[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 9/44

[12] 发明专利申请公开说明书

[21] 申请号 02814191.1

[43] 公开日 2004 年 9 月 15 日

[11] 公开号 CN 1529847A

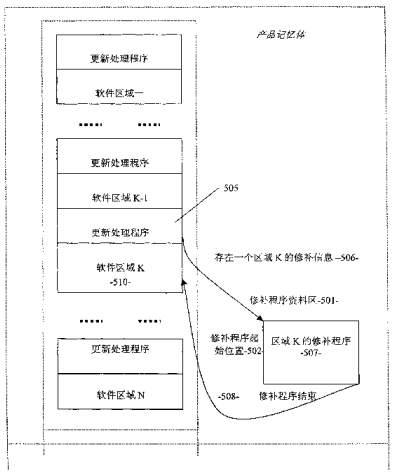
[22] 申请日 2002.7.15 [21] 申请号 02814191.1
[30] 优先权
[32] 2001. 7.16 [33] US [31] 60/305,704
[32] 2002. 2. 8 [33] US [31] 60/354,915
[86] 国际申请 PCT/US2002/022412 2002.7.15
[87] 国际公布 WO2003/009136 英 2003.1.30
[85] 进入国家阶段日期 2004.1.14
[71] 申请人 任宇清
地址 美国加利福尼亚圣地亚哥
[72] 发明人 任宇清
[74] 专利代理机构 北京北新智诚知识产权代理有限公司
代理人 赵郁军

权利要求书 10 页 说明书 24 页 附图 10 页

[54] 发明名称 内嵌软件更新系统

[57] 摘要

本发明提供了一个内嵌软件更新系统的系统及方法，本系统及方法可协助厂商或贩卖商避免因软件错误或硬件问题而导致的高成本的产品回收。本发明一方面展示一个错误更正系统，可以远端遥控数字产品，更正其软件错误并降低硬件问题对产品所造成的影响。另一方面展示一个软件更新系统，能够利用软件修补数据而将数字产品中的软件模块予以更新。本发明中的软件修补系统使制造商能够经由传送软件修补数据到一个正在使用的数字产品中进行软件错误的修补以及降低硬件问题的影响。该软件修补数据也可包含更新一数字产品的 NVM 存储器或可删除可编程只读存储器中的某些数据所需的参数，这对一个服务提供商或制造商在修改其服务项目内容或是产品功能时非常具有实用性。



ISSN1008-4274

知识产权出版社出版

1、一个调整运作于数字产品中的内嵌软件的方法，使其在调整后能够通过一软件修补程序来做就地的修改，其步骤为：

a) 根据一个事先定好的规则，在该内嵌软件内选定多个插入位置，以每一个选定的插入位置定义该内嵌软件的一个区域；

b) 在选定的多个插入位置上配置相应的更新处理程序，每一个更新处理程序都可被改写而得以修正该内嵌软件在相对应的区域的执行，这个改写与修正的动作发生在该区域存在一个软件修补程序时；

