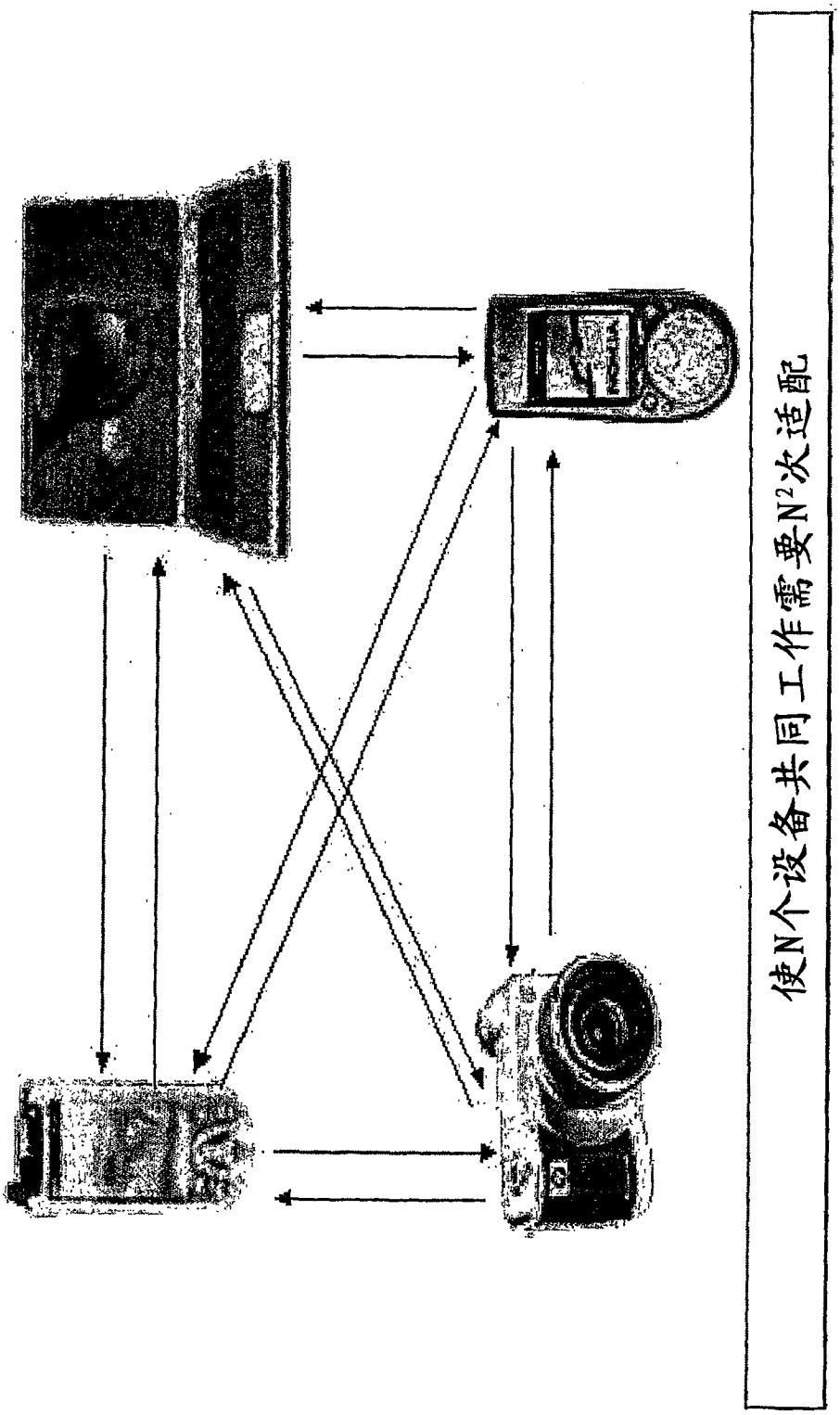


图 1

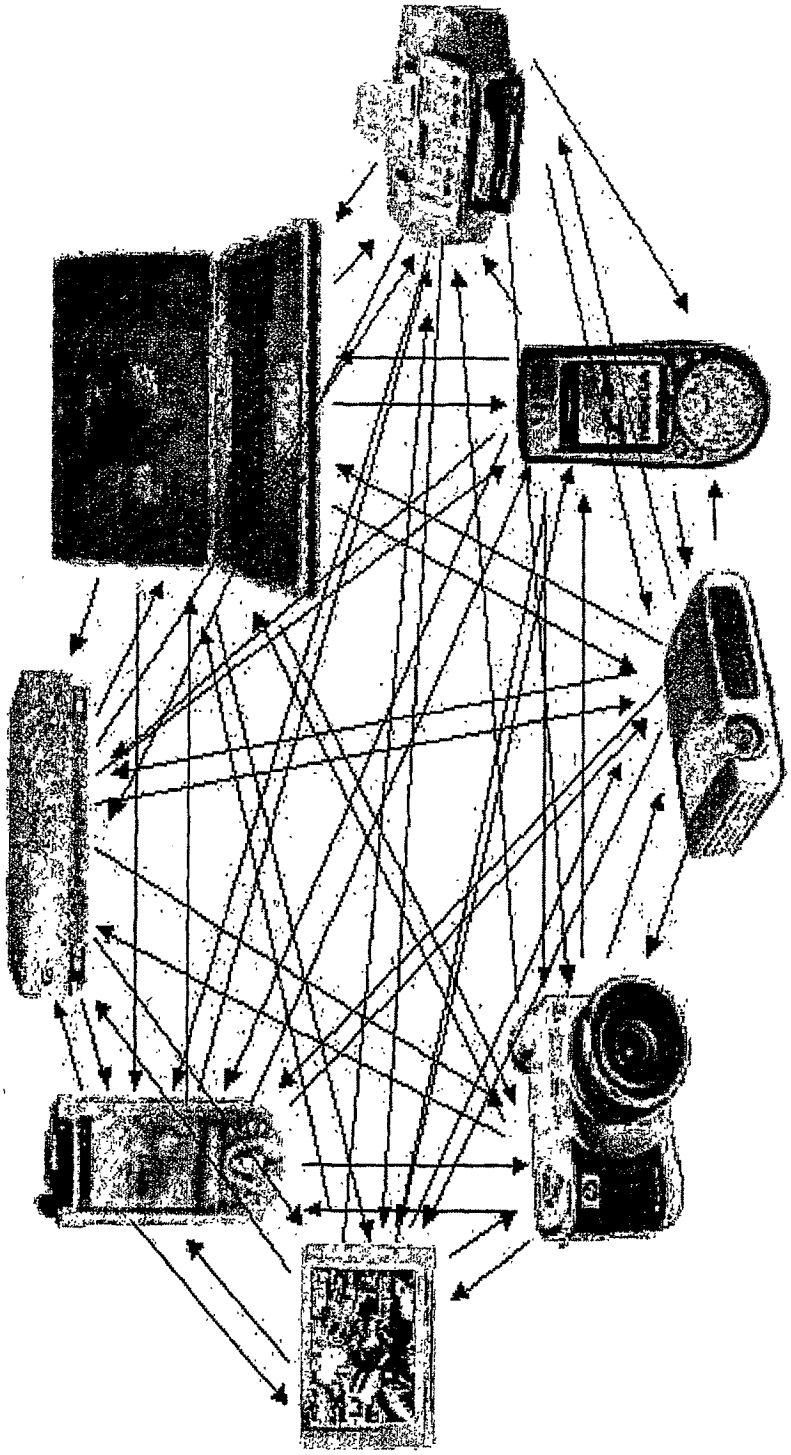


图 2

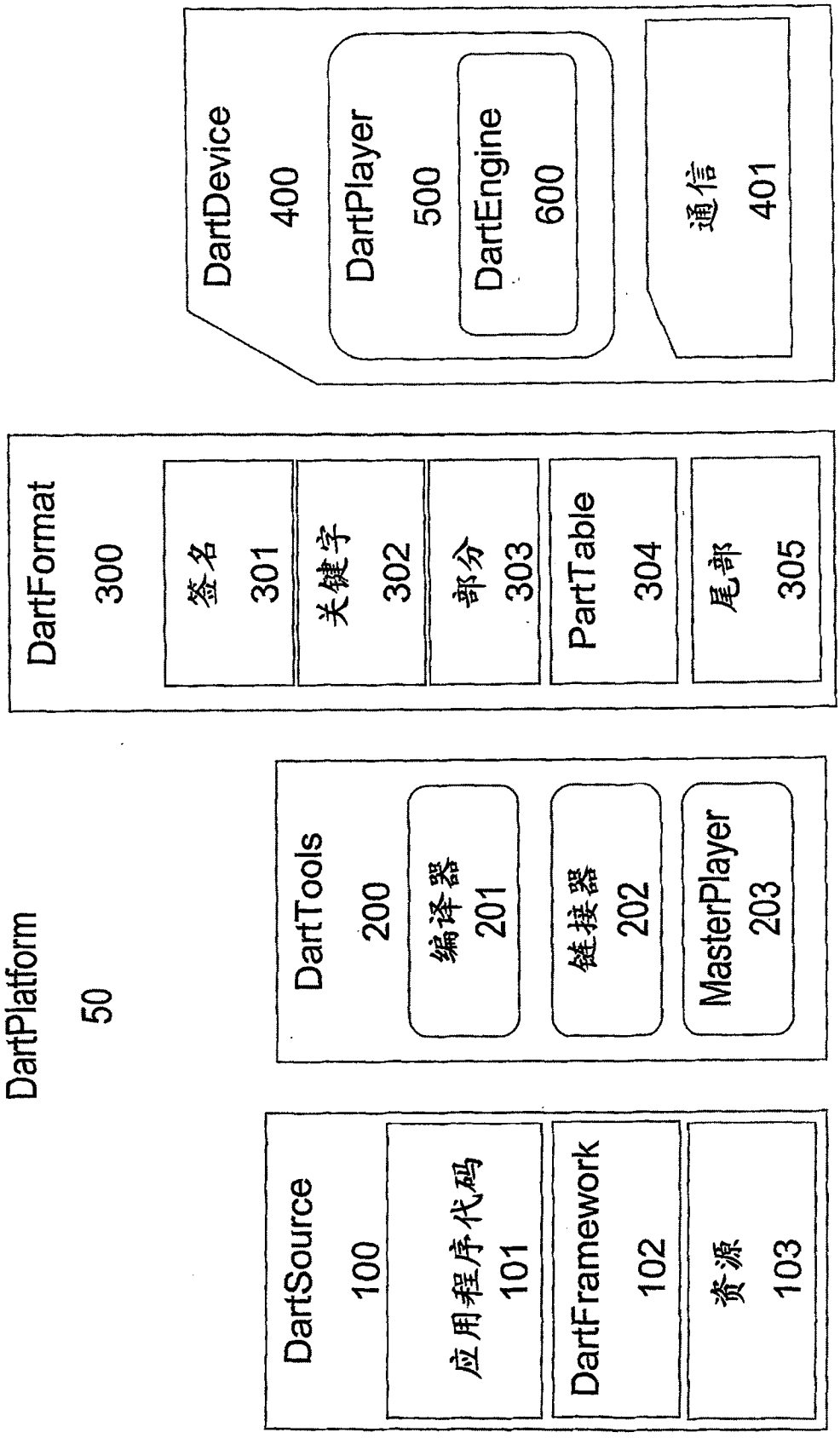

DartPlatform
50
DartSource
100
应用程序代码
101
DartFramework
102
资源
103
DartTools
200
编译器
201
链接器
202
MasterPlayer
203
DartFormat
300
签名
301
关键字
302
部分
303
PartTable
304
尾部
305
DartDevice
400
DartPlayer
500
DartEngine
600
通信
401

图 3

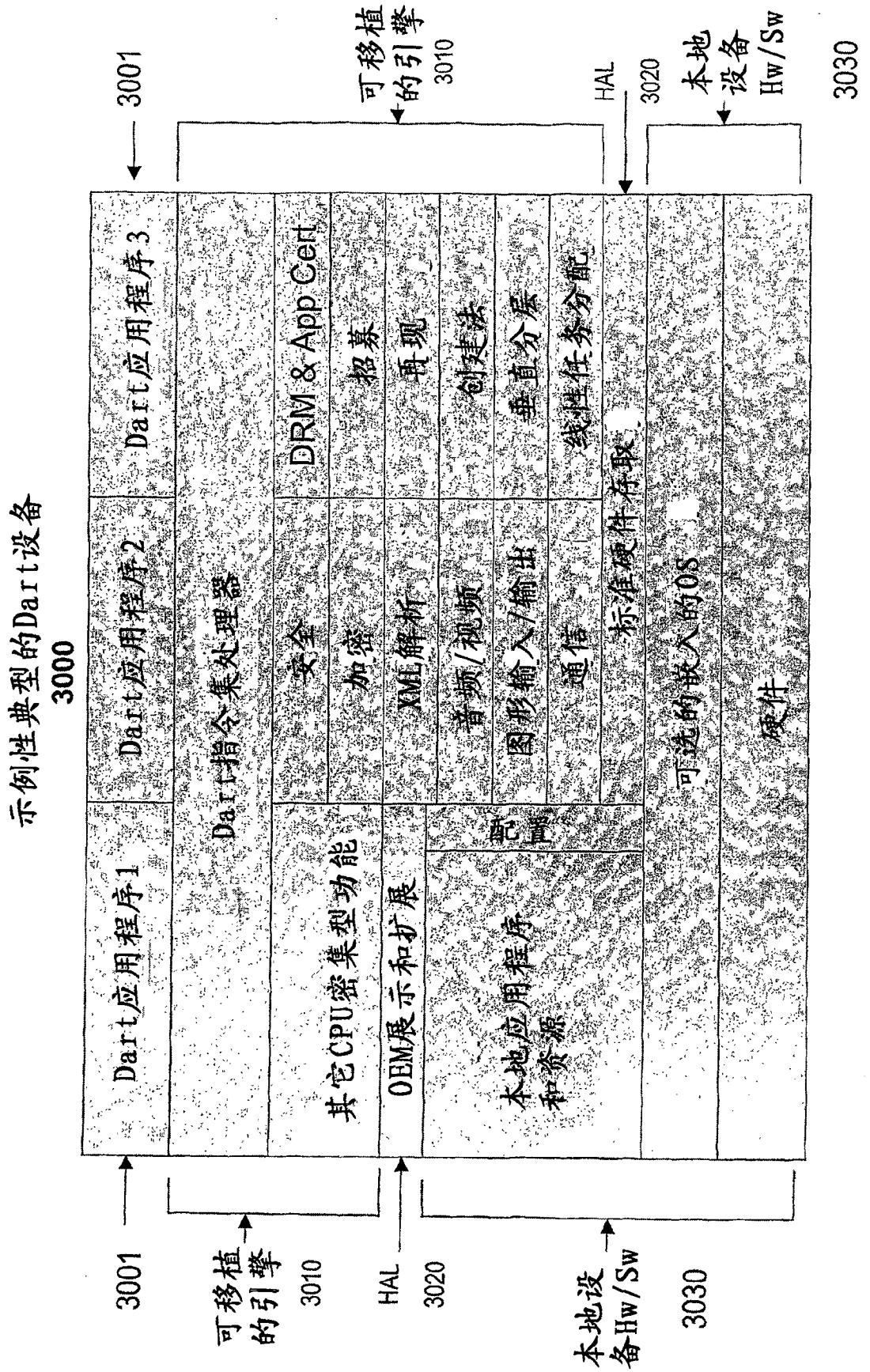

示例性典型的Dart设备
3000
3001
可移植的引擎
3010
HAL
3020
本地设备Hw/Sw
3030
Dart应用程序1
Dart应用程序2
Dart应用程序3
Dart指令集处理器
其它CPU密集型功能
OEM展示和扩展
本地应用程序和资源
安全
加密
XML解析
通信
DRM & App Cert
招募
再现
创建法
音频/视频
图形输入/输出
垂直分层
线性任务分配
标准硬件存取
可选的嵌入的OS
硬件