c) 在数字产品中决定一个记忆区来供该软件修补程序使用；

d) 将该配置有多个更新处理程序的内嵌软件载入该数字产品的一个内嵌软件程序码区；

2、权利要求 1 的方法中，供修补程序用的记忆区位于该内嵌软件的程序码区之内。

3、权利要求 1 的方法中，供修补程序用的记忆区位于该内嵌软件的程序码区之外。

4、权利要求 1 的方法中，决定插入位置的步骤是以至少一个下列的方法为基础：

基于该内嵌软件程序的程序行数；

基于该内嵌软件的程序功能单元；

基于该内嵌软件的指令类型；

基于该数字产品的软件设计方式；

基于该数字产品的硬件设计方式；

基于该内嵌软件的修改机制；

基于该内嵌软件的错误更正机制

5、权利要求 1 的方法中，配置多个更新处理程序的步骤是以至少一个下列的方法为基础：

a) 在将该内嵌软件编译为可执行的机器码前，将该更新处理程序配置在该内嵌软件中；

b) 将该内嵌软件编译为相对应的组合程序码后，将该更新处理程序配置于该组合程序码中，然后将该组合程序码编译为可执行的机器码；

c) 将该内嵌软件编译为相对应的目的码后，将该更新处理程序配置于该目的码中，然后将该目的码编译为可执行的机器码；

d) 将该内嵌软件编译为可执行的机器码后，将该更新处理程序配置在该可执行的机器码中。

6、权利要求 1 的方法中，配置多个更新处理程序的步骤进一步包含修改至少一个该内嵌软件的程序指令。

7、权利要求 5 的方法中，配置多个更新处理程序的步骤进一步包含修改至少一个该内嵌软件的程序指令。

8、权利要求 1 的方法中，至少一个所述的更新处理程序被改写而具备下面的功能：

如果存在一个用来更新该内嵌软件中某个区域的软件修补程序，将该内嵌软件的执行导向该软件修补程序所在的位置去执行。

如果没有这样的软件修补程序，执行该内嵌软件的该区域程序。

9、权利要求 4 的方法中，至少一个所述的更新处理程序被改写而具备下面的功能：

如果存在一个用来更新该内嵌软件中某个区域的软件修补程序，将该内嵌软件的执行导向该软件修补程序去执行。

如果没有这样的软件修补程序，执行该内嵌软件的该区域程序。

10、权利要求 1 的方法中，至少一个所述的更新处理程序被改写而具备下面的功能：

检查是否有一个用来更新该内嵌软件中一区域的软件修补程序，及

如果存在这样的软件修补程序，将该内嵌软件的执行导向该软件修补程序的位置去执行。

如果没有这样的软件修补程序，执行该内嵌软件的该区域程序。

11、权利要求 4 的方法中，至少一个所述的更处理程序被改写而具备下面的功能：

检查是否有一个用来更新该内嵌软件中一区域的软件修补程序，及

如果存在这样的软件修补程序，将该内嵌软件的执行导向该软件修补程序的位置去执行。

如果没有这样的软件修补程序，执行该内嵌软件的该区域程序。

12、权利要求 8 的方法中，将该内嵌软件的执行导向该软件修补程序所在位置的步骤，进一步包含：

将该内嵌软件的执行导向一个修补控制程序所在的的第一个位置，然后再根据该修补控制程序，将该内嵌软件的执行导向该软件修补程序所在的第二个位置，并执行该软件修补程序。

13、权利要求 10 的方法中，将该内嵌软件的执行导向该软件修补程序所在位置的步骤，进一步包含：

将该内嵌软件的执行导向一个修补控制程序所在的的第一个位置，然后再根据该修补控制程序，将该内嵌软件的执行导向该软件修补程序所在的第二个位置，并执行该软件修补程序。

14、权利要求 1 的方法中，多个更新处理程序包含下列至少一项：

第一个程序被改写成，如果存在一个用来更新该内嵌软件中某一区域的软件修补程序，将该内嵌软件的执行导向该软件修补程序所在的位置去执行，如果没有这样的软件修补程序，执行该内嵌软件的该区域；

第二个程序被改写成能够检查是否有一个用来更新该内嵌软件中一区域的软件修补程序，且如果存在这样的软件修补程序，将该内嵌软件的执行导向该软件修补程序所在的位置去执行，如果没有这样的软件修补程序，则执行该内嵌软件的该区域；

第三个程序被改写成能够检查是否有一个用来更新该内嵌软件中一区域的软件修补程序，且如果存在这样的软件修补程序，将该内嵌软件的执行导向一个修补控制程序所在的第一个位置，然后再根据该修补控制程序，将该内嵌软件的执行导向该软件修补程序所在的第二个位置去执行，如果没有这样的软件修补程序，则执行该内嵌软件的该区域；

第四个程序被改写成，如果存在一个用来更新该内嵌软件中一区域的软件修补程序，将该内嵌软件的执行导向一个修补控制程序所在的第一个位置，然后再根据该修补控制程序，将该内嵌软件的执行导向该软件修补程序所在的第二个位置去执行，如果没有这样的软件修补程序，则执行该内嵌软件的该区域。.

15、权利要求 1 的方法中，进一步包含：

e) 产生一个软件修补程序，使它提供一个预先决定的功能；

f) 通过一个通讯连接方法将该软件修补程序传到该数字产品中；

g) 由该数字产品接收该软件修补程序；

h) 将该软件修补程序写入该记忆区中。

16、权利要求 8 的方法中，进一步包含：

e) 产生一个软件修补程序，使它提供一个预先决定的功能；

f) 通过一个通讯连接方法将该软件修补程序传到该数字产品中；

g) 由该数字产品接收该软件修补程序；

i)将该软件修补程序写入该记忆区中。

17、权利要求 10 的方法中，进一步包含：

e) 产生一个软件修补程序，使它提供一个预先决定的功能；

f) 通过一个通讯连接方法将该软件修补程序传到该数字产品中；

g) 由该数字产品接收该软件修补程序；

h) 将该软件修补程序写入该记忆区中。

18、权利要求 12 的方法中，进一步包含：

e) 产生一个软件修补程序，使它提供一个预先决定的功能；

f) 通过一个通讯连接方法将该软件修补程序传到该数字产品中；

g) 由该数字产品接收该软件修补程序；

h) 将该软件修补程序写入该记忆区中。

19、权利要求 15 的方法中，进一步包含使用该软件修补程序来改写该内嵌软件中该更新处理程序的至少一部份的步骤。

20、权利要求 19 的方法中的该被改写的部分是一个预先决定的跳入位差。

21、权利要求 15 的方法中，产生软件修补程序的步骤包含使用一个修补服务器，这个修补服务器至少具备下列功能之一：

产生该软件修补程序；

以一个预先决定的加密演算法对该软件修补程序进行加密。

22、权利要求 15 的方法中，传输软件修补程序的步骤包含使用一个修补服务器，这个修补服务器至少具备下列功能之一：

传输该软件修补程序；

接收至少一个自该数字产品传来的信息；

处理该软件修补程序的重新传输。

23、权利要求 22 的方法中，进一步包含从该数字产品传送至少一个信息到该修补服务器，传送的动作发生在至少下列时间之一：

在该数字产品接收到该软件修补程序后；

在将该软件修补程序写入该记忆区之后。

24、权利要求 1 的方法中，该软件修补程序至少包含下列各项之一：

制造识别码

产品型号

软件版本号码

软件区域识别码

修补数据识别码

CPU 识别码

MAC 层地址

修补数据的地址信息

更新该内嵌软件的修补程序

有关如何转译修补程序的数据

重置旗标设定数据

可用作修补数据写入后的后续处理之用的数据

可用作识别一数字产品之用的数据

可用作识别一个用户之用的数据

用来更新该数字产品的数据区的数据参数

25、一个调整运作于数字产品中的内嵌软件的方法，使其在调整后能够藉由一个软件修补程序来做就地的修改，其步骤为：

a) 根据一个事先定好的规则，在该内嵌软件内选定多个插入位置，至少两个选定的插入位置定义该内嵌软件的两个区域；

b) 在该内嵌软件中配置多个修补处理区；

c) 设定该修补处理区为第一个内容，而第一个内容可以有选择性地被改为第二个内容