图 4

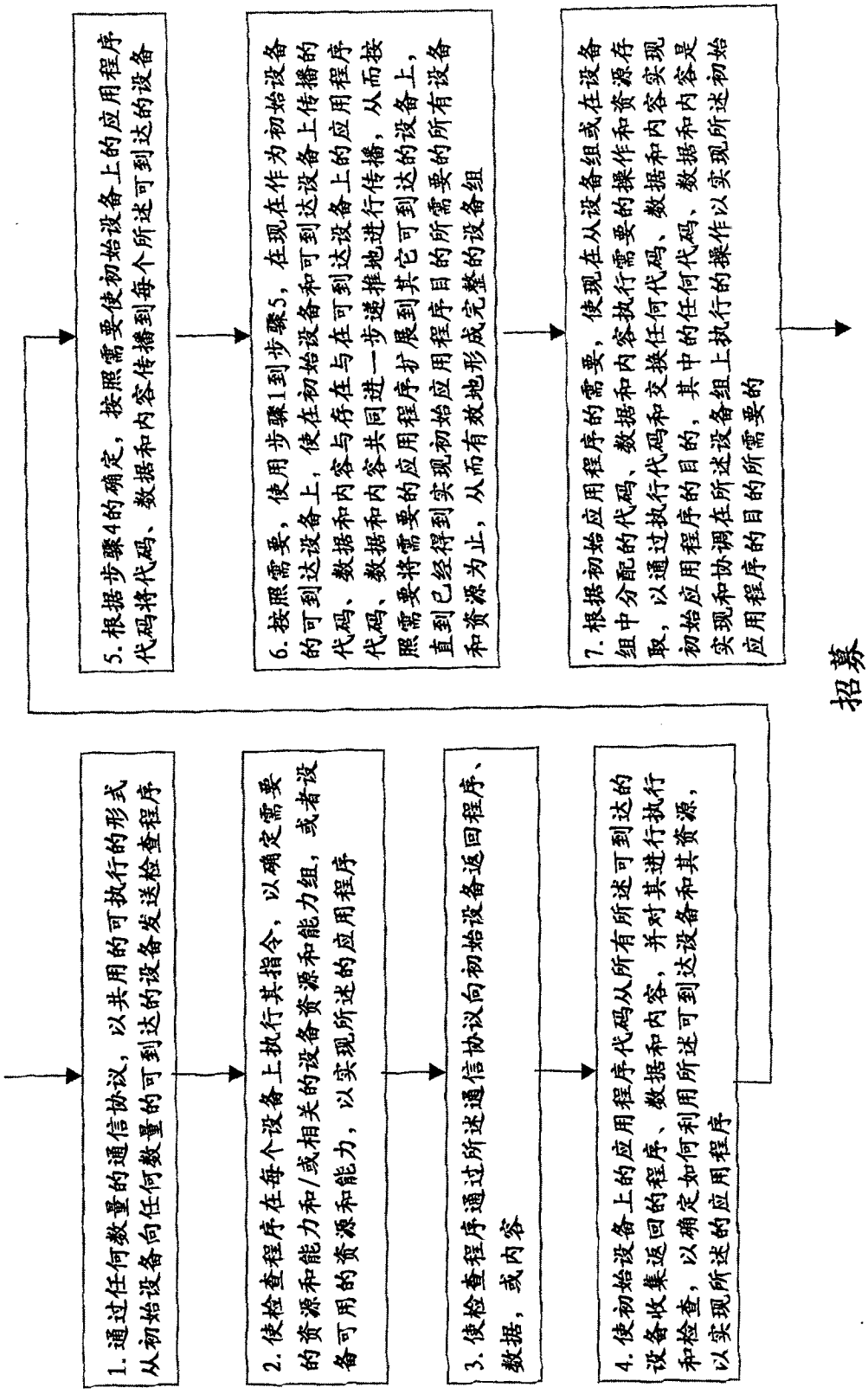

1. 通过任何数量的通信协议，以共用的可执行的形式从初始设备向任何数量的可到达的设备发送检查程序
2. 使检查程序在每个设备上执行其指令，以确定需要的资源和能力和/或相关的设备资源和能力组，或者设备可用的资源和能力，以实现所述的应用程序
3. 使检查程序通过所述通信协议向初始设备返回程序、数据，或内容
4. 使初始设备上的应用程序代码从所有所述可到达的设备收集返回的程序、数据和内容，并对其进行执行和检查，以确定如何利用所述可到达设备和其资源，以实现所述的应用程序
5. 根据步骤4的确定，按照需要使初始设备上的应用程序代码将代码、数据和内容传播到每个所述可到达的设备
6. 按照需要，使用步骤1到步骤5，在现在作为初始设备的可到达设备上，使在初始设备和可到达设备上传播的代码、数据和内容与存在与在可到达设备上的应用程序代码、数据和内容共同进一步递推地进行传播，从而按照需要将需要的应用程序扩展到其它可到达的设备上，直到已经得到实现初始应用程序目的所需要的所有设备和资源为止，从而有效地形成完整的设备组
7. 根据初始应用程序的需要，使现在从设备组或在设备组中分配的代码、数据和内容执行需要的操作和资源存取，以通过执行代码和交换任何代码、数据和内容实现初始应用程序的目的，其中的任何代码、数据和内容是实现和协调在所述设备组上执行的操作以实现所述初始应用程序的目的所需要的
招募

图 5

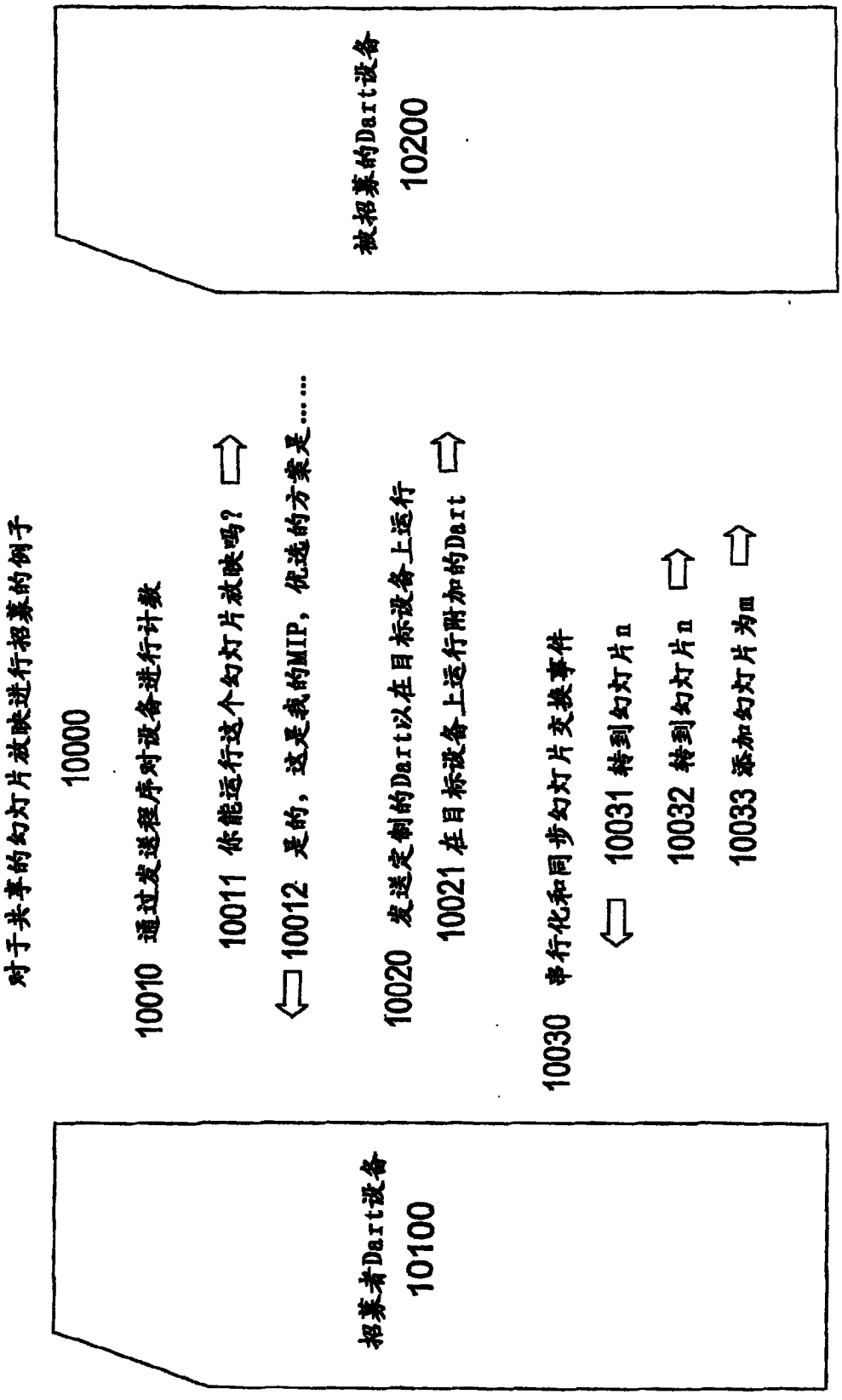

招募者Dart设备
10100
被招募的Dart设备
10200
对于未来的幻灯片放映进行招募的例子
10000
10010 通过发送程序对设备进行计数
10011 你能运行这个幻灯片放映吗?
10012 是的，这是我的MIP，优选的方案是……
10020 发送定制的Dart以在目标设备上运行
10021 在目标设备上运行附加的Dart
10030 串行化和同步幻灯片交换事件
10031 转到幻灯片n
10032 转到幻灯片n
10033 添加幻灯片为m

图 6

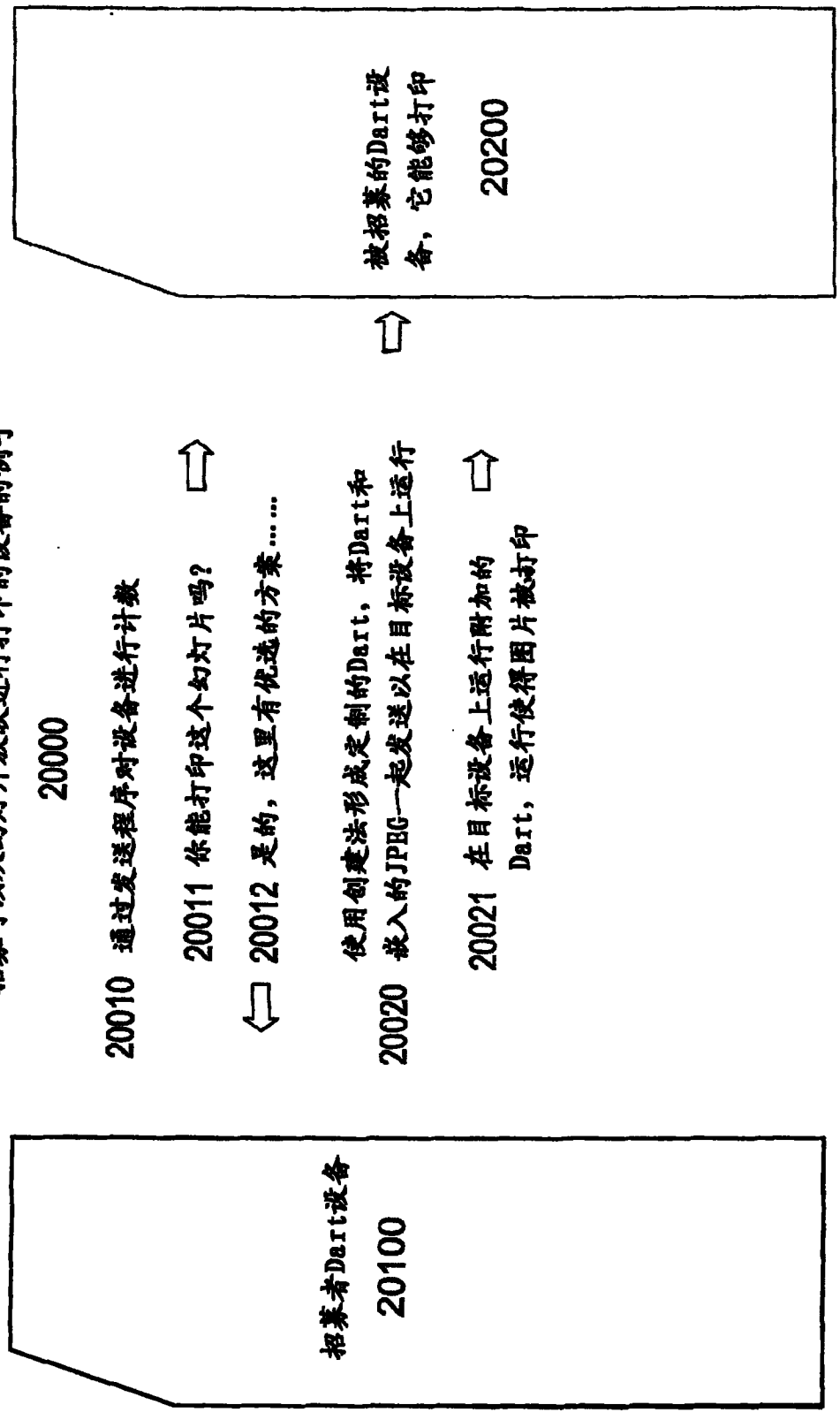

招募可以从幻灯片放映进行打印的设备的例子
20000
20010 通过发送程序对设备进行计数
20011 你能打印这个幻灯片吗?
20012 是的，这里有优选的方案……
使用创建法形成定制的Dart，将Dart和
20020 嵌入的JPEG一起发送以在目标设备上运行
20021 在目标设备上运行附加的
Dart，运行使得图片被打印
招募者Dart设备
20100
被招募的Dart设备，它能够打印
20200

图 7

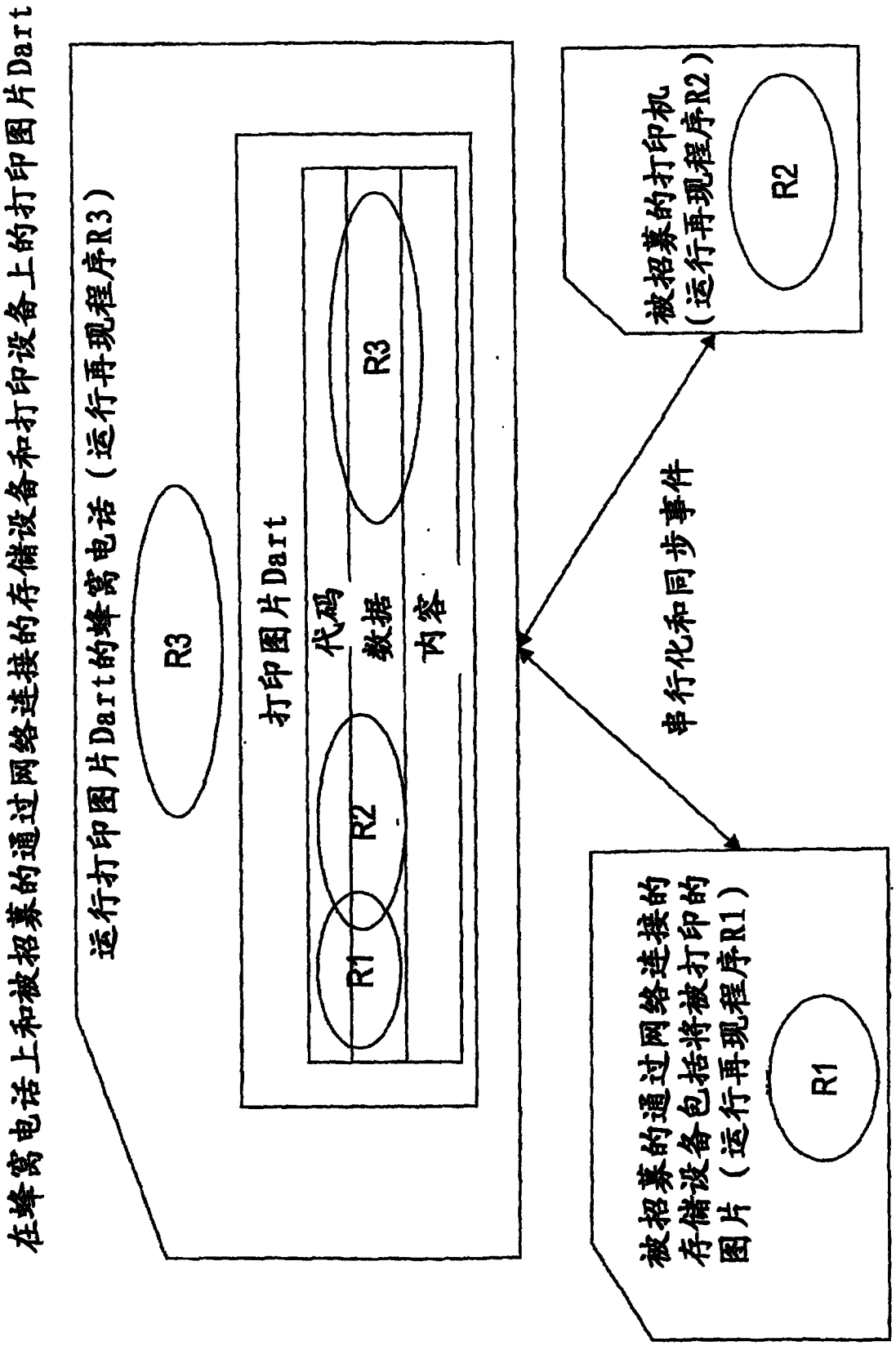

在蜂窝电话上和被招募的通过网络连接的存储设备和打印设备上的打印图片Dart
运行打印图片Dart的蜂窝电话（运行再现程序R3）
R3
打印图片Dart
代码
数据
内容
R1
R2
R3
串行化和同步事件
被招募的打印机（运行再现程序R2）
R2
被招募的通过网络连接的存储设备包括将被打印的图片（运行再现程序R1）
R1