d) 将该配置有多个修补处理区的内嵌软件装入该数字产品中。

26.、权利要求 25 的方法中，配置的步骤包含：

将该修补处理区配置在该内嵌软件的相应的插入位置上。

27、权利要求 25 的方法中，进一步包含在内嵌软件的程序码区之外的一个预先决定的记忆区中，配置至少其中一个修补处理区。

28、权利要求 26 的方法中，进一步包含在内嵌软件的程序码区之外的一个预先决定的记忆区中，配置至少其中一个修补处理区。

29、权利要求 25 的方法中，配置的步骤包含：

以在该内嵌软件中写入程序指令的方式保留记忆区以作为该修补处理区

30、权利要求 26 的方法中，配置的步骤包含：

以在该内嵌软件中写入程序指令的方式保留记忆区以作为该修补处理区。

31、权利要求 27 的方法中，配置的步骤包含：

以在该内嵌软件中写入程序指令的方式保留记忆区以作为该修补处理区。

32、权利要求 25 的方法中，决定插入位置的步骤是以至少一个下列的方法为基础：

基于该内嵌软件程序的程序行数

基于该内嵌软件的程序功能单元

基于该内嵌软件的指令类型

基于该数字产品的一个已预先决定的软件设计方式

基于该数字产品的一个已预先决定的硬件设计方式

基于该内嵌软件的一个已预先决定的修改机制

基于该内嵌软件的一个已预先决定的错误更正机制

33、权利要求 25 的方法中，配置多个修补处理区的步骤进一步包含修改至少一个该内嵌软件的程序指令。

34、权利要求 25 的方法中，所述之修补处理区包含下列至少一项：

第一个更新处理程序被改写成，如果存在一个用来更新该内嵌软件中某一区域的软件修补程序，将该内嵌软件的执行导向该软件修补程序所在的位置去执行，如果没有这样的软件修补程序，执行该内嵌软件的该区域；

第二个更新处理程序被改写成能够检查是否有一个用来更新该内嵌软件中一区域的软件修补程序，且如果存在这样的软件修补程序，将该内嵌软件的执行导向该软件修补程序所在的位置去执行，如果没有这样的软件修补程序，则执行该内嵌软件的该区域；

第三个更新处理程序被改写成能够检查是否有一个用来更新该内嵌软件中一区域的软件修补程序，且如果存在这样的软件修补程序，将该内嵌软件的执行导向一个修补控制程序所在的第一个位置，然后再根据该修补控制程序，将该内嵌软件的执行导向该软件修补程序所在的第二个位置去执行；如果没有这样的修补程序，则执行该内嵌软件的该区域；

第四个更新处理程序被改写成，如果存在一个用来更新该内嵌软件中一区域的软件修补程序，将该内嵌软件的执行导向一个修补控制程序所在的第一个位置，然后再根据该修补控制程序，将该内嵌软件的执行导向该软件修补程序所在的第二个位置去执行；如果没有这样的软件修补程序，执行该内嵌软件的该区域；

第五个更新处理程序被改写成能够检查是否存在一个软件修补程序，如果有，则修改该内嵌软件中相对应区域的执行，使其跳到该软件修补程序去执行；

一个 jump 指令被改写成，如果该软件修补程序存在，则修改该 jump 指令的位差使该内嵌软件中相对应修补程序的区域得以执行，如果该软件修补程序不存在，则跳到该区域；