图 8

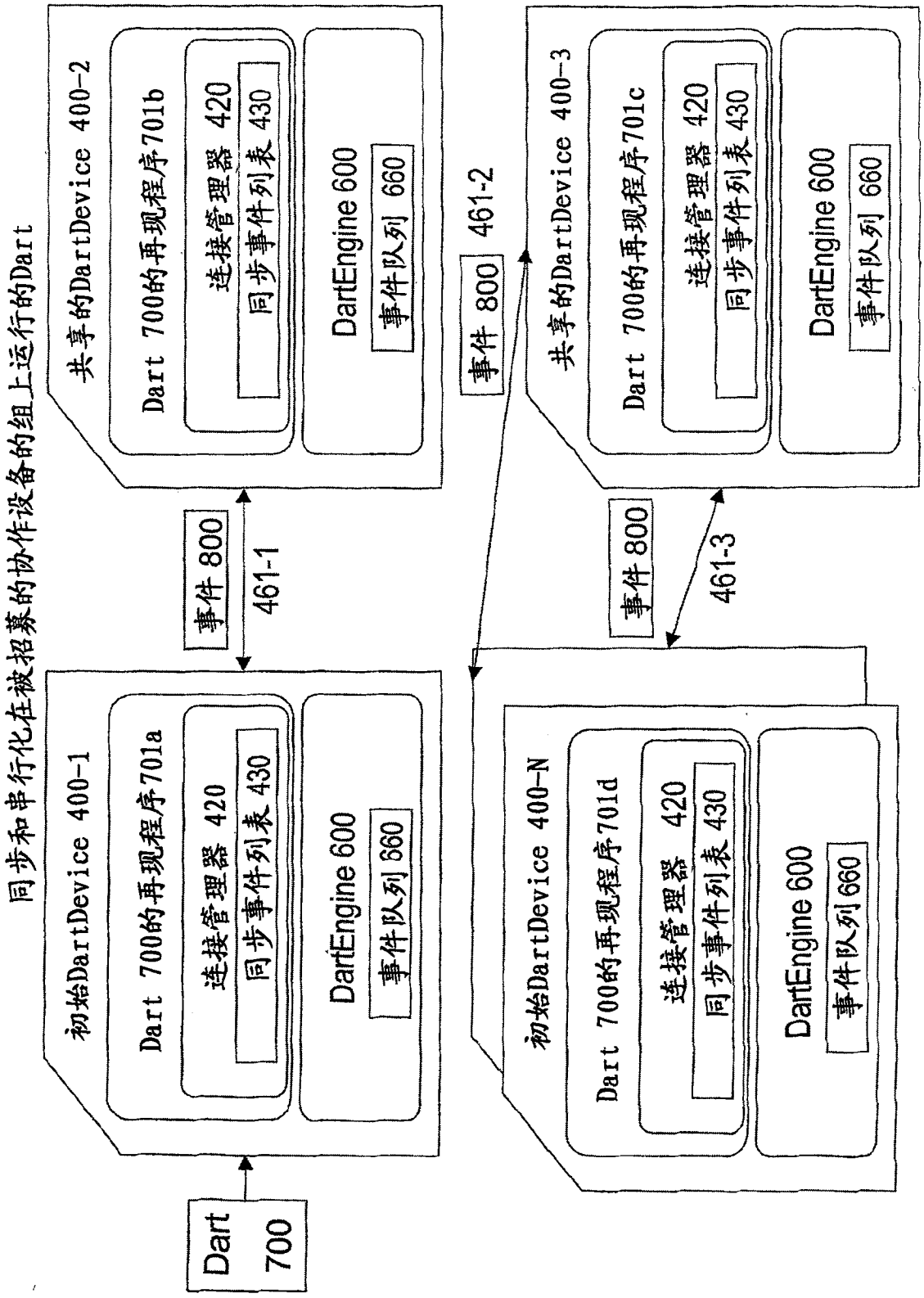

同步和串行化在被招募的协作设备的组上运行的Dart
Dart 700
初始DartDevice 400-1
Dart 700的再现程序701a
连接管理器 420
同步事件列表 430
DartEngine 600
事件队列 660
事件 800
461-1
共享的DartDevice 400-2
Dart 700的再现程序701b
连接管理器 420
同步事件列表 430
DartEngine 600
事件队列 660
事件 800
461-2
共享的DartDevice 400-3
Dart 700的再现程序701c
连接管理器 420
同步事件列表 430
DartEngine 600
事件队列 660
事件 800
461-3
初始DartDevice 400-N
Dart 700的再现程序701d
连接管理器 420
同步事件列表 430
DartEngine 600
事件队列 660

图 9

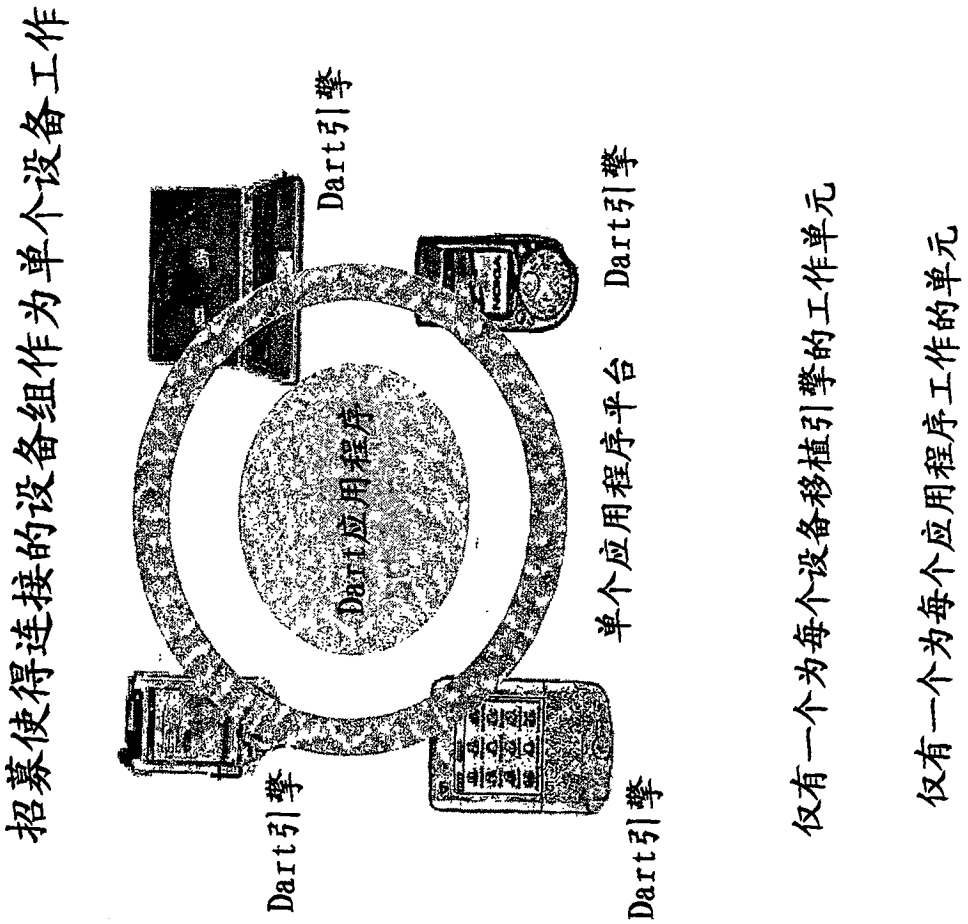

招募使得连接的设备组作为单个设备工作
Dart引擎
Dart引擎
Dart应用程序
单个应用程序平台
Dart引擎
Dart引擎
仅有一个为每个设备移植引擎的工作单元
仅有一个为每个应用程序工作的单元

图 10

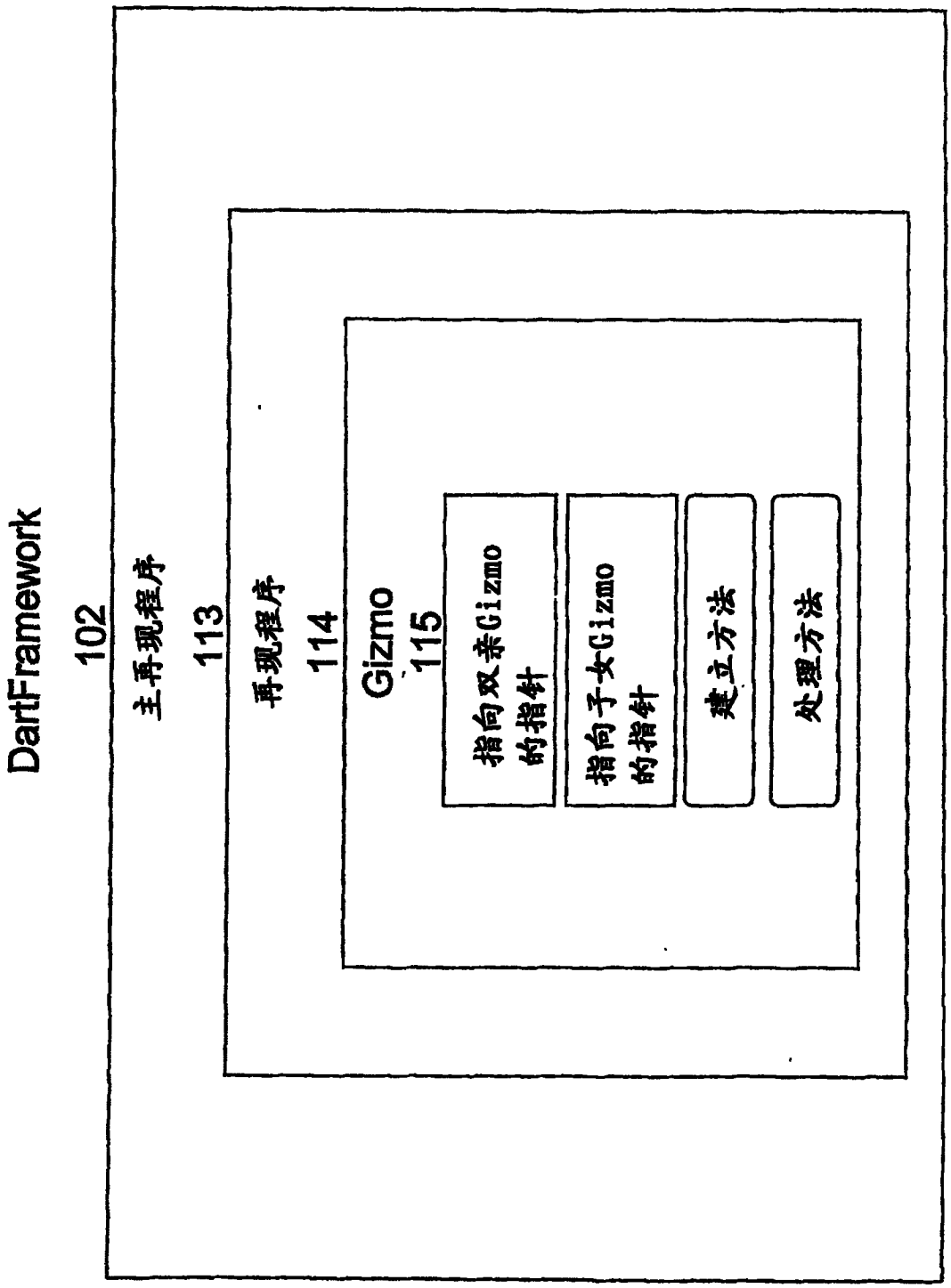

DartFramework
102
主再现程序
113
再现程序
114
Gizmo
115
指向双亲Gizmo的指针
指向子女Gizmo的指针
建立方法
处理方法

图 11

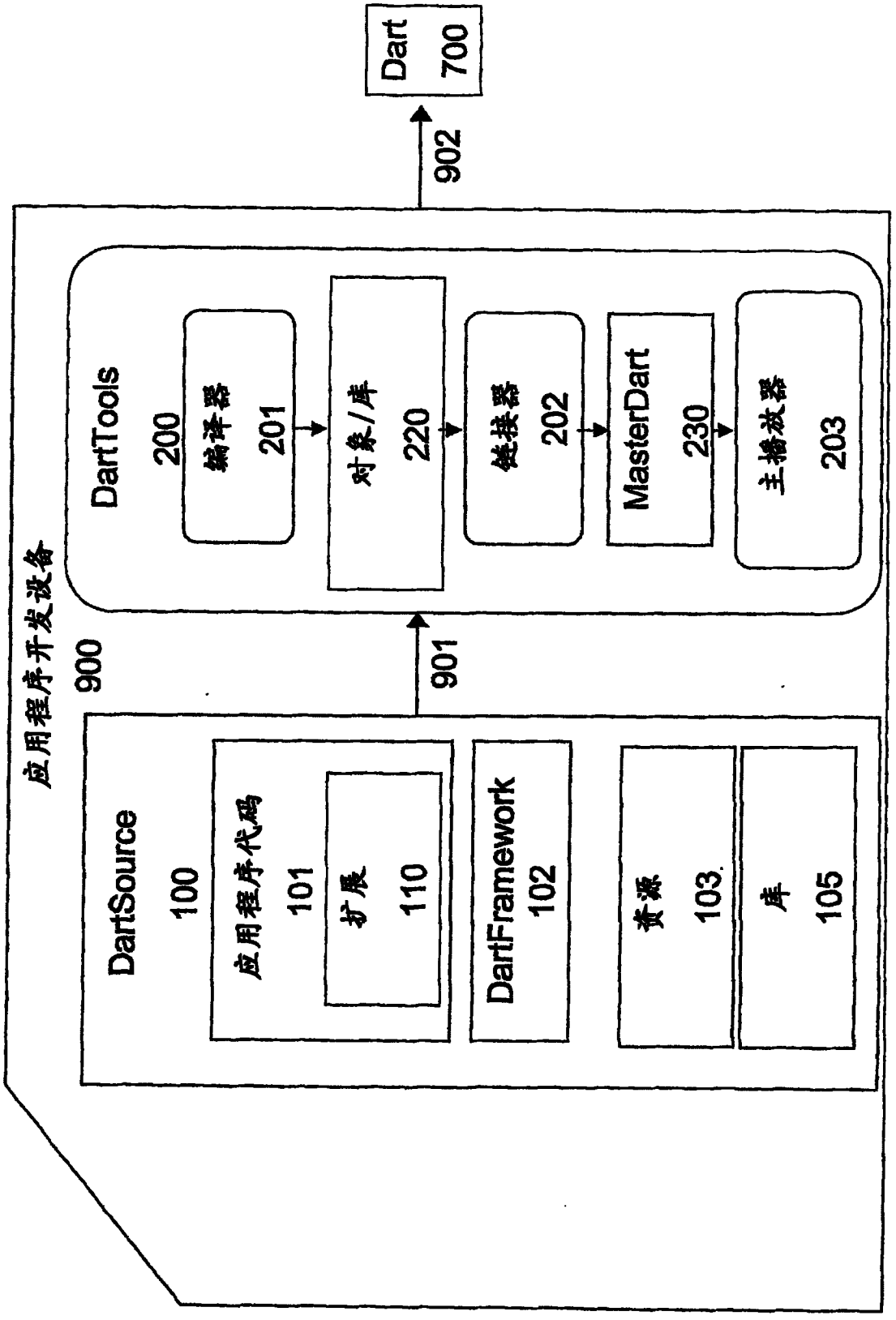

Dart
700
902
应用程序开发设备
900
DartTools
200
编译器
201
对象/库
220
链接器
202
MasterDart
230
主播放器
203
901
DartSource
100
应用程序代码
101
扩展
110
DartFramework
102
资源
103
库
105