一个被该 jump 指令所用的 jump 位差值，可有选择性地被改成另一个 jump 位差值；

一个修补控制程序被改写成能够造成跳入到该软件修补程序；

一个修补程序被改写而能修改该内嵌软件的执行；

一个修补控制区块有选择地被变更为一个修补更正程序；

第一个记忆区被设定成预先决定的内容，用来修改该内嵌软件的执行；

第二个记忆区被设定成预先决定的内容，可有选择性地被变更为用来修改该内嵌软件之执行的参数；

第三个记忆区被设定成预先决定的内容，可有选择性地被变更为用来修改该内嵌软件之执行的软件程序。

35、权利要求 34 的方法中，该修补控制区块是由该内嵌软件中的多个区域所共用。

36、权利要求 25 的方法中，多个修补处理区的配置时机，是根据以下至少一种方法：

a ）在将该内嵌软件编译为可执行的机器码之前，配置至少一个修补处理区到该内嵌软件中；

b) 将该内嵌软件编译为相对应的组合程序码后，配置至少一个所述的修补处理区到组合程序码中，然后将该组合程序码编译为可执行的机器码；

c) 将该内嵌软件编译为相对应的目的码后，在该目的码中配置至少一个所述的修补处理区，然后将该目的码编译为可执行的机器码；

d) 将该内嵌软件编译为可执行的机器码后，在该可执行的机器码中配置至少一个所述的修补处理区。

37、权利要求 25 的方法中，进一步包含：

e) 产生一个软件修补程序，使它提供一个预先决定的功能；

f) 通过一个通讯连接方法将该软件修补程序传到该数字产品中；

g) 由该数字产品接收该软件修补程序；

h) 将该软件修补程序写入至少一个所述的修补处理区中。

38、权利要求 27 的方法中，进一步包含：

e) 产生一个软件修补程序，使它提供一个预先决定的功能；

f) 通过一个通讯连接方法将该软件修补程序传到该数字产品中；

g) 由该数字产品接收该软件修补程序；

h) 将该软件修补程序写入至少一个所述的修补处理区中。

39、权利要求 28 的方法中，进一步包含：

e) 产生一个软件修补程序，使它提供一个预先决定的功能；

f) 通过一个通讯连接方法将该软件修补程序传到该数字产品中；

g) 由该数字产品接收该软件修补程序；

h) 将该软件修补程序写入至少一个所述的修补处理区中。

40、权利要求 29 的方法中，进一步包含：

e) 产生一个软件修补程序，使它提供一个预先决定的功能；

f) 通过一个通讯连接方法将该软件修补程序传到该数字产品中；

g) 由该数字产品接收该软件修补程序；

h) 将该软件修补程序写入至少一个所述的修补处理区中。

41、权利要求 34 的方法中，进一步包含：

e) 产生一个软件修补程序，使它提供一个预先决定的功能；

f) 通过一个通讯连接方法将该软件修补程序传到该数字产品中；

g) 由该数字产品接收该软件修补程序；

h) 将该软件修补程序写入至少一个所述的修补处理区中。

42、权利要求 37 的方法中，该写入的步骤包含使用该软件修补程序来改变至少一个 jump 位差值的步骤。

43、权利要求 37 的方法中，产生软件修补程序的步骤包含使用一修补服务器，这个修补服务器至少具备下列功能之一：

产生该软件修补程序；

以一个预先决定的加密演算法对该软件修补程序进行加密。

44、权利要求 37 的方法中，传输软件修补程序的步骤包含使用一个修补服务器，这个修补服务器至少具备下列功能之一：

传输该软件修补程序；

接收至少一个自该数字产品传来的信息；

处理该软件修补程序的重新传输。

45、权利要求 37 的方法中，传输的步骤包含使用 CDMA 标准中所定义的 Data Burst Message 来传送该软件修补程序。

46、权利要求 44 的方法中，进一步包含从该数字产品传送至少一个信息到该修补服务器，传送的动作发生在至少下列情形之一：

在该数字产品接收到该软件修补程序之后；

在将该软件修补程序写入该修补处理区之后。

47、权利要求 37 的方法中，该软件修补程序至少包含下列数据之一：

制造识别码

产品型号

软件版本号码

软件区域识别码

修补数据识别码

CPU 识别码

MAC 层地址

修补数据的地址数据

为更新该内嵌软件的修补程序

有关如何转译修补程序的数据

重置旗标设定数据

可供作修补数据写入后之后续处理之用的数据

可供作识别一数字产品之用的数据

可供作识别一个用户之用的数据

用来更新该数字产品的数据区的数据参数

48、权利要求 37 的方法中，进一步包含由该数字产品以下列的至少一种方法来转译该软件修补程序的至少一个项目：