图 12

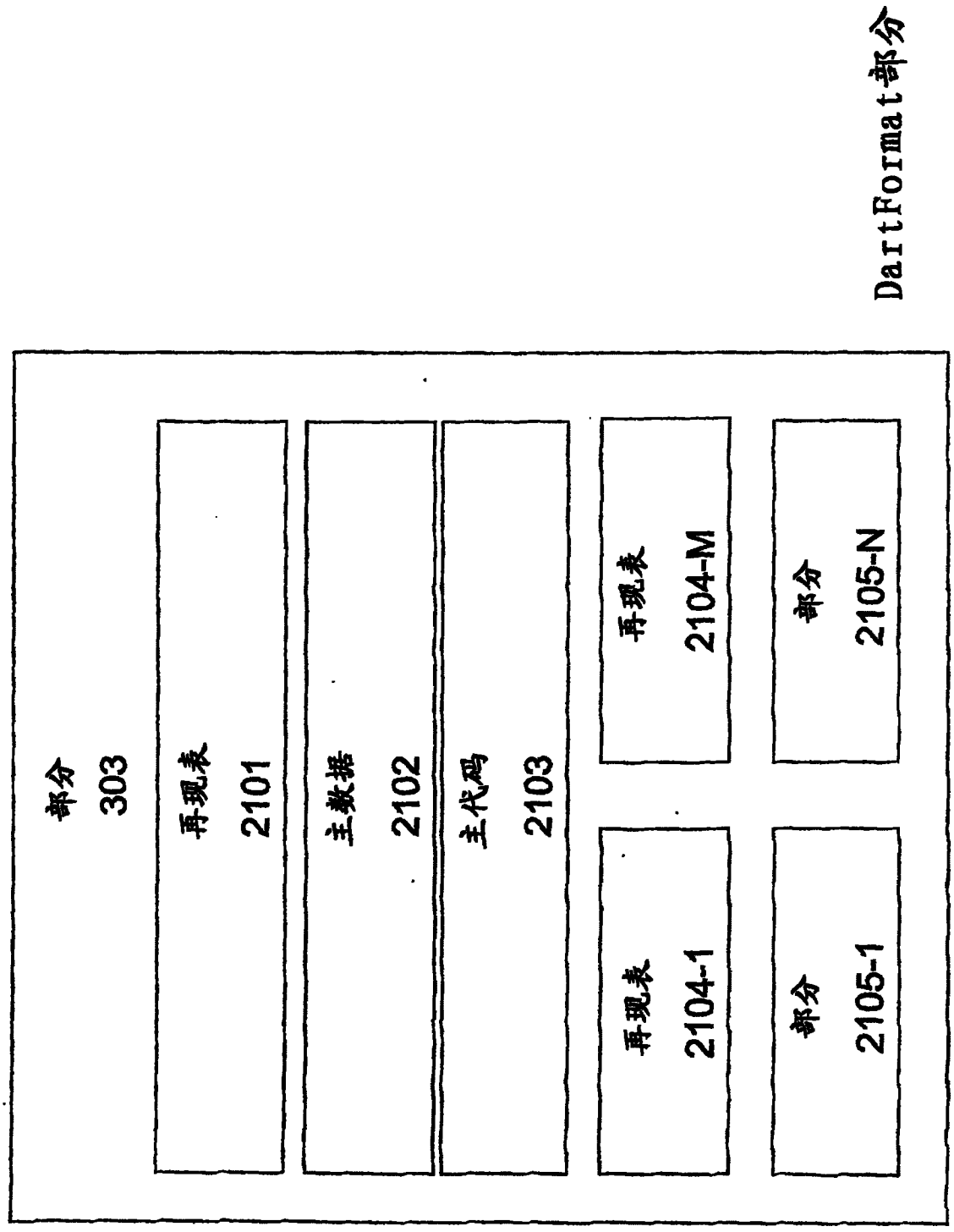

DartFormat部分
部分
303
再现表
2101
主数据
2102
主代码
2103
再现表
2104-1
再现表
2104-M
部分
2105-1
部分
2105-N

图 13

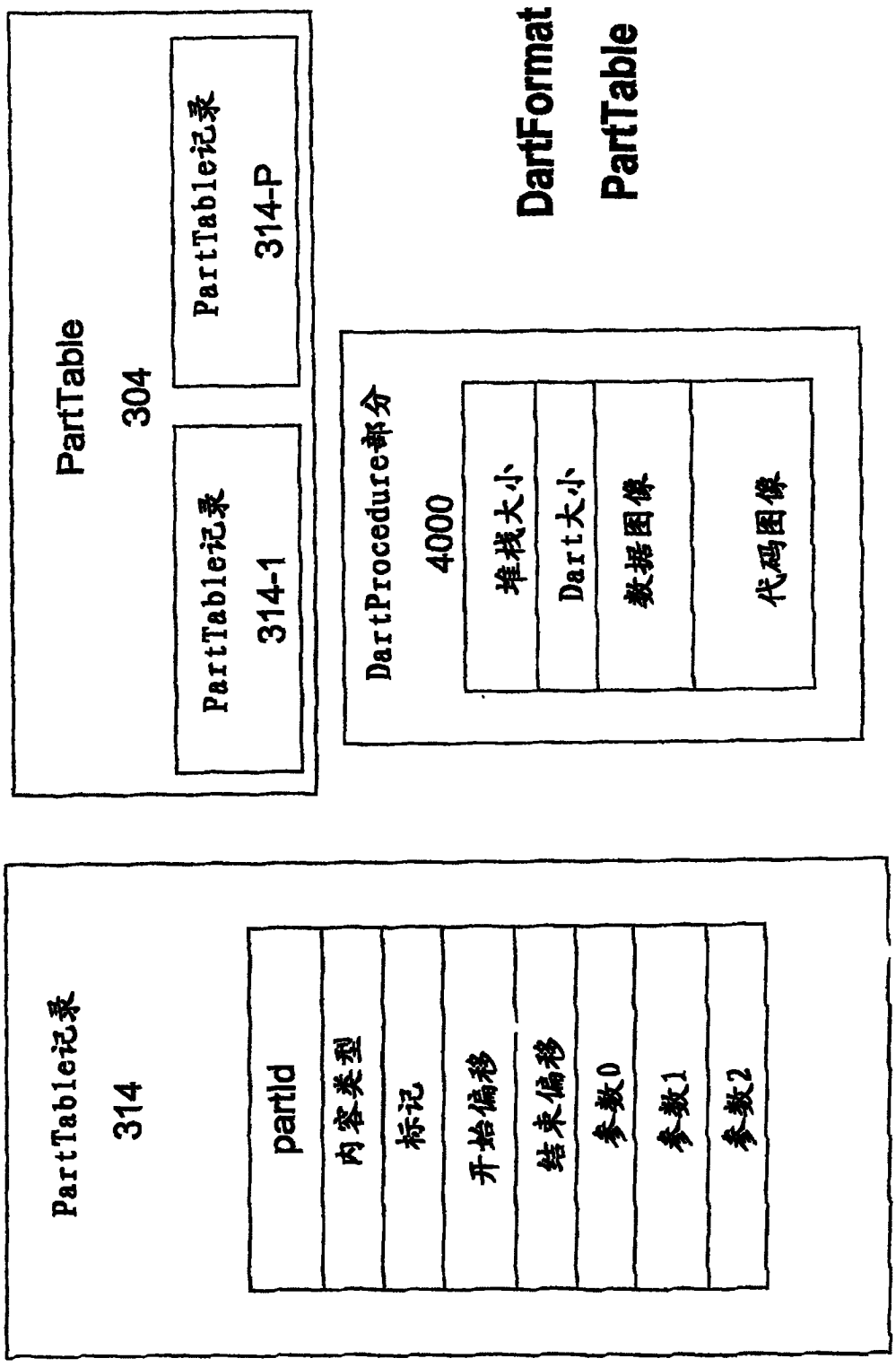

PartTable
304
PartTable记录
314-1
PartTable记录
314-P
DartProcedure部分
4000
堆栈大小
Dart大小
数据图像
代码图像
DartFormat
PartTable
PartTable记录
314
partId
内容类型
标记
开始偏移
结束偏移
参数0
参数1
参数2

图 14

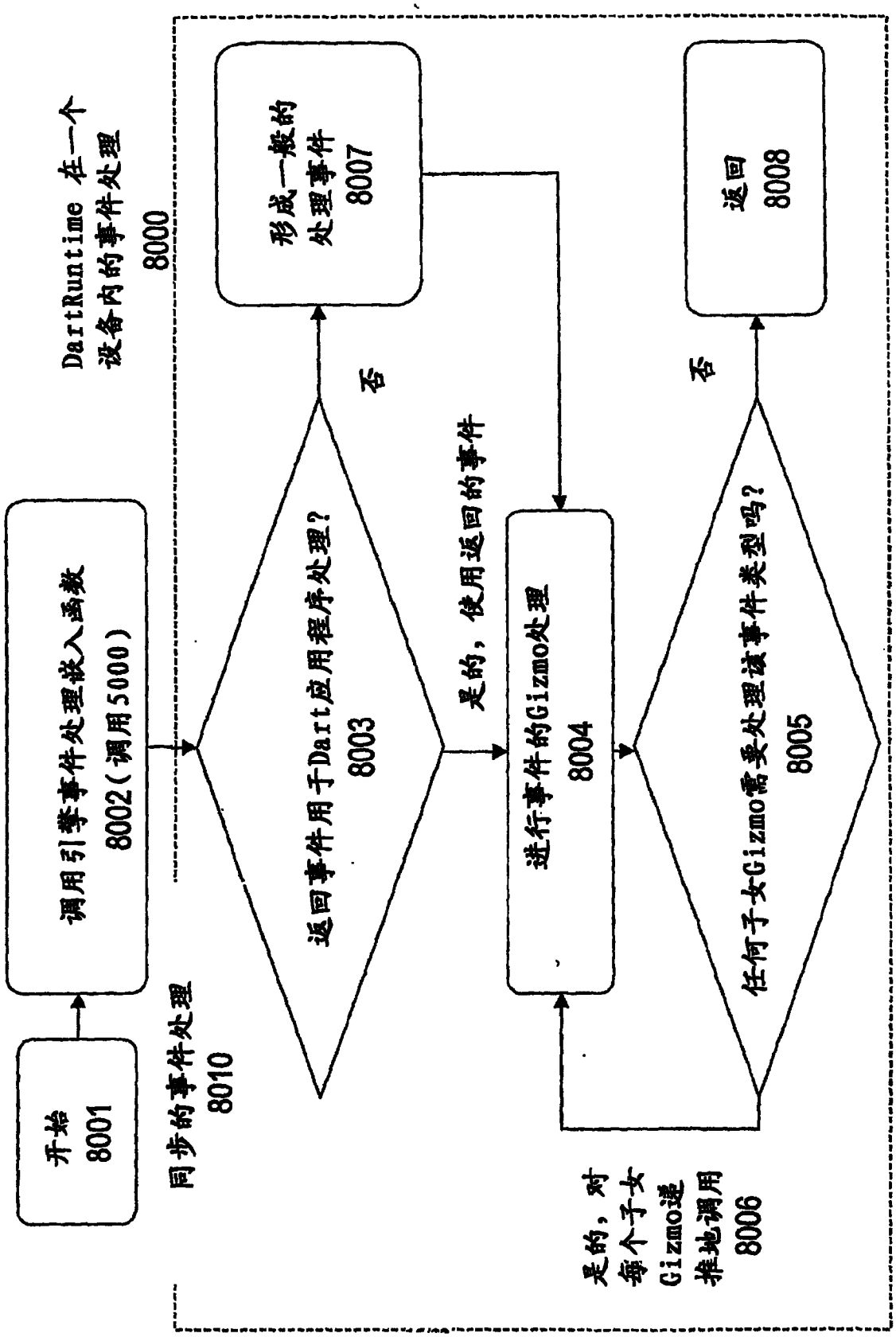

DartRuntime 在一个设备内的事件处理
8000
开始
8001
调用引擎事件处理嵌入函数
8002(调用5000)
同步的事件处理
8010
返回事件用于Dart应用程序处理?
8003
否
形成一般的处理事件
8007
是的，使用返回的事件
进行事件的Gizmo处理
8004
任何子女Gizmo需要处理该事件类型吗?
8005
否
返回
8008
是的，对每个子女Gizmo递推地调用
8006

图 15

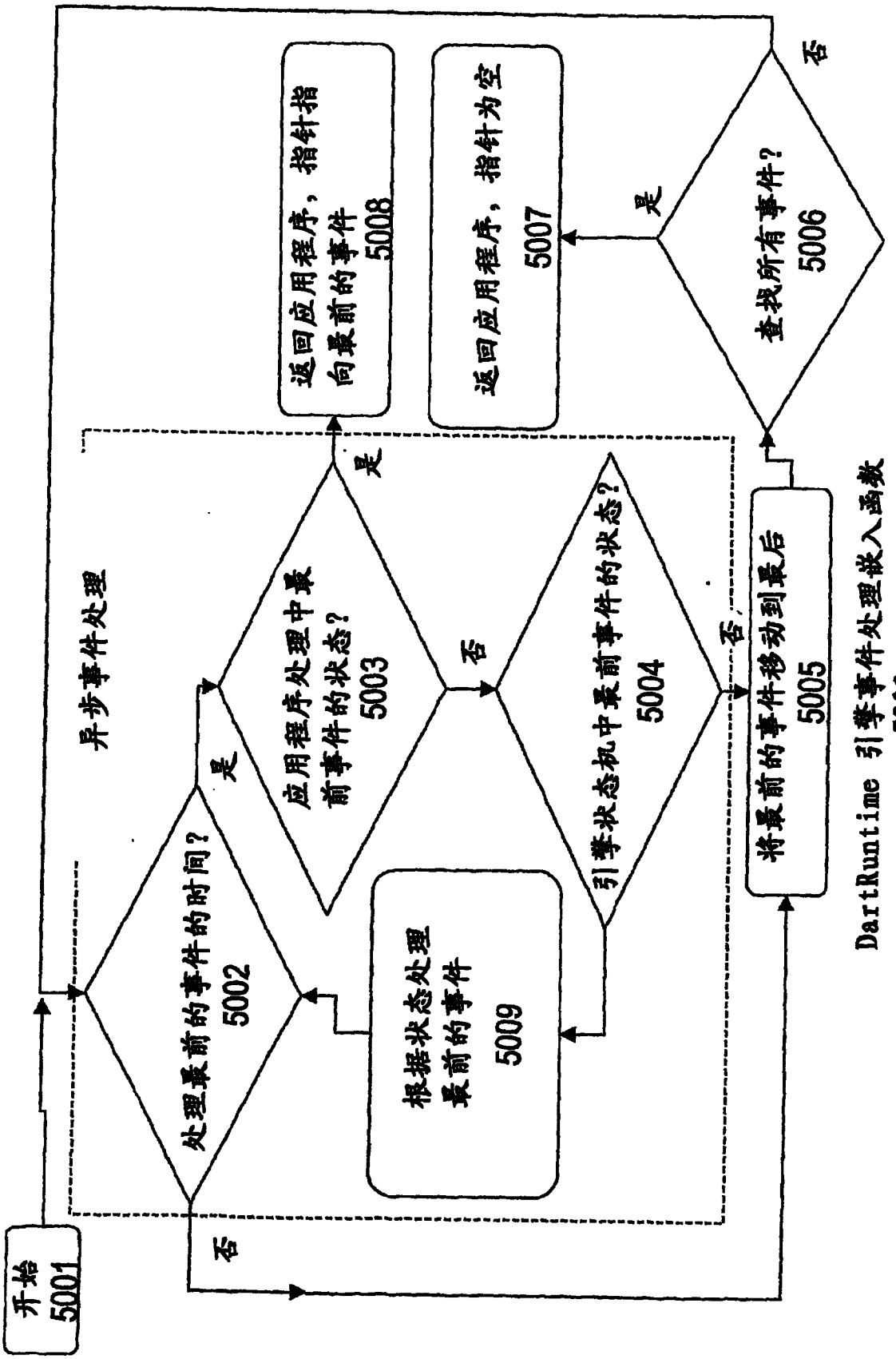

开始
5001
异步事件处理
处理最前的事件的时间？
5002
否
是
应用程序处理中最前事件的状态？
5003
是
返回应用程序，指针指向最前的事件
5008
否
引擎状态机中最前事件的状态？
5004
否
根据状态处理最前的事件
5009
将最前的事件移动到最后
5005
查找所有事件？
5006
是
返回应用程序，指针为空
5007
否
DartRuntime 引擎事件处理嵌入函数
5000

图 16

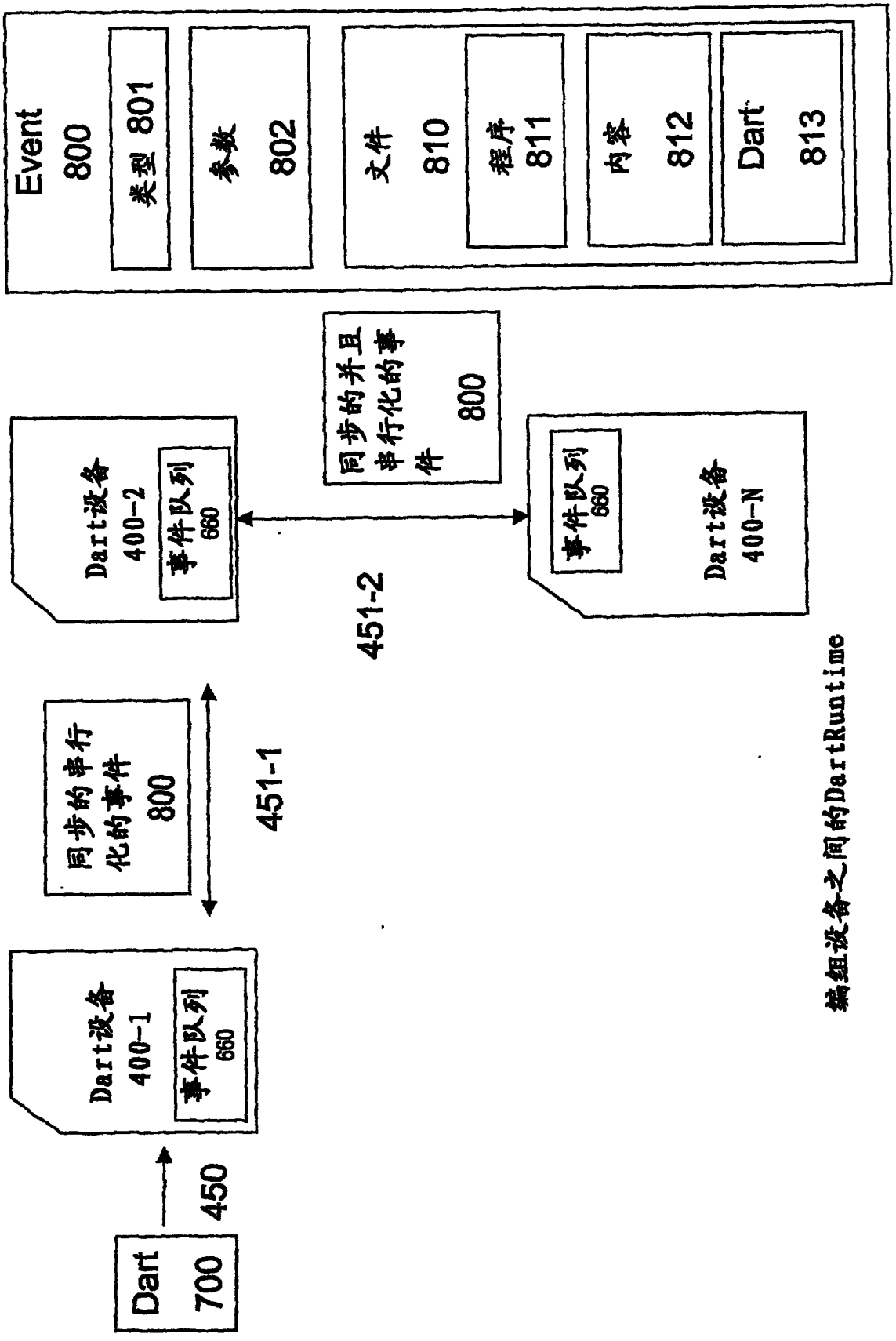

Event
800
类型 801
参数
802
文件
810
程序
811
内容
812
Dart
813
同步的并且串行化的事件
800
Dart设备
400-2
事件队列
660
451-2
事件队列
660
Dart设备
400-N
同步的串行化的事件
800
451-1
Dart设备
400-1
事件队列
660
450
Dart
700
编组设备之间的DartRuntime

图 17

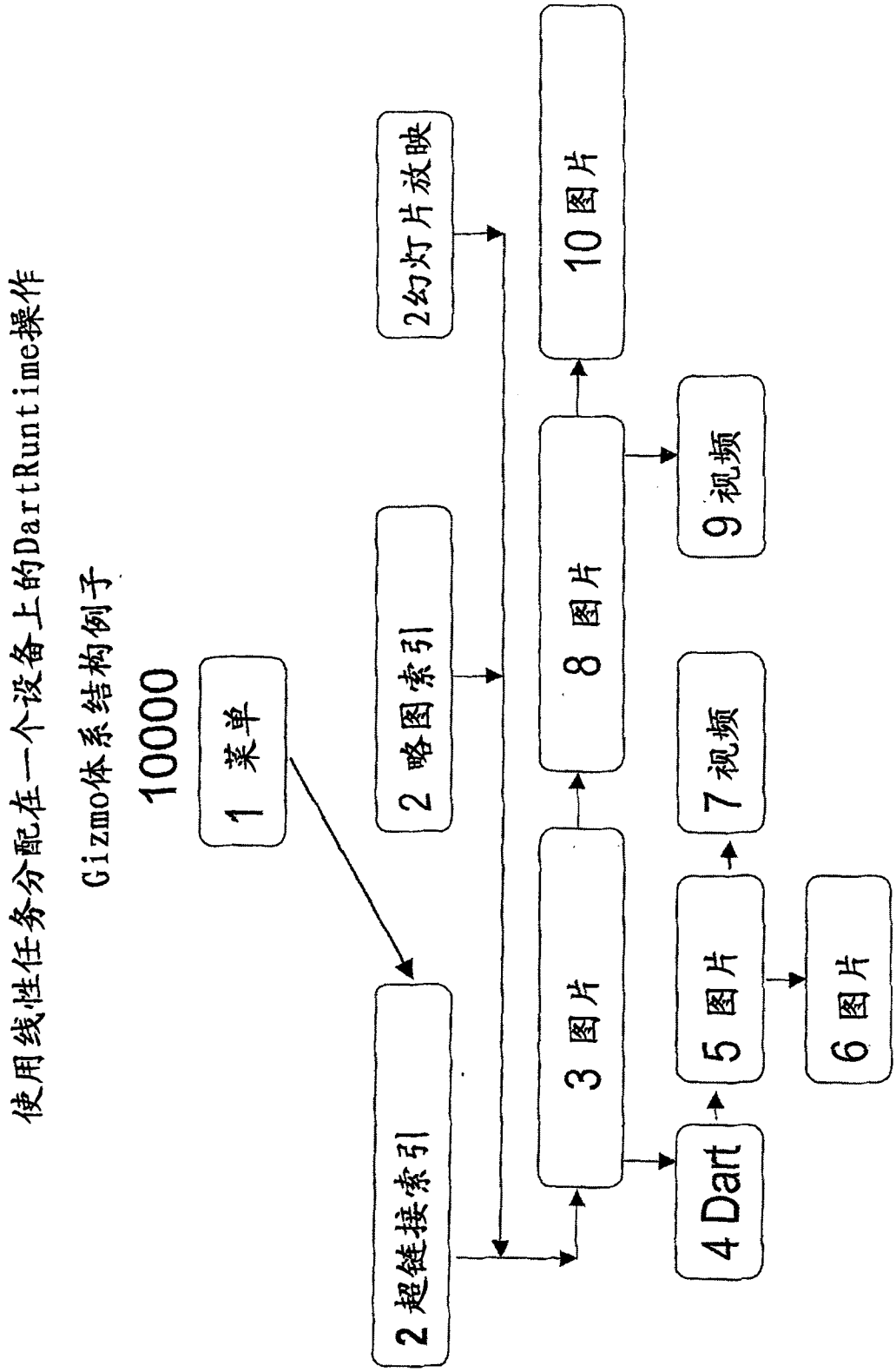

使用线性任务分配在一个设备上的DartRuntime操作
Gizmo体系结构例子
10000
1 菜单
2 略图索引
2幻灯片放映
10 图片
2 超链接索引
3 图片
8 图片
9 视频
4 Dart
5 图片
7 视频
6 图片

图 18

DART 应用程序级错误恢复

步骤1：A->B->C，A上的Dart招募B，接着B招募C；所有这三个设备通过共享一组同步的串行化的事件，运行最初为A上的初始Dart的一部分的Dart再现程序。

步骤2：A->B C，失去到C的连接，由B上的通信代码生成失去连接类型事件，由于B是被招募的设备组的一部分因此它自动地回到A

步骤3：A->B C，现在在A和B上运行的Dart尽可能地补偿C的失去，例如：

a 不试图与C同步事件，因为C不是必须的，或者

b A和或B接替C的任务，或者

c A->B->C，A和或B招募另一个设备，D，以承担C的任务并关闭失去连接的会话，或者

d A和或B中止其操作，直到C回来或者超时和关闭，因为C对于实现初始Dart的目的是必须的

步骤4：A->B C，只要A和或B没有通过关闭失去连接的会话来对失去连接的事件进行响应，则B的通信系统继续查找C，因为C没有使用事件进行关闭

步骤5：B的通信可能通过不同的协议再次建立与C的通信。

步骤6：A->B->C，B上的Dart再现程序重新招募具有当前运行状态的C，当C失去连接时，该状态可能变化

图 19

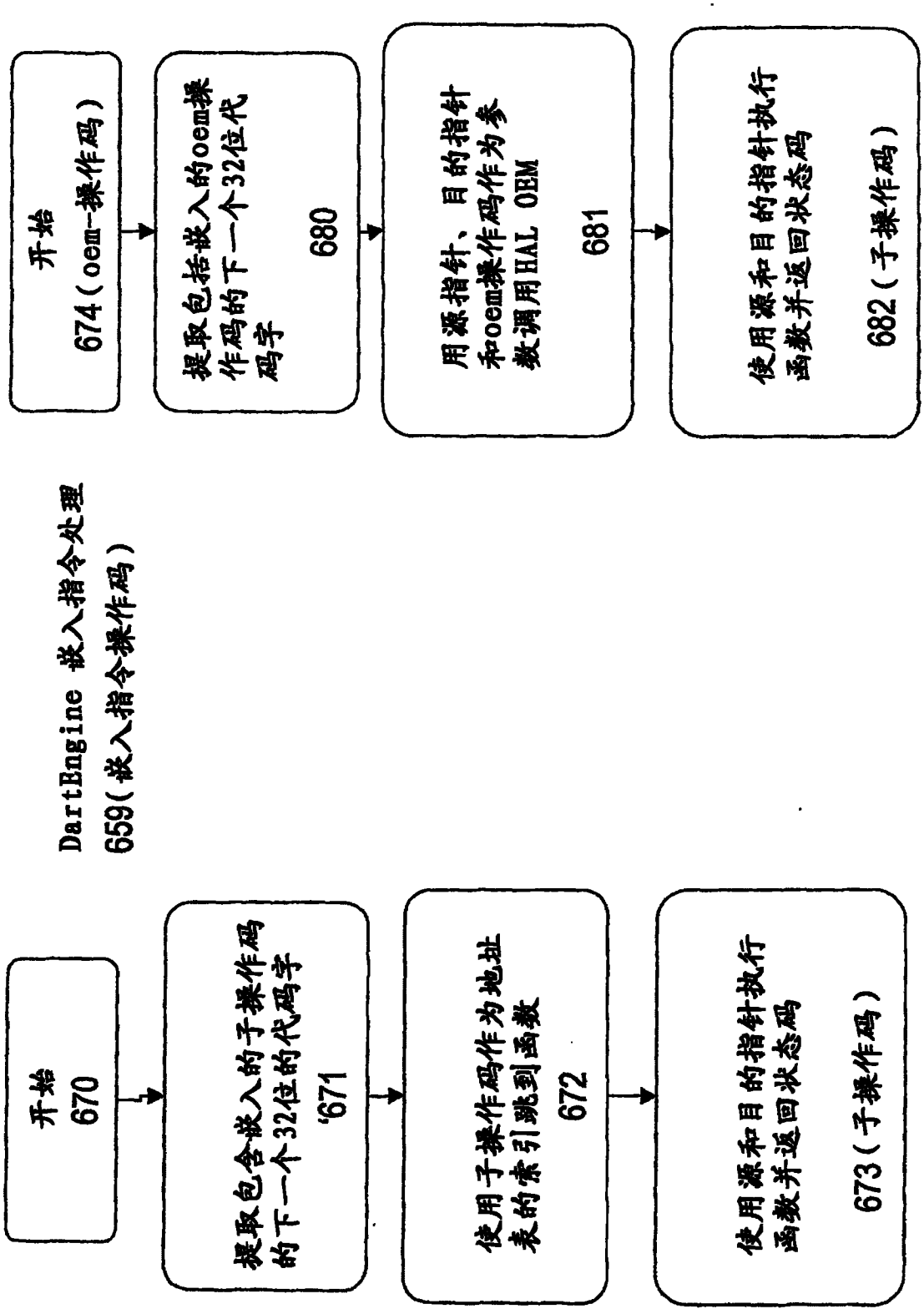

开始
674(oem-操作码)
提取包括嵌入的oem操作码的下一个32位代码字
680
用源指针、目的指针和oem操作码作为参数调用HAL OEM
681
使用源和目的指针执行函数并返回状态码
682(子操作码)
DartEngine 嵌入指令处理
659(嵌入指令操作码)
开始
670
提取包含嵌入的子操作码的下一个32位的代码字
671
使用子操作码作为地址表的索引跳到函数
672
使用源和目的指针执行函数并返回状态码
673(子操作码)