在将该软件修补程序写入到该修补处理区之前，将所述的至少一个项目从第一个程序语格式变更为第二个程序语言格式。

在使用该软件修补程序来执行之前，将所述的至少一个项目从第一个程序语言格式变更为第二个程序语言格式。

在将该软件修补程序写入到该修补处理区之前，将所述的至少一个项目从第一个数据格式变更为第二个数据格式。

在使用该软件修补程序来执行之前，将所述的至少一个项目从第一个数据格式变更为第二个数据格式。

49、权利要求 37 的方法中，将该软件修补程序写入该修补处理区，是由至少一个能够将数据写入 FLASH 存储器中的 FLASH 数据写入的程序来执行的。

50、权利要求 49 的方法中，写入软件修补程序的步骤，进一步包含在将数据写入 FLASH 之前，先把该 FLASH 数据写入的程序放入 RAM 存储器中。

51、一个修改含有 FLASH 存储器及 non-volatile 存储器 (NVM)，且由一内嵌软件来处理数字产品的方法，其步骤为：

a) 产生一个软件修补程序，该软件修补程序包含至少一个用来更新 FLASH 存储器内容的修补程序，及至少一个用来更新 NVM 内容的数据参数

b) 通过一个无线通讯连接方式将该软件修补程序传到该数字产品中

c) 由该数字产品接收该软件修补程序

d) 将该软件修补程序写入该数字产品中

52、权利要求 51 的方法中，写入的步骤包含至少下列步骤之一：

将至少一个数据参数以改写该 NVM 中既有数据的方法写入该 NVM 存储器中；

将所述的至少一个修补程序写入该 FLASH 存储器中。

53、权利要求 52 的方法中，进一步包含至少下列步骤之一：

将该软件修补程序的识别数据写入该内嵌软件的修补控制表中；

将该软件修补程序的修补控制数据写入该内嵌软件的修补控制表中。

54、一个电脑可读取的媒介，其中包含用以更新数字产品中内嵌软件的指令，它们可以是作为下列用途的指令：

执行该内嵌软件；

决定一个可供该内嵌软件中一区域使用的修补程序是否存在；

如果该修补程序可供该区域使用，将该内嵌软件在该区域的执行导向该修补程序，

在执行该修补程序后，将执行导回到该内嵌软件中一个预先决定的位置。

55、权利要求 54 中的电脑可读取之媒介，其中该内嵌软件配置了多个被改写而能够判定是否有可用之修补程序的程序。

56、权利要求 54 中的电脑可读取之媒介，其中该预先决定的位置为需被更新的部份程序的末端。

57、权利要求 55 中的电脑可读取之媒介，进一步包含作为下列用途的指令：

通过一个通讯连接方法来接收一个修补程序；

将该修补程序写入该数字产品中。

内嵌软件更新系统

相关申请

本专利申请要求以下临时申请的优先权：申请日为 2001 年 7 月 16 日，发明名称为“内嵌软件修补系统”，申请号 60/305,704；申请日为 2002 年 2 月 8 日，发明名称为“内嵌软件修补系统”，申请号 60/354,915。本专利申请将前述两个申请案列为参考文献，视同将其内容完全结合于本申请案中。

技术领域

本发明可应用于具备内嵌软件处理系统的数字电子产品，更具体地说，是在一数字产品上市销售前或上市后，对该产品之中的内嵌软件进行更新、更正、修改或升级的系统及方法。

背景技术

数字科技的发展与革新充分地改变人们的生活、工作与休闲。日益轻薄短小的电脑、手机、个人数字助理、携带型传呼器、PCMCIA 无线数据卡，或其他使用公用标准或专属 I/O 连接器的无线数据卡是科技成为人们越来越不可或缺的最好例子。这些数字科技帮助人们更有效率地工作、与亲属保持联系及获取更多的数据及娱乐。

手机、个人数字助理(PDA)及机顶盒 (set-top boxes) 等数字产品都使用大规模的内嵌软件系统。这些产品的内嵌软件系统通常都拥有几万行的原始程序码。不可避免的，这些内嵌软件会出现错误，而这些错误将造成数字产品的故障。有些致命的严重错误甚至迫使厂商必须回收他们的产品。由于产品回收所造成的财务损失，一些厂商不得不决定避开某些特定的商业市场，或至少等到产品的品质及可靠度得到改善为止。当然，这些做法所导致的厂商公共关系的恶果及消费者信心的丧失，也是厂商做决定时的重要考量因素。

尽管数字产品的研发与制造不断地改善，产品缺陷仍不可避免。而且越先进复杂的数字产品越容易产生软件、硬件、材料或生产上的错误或缺陷。厂商可使用许多预防措施，以在产品销售至市场之前便尽可能降低或检测出缺陷。然而，一旦这些产品流入市场，尝试修理这些数字产品的努力将有可能成为产品回收的恶果。若仅有少量的产品出现缺陷，贩卖商可提供消费者维修或更换的服务，但唯有当如此的不便很少发生或贩卖商于维修期间内自行提供替代产品时，消费者才可能忍受如此的不便。