图 20

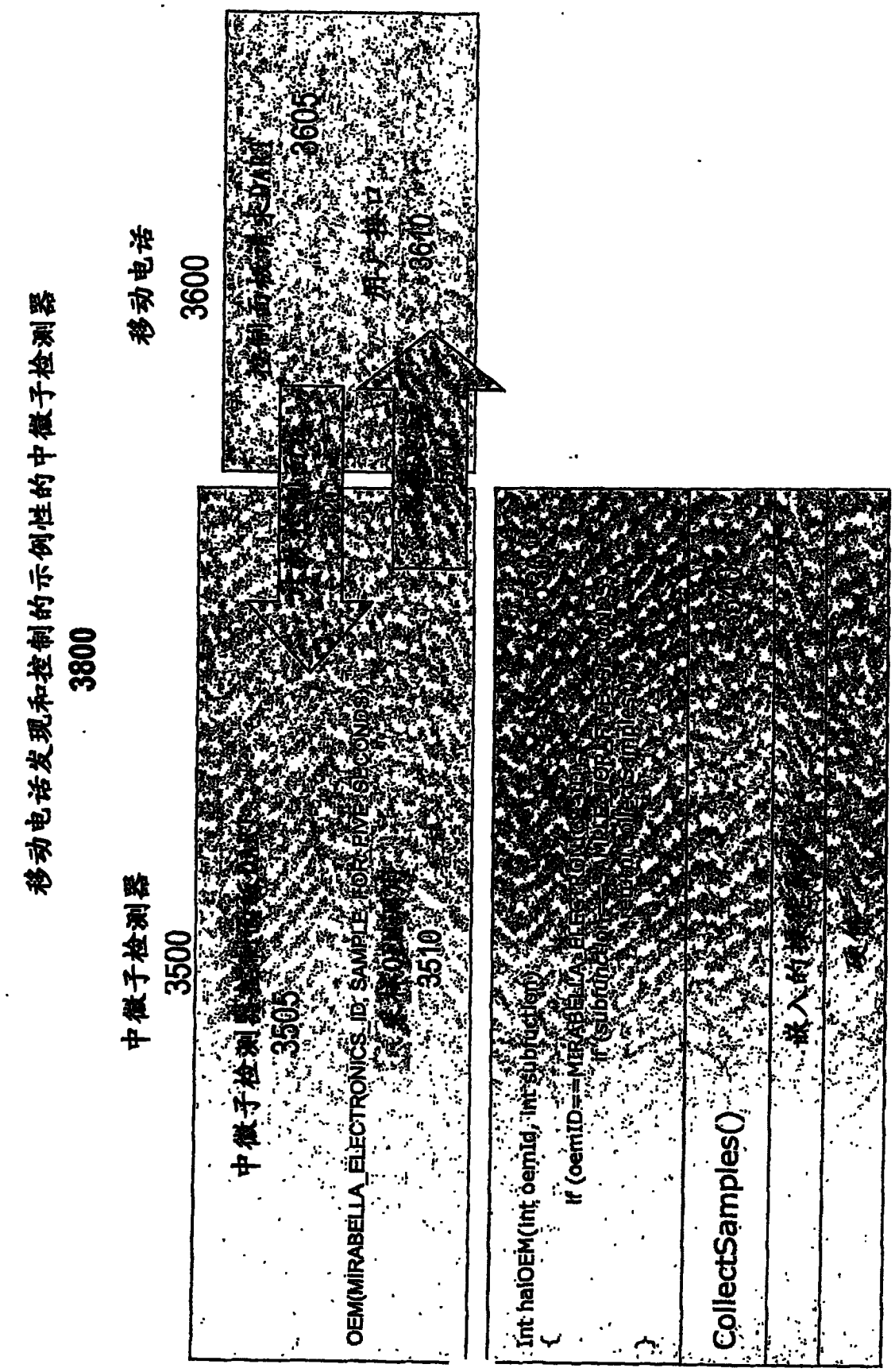

移动电话发现和控制的示例性的中微子检测器
3800
中微子检测器
3500
移动电话
3600
3605
3610
3505
3510
OEM(MIRABELLA_ELECTRONICS_ID, SAMPLE_FOR_FIVE_SECONDS);
int haloOEM(int oemId, int subfunction)
if (oemID==MIRABELLA_ELECTRONICS_ID)
CollectSamples()

图 21

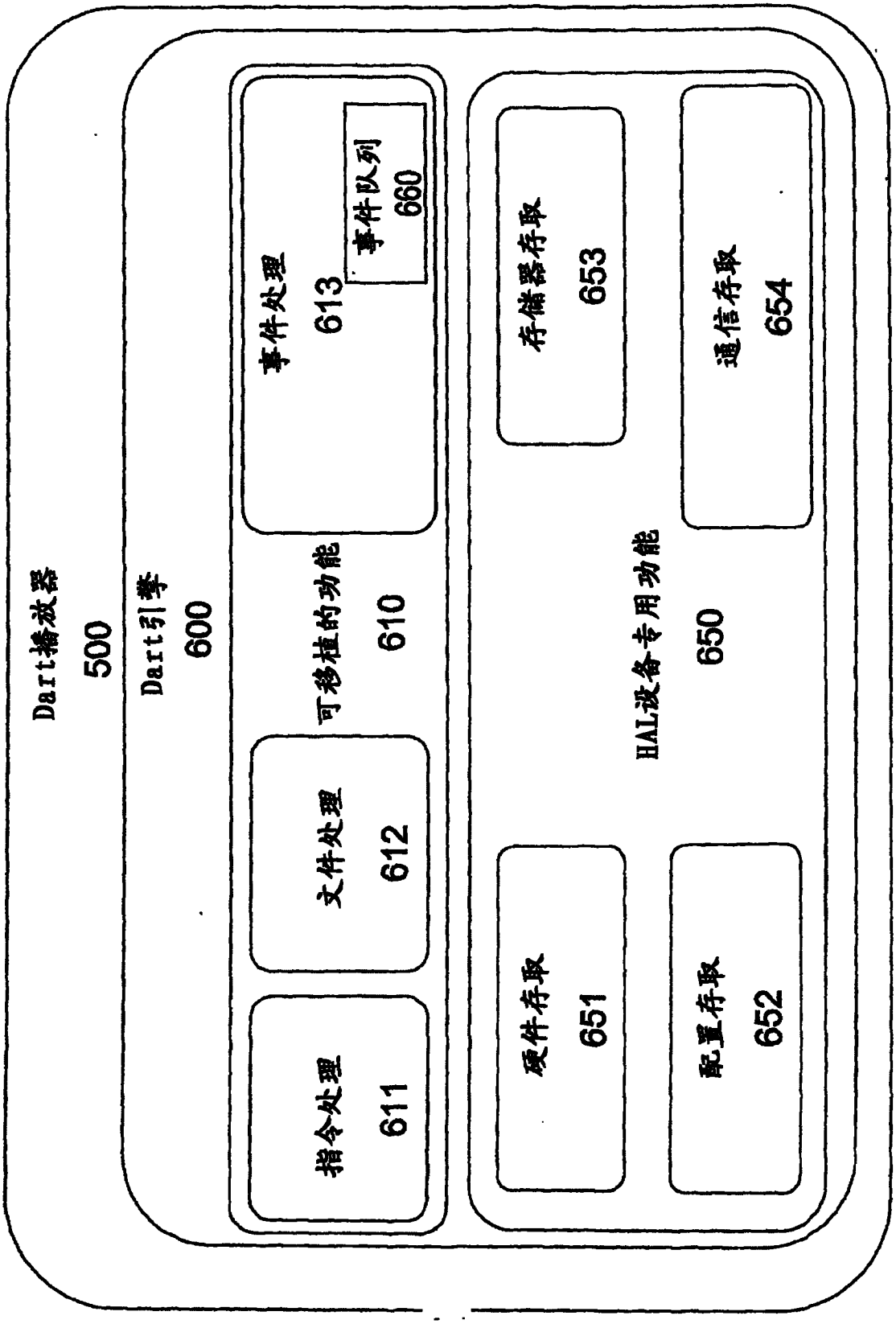

Dart播放器
500
Dart引擎
600
可移植的功能
610
指令处理
611
文件处理
612
事件处理
613
事件队列
660
HAL设备专用功能
650
硬件存取
651
配置存取
652
存储器存取
653
通信存取
654

图 22

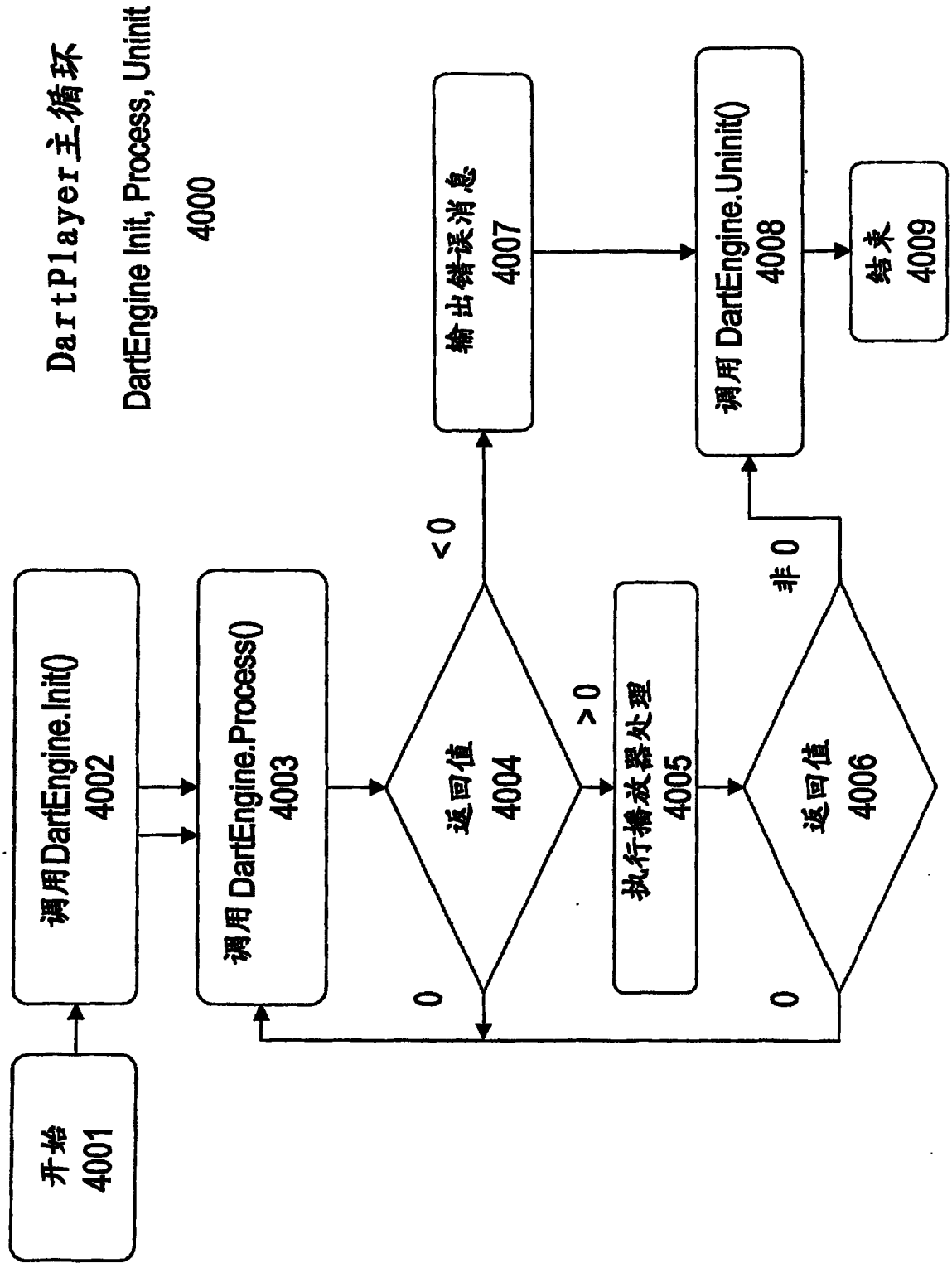

DartPlayer主循环
DartEngine Init, Process, Uninit
4000
开始
4001
调用DartEngine.Init()
4002
调用 DartEngine.Process()
4003
返回值
4004
<0
>0
0
输出错误消息
4007
执行播放器处理
4005
返回值
4006
非0
0
调用 DartEngine.Uninit()
4008
结束
4009

图 23

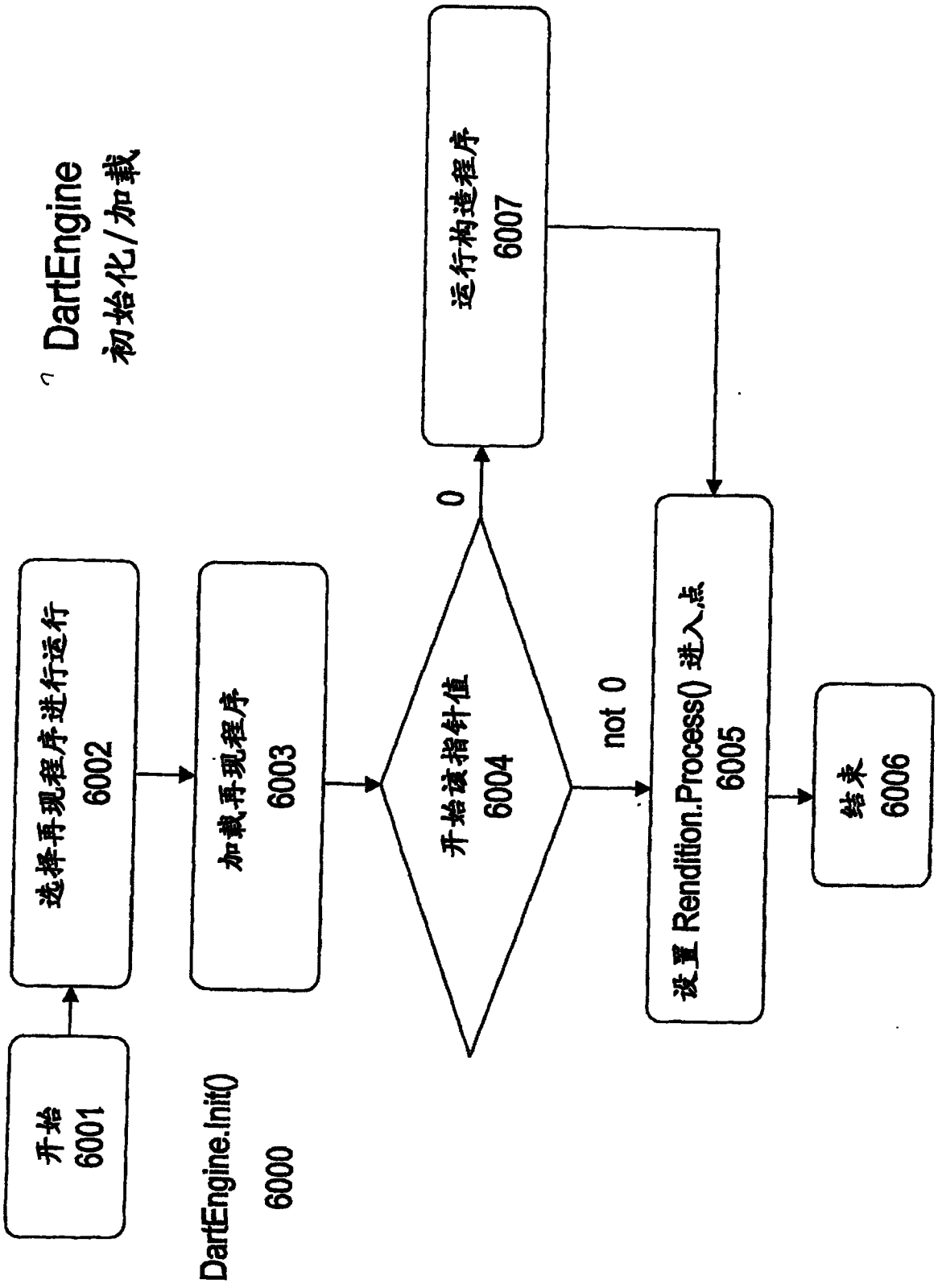

DartEngine
初始化/加载
开始
6001
选择再现程序进行运行
6002
加载再现程序
6003
开始读指针值
6004
0
运行构造程序
6007
not 0
设置 Rendition.Process() 进入点
6005
结束
6006
DartEngine.Init()
6000