当产品出现缺陷的数量很大且牵涉多批产品时，贩卖商仍有可能提供消费者维修或更换的服务。但是对于如此庞大的数量，贩卖商可能根本没有足够的资源来及时修复或更换这些有错误的产品。虽然贩卖商可于维修的期间内提供替代产品，但它必须先有足

够的库存替代产品来提供给消费者暂时使用。而这种提供暂时使用的方式也导致所涉及的存货丧失营收的机会，因为贩卖商将无法以正常的方式将它们销售获利。

此外，即使在数字产品已经销售到市场后，厂商有时仍需要对该产品进行改善、加强或升级，因而可能需要更改该产品的内嵌软件。若为重大改变时，厂商可能决定发行一新版的产品，这样的决定可能是昂贵的，但却是必须的。而对于小幅度的更改，厂商将面临是否提早发行新版或等到累积许多小幅更改之后再发行的抉择。产品过早的发行或是不适时的发行将耗费公司的资源，但若等到有较多改变时再发行，将背负引起消费者抱怨的风险。

所以，若能减少这些具备内嵌软件系统的数字产品的回收，则为较符合厂商需求的作法。

若能减少这些具备内嵌软件系统的数字产品过早与不适时的发行，亦为较符合厂商需求的作法。

若能以修理、修补、更新、升级或加强的方式来减少这些数字产品发行的次数，亦为较符合厂商需求的作法。

发明内容

本发明的目的是提供一个内嵌软件更新系统的系统及方法，本系统及方法可协助厂商或贩卖商避免因软件错误 (“bugs”) 或硬件问题而导致的高成本的产品回收。本发明一方面展示一个错误更正系统 (Error correction system)，可以远端地 (remotely) 遥控数字产品，更正软件的错误与降低硬件问题对产品所造成的影响。另一方面展示一个软件更新系统，能够利用软件修补数据 (Software patches) 而将数字产品中的软件模块予以更新。本发明中的软件修补系统使制造商能够经由传送软件修补数据到一个正在使用的数字产品中以进行软件的修补以及降低硬件问题的影响。该软件修补数据也可包含用以更新一数字产品的 NVM 存储器或可删除可编程只读存储器 (EEPROM, Electrically Erasable Programmable Read-Only Memory) 中的某些数据所需的参数，这对一个服务提供商或制造商在修改其服务项目内容或是产品功能时是非常具有实用性的。

本发明的软件更新系统具有传统软件升级/更新系统所不具备的优点。相比较之下，本发明的软件更新系统仅需要很小的存储器空间来储存用以修复错误或更新关键模块的软件修补数据。这项优点对于小且轻便的数字产品而言更为显著，它不像传统的软件升级/更新系统那样需要庞大的存储器空间来容纳整个应用软件数据以进行新版本软件的升级。

由于本发明的软件更新系统的每个修补数据的大小一般都很小，它仅仅需要有限的网络资源来做修补程序的传输。相对而言，传统的软件升级/更新系统则需要很宽的频宽或很大量的网络资源来传输大量的应用软件数据以进行新版本软件的更新。

由于数字科技的发展日新月异，新的标准、规格与服务项目升级的需求也比以往来得更加频繁。这类升级通常需要新的数字硬件设备来支持新的先进项目。

本发明的软件更新系统的另一方面用途在于厂商可以经由软件更新，以较简便且经济的方式来进行产品升级，如此也尽可能地减少了更改硬件的次数。此外，本发明的软件更新系统可被用来修复隐藏于操作系统中的问题，这些操作系统可能用于支持应用程序的下载，比如 JAVA Virtual Machines 或 BREW (Binary Runtime Environment for Wireless)。

本发明将数字产品之内嵌软件划分为多个不同的区域，并于更新该内嵌软件时，只对必要的区域做更新的动作。本发明所采用的作法相对于传统作法中必须更新整个软件系统的方式是具有优势的，因为本发明所采用的作法对于网络传输所需的资源能做到最佳化的运用。