图 24

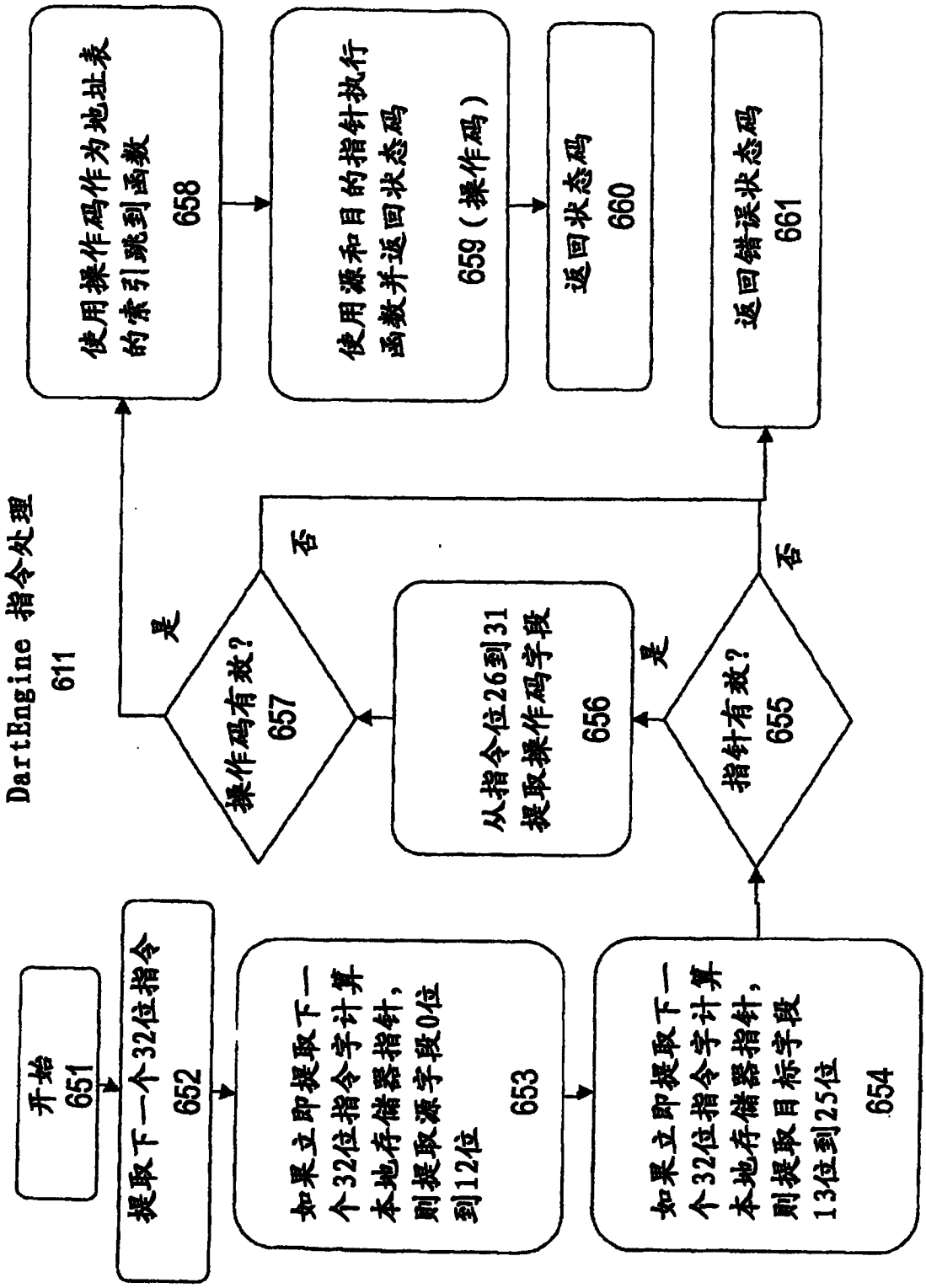

DartEngine 指令处理
611
开始
651
提取下一个32位指令
652
如果立即提取下一个32位指令字计算本地存储器指针，则提取源字段0位到12位
653
如果立即提取下一个32位指令字计算本地存储器指针，则提取目标字段13位到25位
654
指针有效?
655
从指令位26到31提取操作码字段
656
操作码有效?
657
使用操作码作为地址表的索引跳到函数
658
使用源和目的指针执行函数并返回状态码
659（操作码）
返回状态码
660
返回错误状态码
661
是
否
是
否

图 25

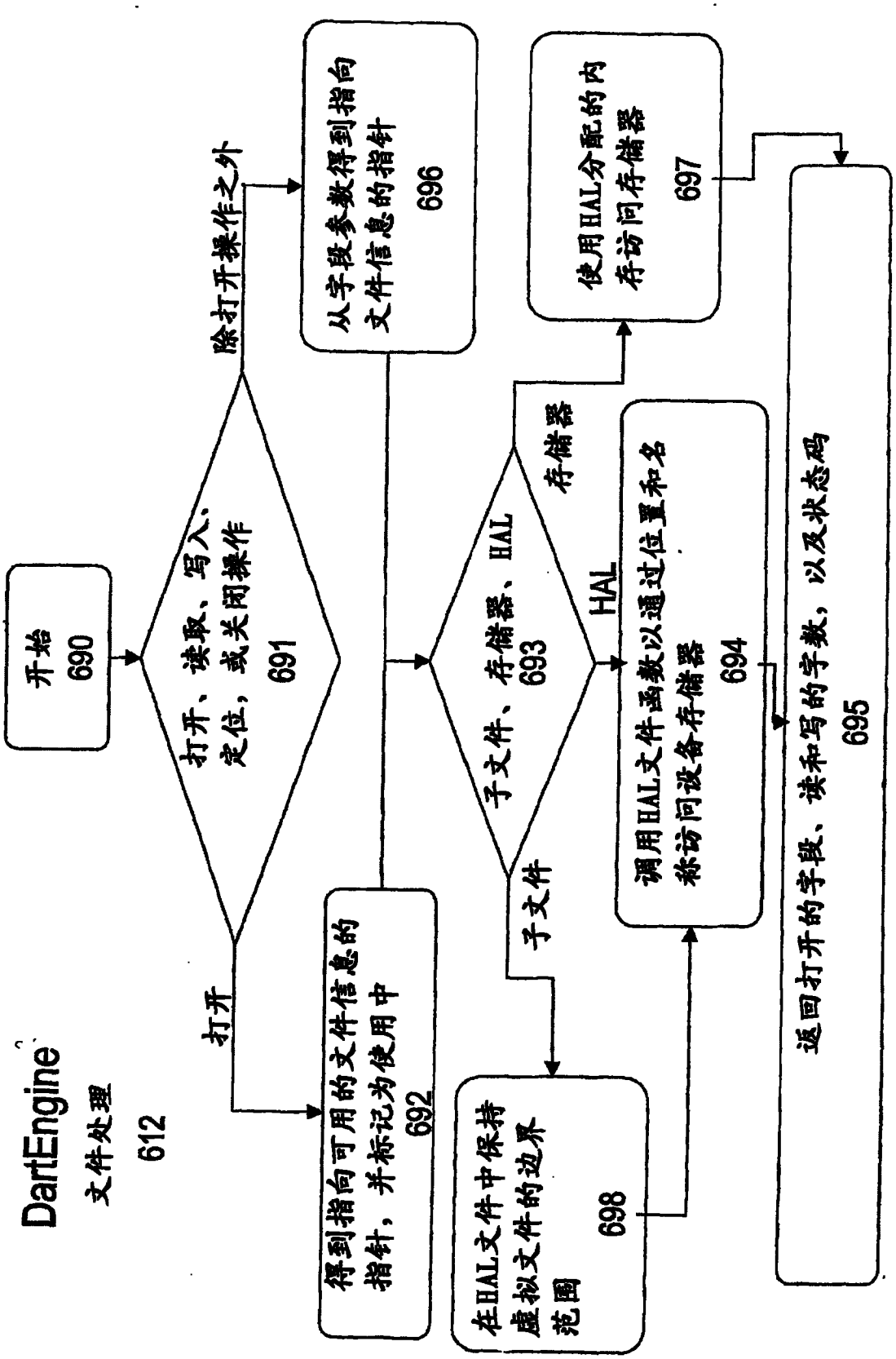

DartEngine
文件处理
612
开始
690
打开、读取、写入、定位，或关闭操作
691
除打开操作之外
从字段参数得到指向文件信息的指针
696
打开
得到指向可用的文件信息的指针，并标记为使用中
692
子文件、存储器、HAL
693
子文件
存储器
HAL
使用HAL分配的内存访问存储器
697
调用HAL文件函数以通过位置和名称访问设备存储器
694
在HAL文件中保持虚拟文件的边界范围
698
返回打开的字段、读和写的字数，以及状态码
695

图 26

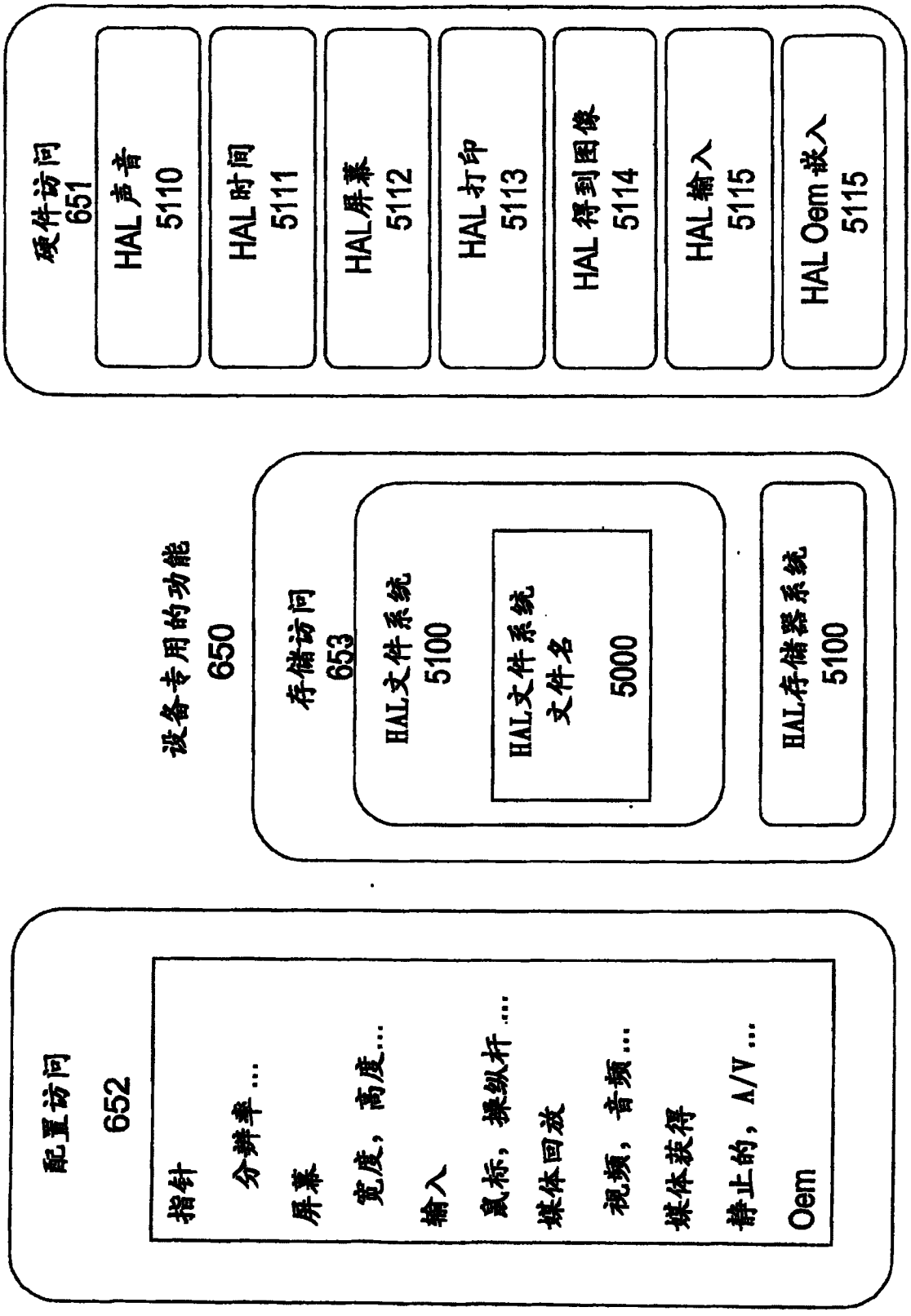

硬件访问
651
HAL 声音
5110
HAL 时间
5111
HAL屏幕
5112
HAL 打印
5113
HAL 得到图像
5114
HAL 输入
5115
HAL Oem 嵌入
5115
设备专用的功能
650
存储访问
653
HAL文件系统
5100
HAL文件系统
文件名
5000
HAL存储器系统
5100
配置访问
652
指针
分辨率...
屏幕
宽度，高度...
输入
鼠标，操纵杆...
媒体回放
视频，音频...
媒体获得
静止的，A/V...
Oem

图 27

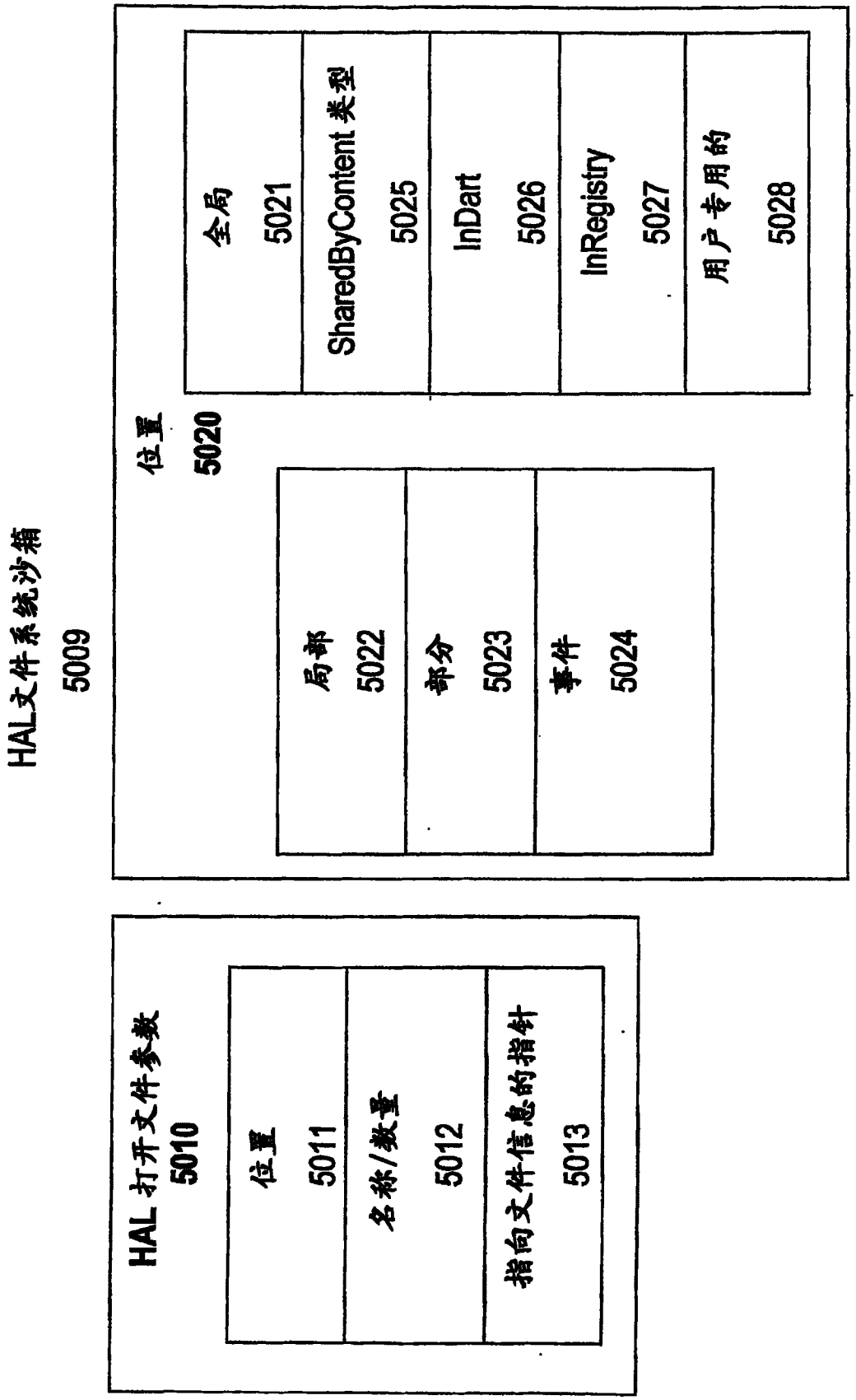

HAL文件系统沙箱
5009
位置
5020
全局
5021
SharedByContent类型
5025
InDart
5026
InRegistry
5027
用户专用的
5028
局部
5022
部分
5023
事件
5024
HAL 打开文件参数
5010
位置
5011
名称/数量
5012
指向文件信息的指针
5013

图 28

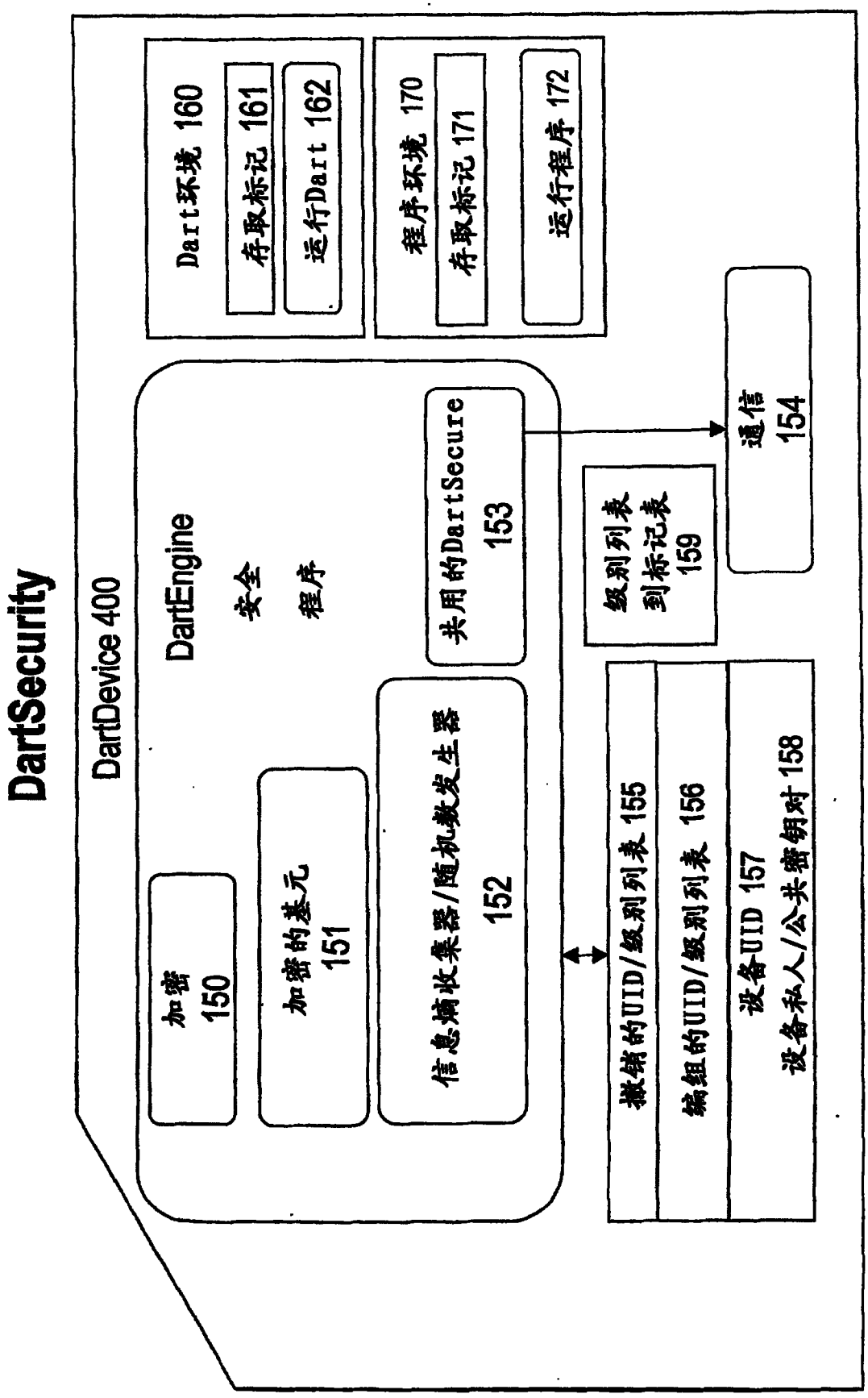

DartSecurity
DartDevice 400
DartEngine
安全
程序
加密
150
加密的基元
151
信息摘收集器/随机数发生器
152
共用的DartSecure
153
Dart环境 160
存取标记 161
运行Dart 162
程序环境 170
存取标记 171
运行程序 172
级别列表
到标记表
159
通信
154
撤销的UID/级别列表 155
编组的UID/级别列表 156
设备UID 157
设备私人/公共密钥对 158

图 29

社会同步联系列表事例
步骤1: Dan在设备A上创建联系列表，将设备B端组，接着设备B将设备C端组，所有设备包括输入在A上的两个名称，初始DartX的再现程序现在被指定为其它设备上的Dart Y和Dart Z
步骤2: 所有设备彼此断开连接，将Rachel添加到C上的列表中，设备显示如下:
Dan的膝上型PC DartDevice A
Dan的联系列表 - - 128位UidA
1. Arnold, Matt
2. Barker, Becca
联系列表处理Dart X - - 128位UidX
社会同步对象Uid/DartHandler UiD列表
1. 128位UidA/128位UidX
Dan的PDADartDevice B
Dan的联系列表 - - 128位UidA
1. Arnold, Matt
2. Barker, Becca
联系列表处理Dart Y - - 128位UidY
社会同步对象Uid/DartHandler UiD列表
1. 128位UidA/128位UidY
Dan的蜂窝电话DartDevice C
Dan的联系列表 - - 128位UidA
1. Arnold, Matt 3. Conners, Rachel
2. Barker, Becca
联系列表处理Dart Z - - 128位UidZ
社会同步对象Uid/DartHandler UiD列表
1. 128位UidA/128位UidZ
步骤3: 设备C连接到设备A, 看到在其同步列表中具有相同的128位UIDA, 开始相关的ContactListHandlerDart Z, 它使用相算将其执行传输到C, 并使用包括在Dart中的任何规则合并A和C的列表, 以进制同步
设备A和C现在在其列表中有Matt、Becca、Rachel, 设备B仍然只有Matt、Becca
步骤4: 设备B连接到设备A或C, 看到在其同步列表中具有相同的128位UIDA, 开始相关的ContactListHandlerDart Y, 它使用相算将其执行传输到连接的设备上, 并使用包括在Dart中的任何规则合并列表, 以进制同步
设备A、B和C现在在其列表中有Matt、Becca、Rachel
注意: 即使A在同步之前从没有与C通信, 但是它们知道它们需要同步列表, 因为它们具有用于通过B的列表的相同的唯一ID

图 30

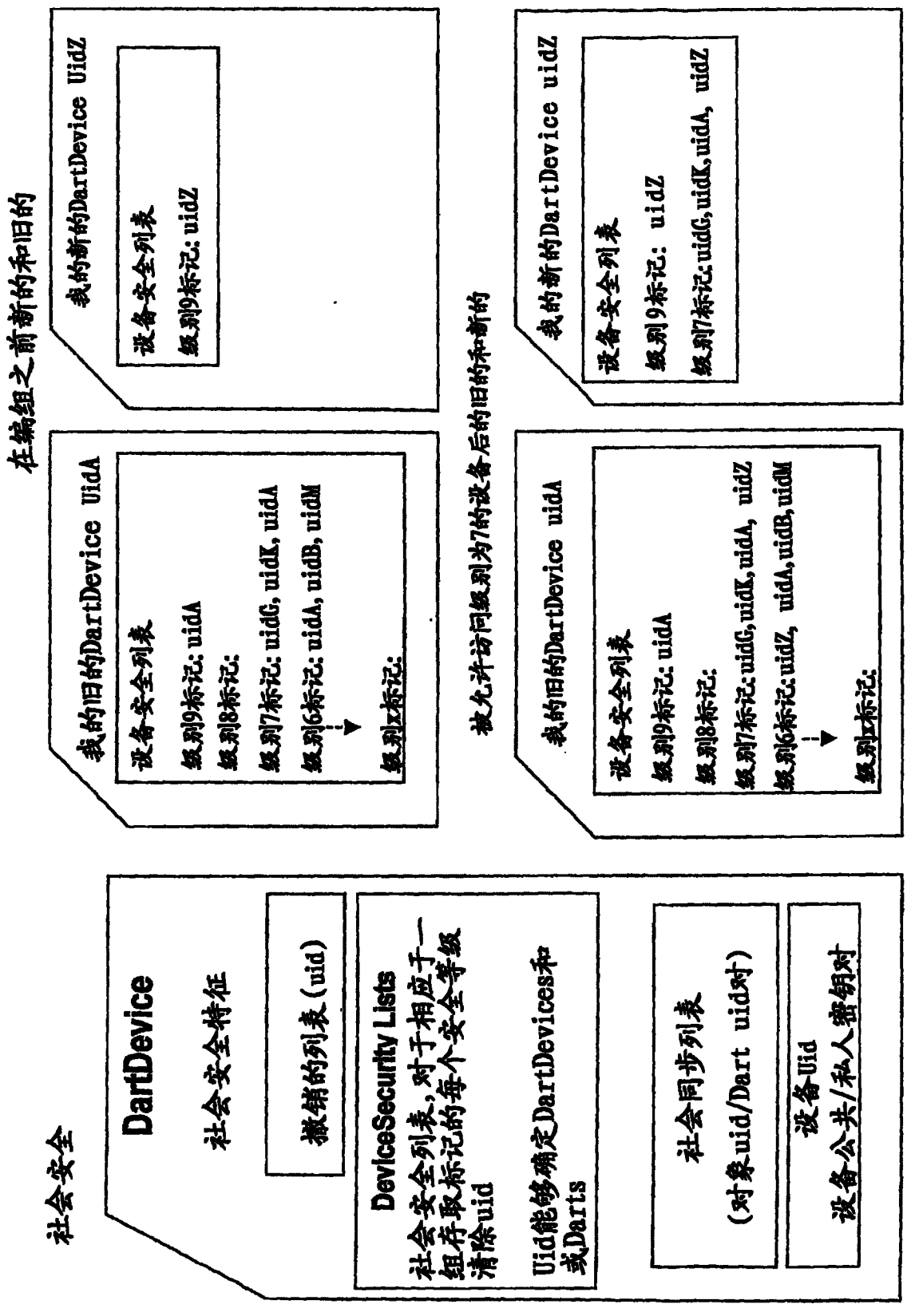

社会安全
DartDevice
社会安全特征
被销的列表（uid）
DeviceSecurity Lists
社会安全列表，对于相应于一组存取标记的每个安全等级清除uid
Uid能够确定DartDevices和或Darts
社会同步列表
（对象uid/Dart uid对）
设备Uid
设备公共/私人密钥对
在编组之前新的和旧的
我的旧的DartDevice UidA
设备安全列表
级别9标记：uidA
级别8标记：
级别7标记：uidG，uidK，uidA
级别6标记：uidA，uidB，uidM
级别x标记：
我的新的DartDevice UidZ
设备安全列表
级别9标记：uidZ
被允许访问级别为7的设备后的旧的和新的
我的旧的DartDevice uidA
设备安全列表
级别9标记：uidA
级别8标记：
级别7标记：uidG，uidK，uidA，uidZ
级别6标记：uidZ，uidA，uidB，uidM
级别x标记：
我的新的DartDevice uidZ
设备安全列表
级别9标记：uidZ
级别7标记：uidG，uidK，uidA，uidZ

图 31

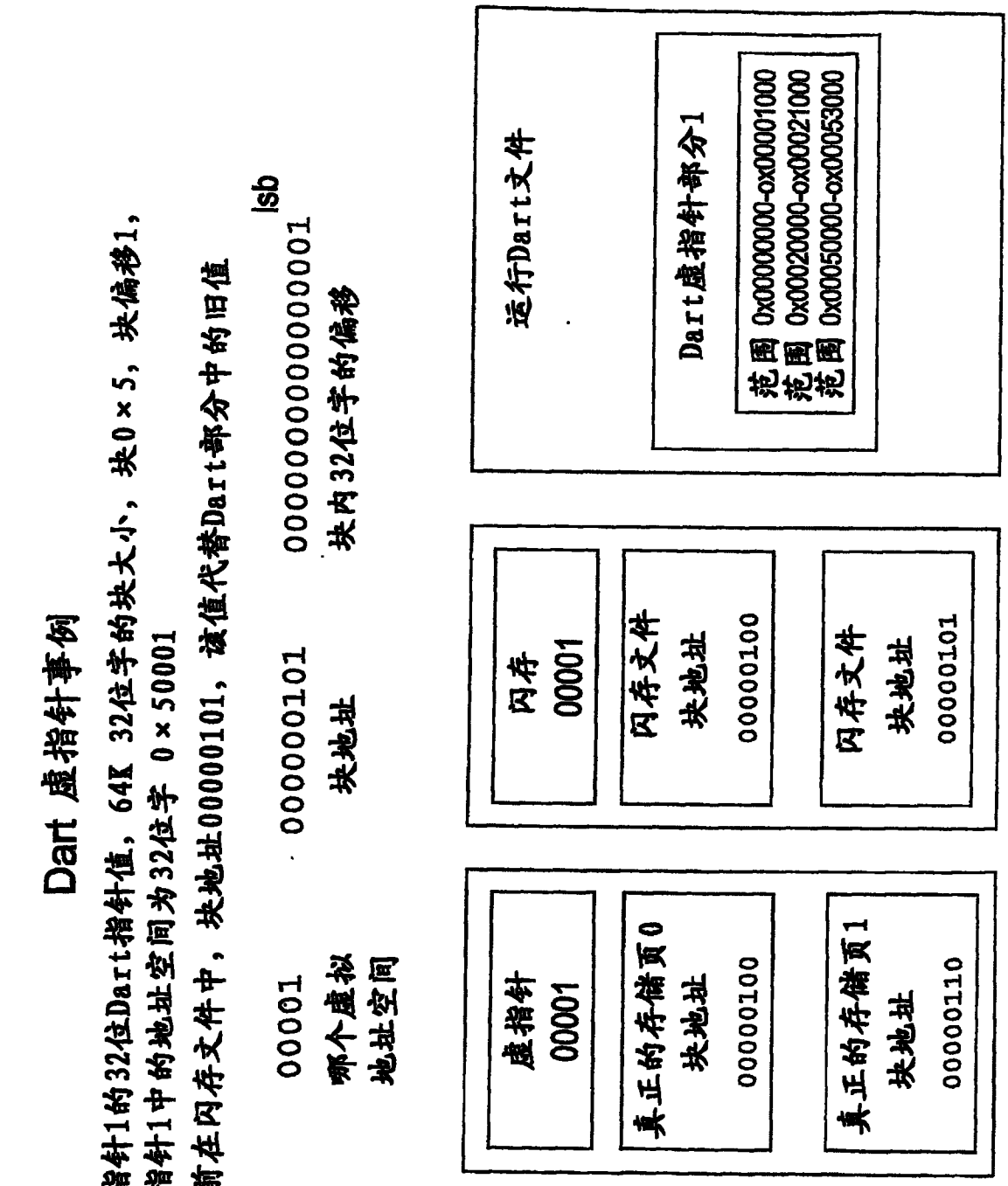

Dart 虚指针事例
用于虚指针1的32位Dart指针值，64K 32位字的块大小，块0×5，块偏移1，
虚指针1中的地址空间为32位字 0×50001
数据值目前在闪存文件中，块地址00000101，该值代替Dart部分中的旧值
lsb
01 00001 00000101 00000000000000001
指针类型 哪个虚拟 块地址 块内32位字的偏移
地址空间
00 数据/堆栈
01 虚指针
10 C++堆
11 共享的
虚指针 00001
真正的存储页0 块地址 00000100
真正的存储页1 块地址 00000110
闪存 00001
闪存文件 块地址 00000100
闪存文件 块地址 00000101
运行Dart文件
Dart虚指针部分1
范围 0x00000000-0x00001000
范围 0x00020000-0x00021000
范围 0x00050000-0x00053000

图 32