在设计中央处理器 (CPU)，微处理器单元 (MCU)，或数字信号处理器 (DSP) 时也可加入一种机制，这种机制能于执行软件区域之前先依本发明检查软件更新数据是否存在。中央处理单元/微处理器单元/数字信号处理器可被设计成能够在内嵌软件执行时自动检测软件更新数据并跳至更新程序码所在的地址。

附图说明

图 1：显示数字产品中典型的微处理器系统简图

图 2：显示数字产品中典型传统数字信号处理器系统简图

图 3：显示数字产品中修补系统的软件结构范例

图 4：显示数字产品中原始内嵌软件的区域划分及更新处理程序配置的范例简图

图 5：显示一个与本发明的应用有关，以程序区域为基础来执行软件更新的设计与实施范例

图 6：显示另一个与本发明的应用有关，经由修补控制程序，使用 JUMP 来跳入修补程序以执行软件更新的设计及实施范例

图 7：显示另一个与本发明的应用有关，经由修补控制区块，使用 JUMP 来跳入修补程序以执行软件更新的设计及实施范例

图 8：显示一个与本发明的应用有关，用以传送修补数据之通讯协定层范例

图 9：显示一个与本发明的应用有关的修补程序与修补程序数据包格式的范例

图 10：显示一个与本发明的应用有关，将修补数据写入存储器的范例图

图 11：显示一个与本发明的应用有关的修补服务器 结构的范例

图 12：显示使用短信信息来传送修补数据的设计范例

较佳实施例的具体说明

1、本专利申请展示了一种用于数字产品中的内嵌软件更新系统的方法与系统。在下述的详尽说明中，许多特定细节是为了让读者对于本发明有一个完整的了解。然而很

明显的，对一位已具备本领域一般技术水准的人来说，无须这些特定的细节描述就得以实施本发明。此外，众所周知的结构与技术内容将不会在以下详细说明，以避免对本发明内容产生不必要的混淆。并且下列内容所使用的数字标题仅系为了描述各自主题上及参阅上的方便而设。必须特别说明的是此处所说的“内嵌软件”是指为了使数字产品运作而以 C/C++/Java/Assembler 或其他软件程序语言所写成的软件，它同样的也可以用任何 CPU/Microprocessor 或是 DSP (Digital Signal Processor) 的可执行机器码来撰写。

2. 系统结构

首先先从系统结构方面来说明本发明中的软件更新系统。参考图 1，它显示一个简化的典型微处理器系统。在一个典型数字产品的微处理器系统中会包含一个中央处理器 (CPU) 100，随机存取存储器(RAM) 110， FLASH/ROM 存储器 115，以及一些周边装置 120。在执行程序时 CPU100 可读取位于 FLASH/ROM 存储器 115 中的软件程序。FLASH 为一种持续有电的不变性存储器，它可以被抹除以及再编程 ，但必须以一个区块 (blocks) 为单位。ROM 为“只读存储器“。

类似于图 1 所示的微处理器系统，图 2 为传统数字产品中的数字信号处理器 (DSP) 系统，其中可包含一个 DSP 核心 230，随机存取存储器 (RAM) 240，FLASH/ROM 存储器 245，以及一些周边装置 250。在执行程序时 DSP 核心 230 可读取位于 FLASH/ROM 存储器 245 中的软件程序。

一个典型的数字产品，例如一个数字行动电话，通常包含一个微处理器系统，并也可能包含一个 DSP 系统。本发明的软件更新系统可以被配置在数字产品的微处理器系统及/或 DSP 系统里的内嵌软件中。根据本发明所实施的软件更新系统被用在一个数字产品中时，如图 3 所示，它可包含一个用来接收修补数据的软件模块 302，一个用来将修补数据写入存储器的软件模块 304，一个储存修补程序的数据库 310，以及一个用来选用修补程序的修补更正处理器 320。图 3 即显示一个简化的数字产品修补系统的软件结构范例。

参照图 3，修补数据接收模块 302 是用以接收软件修补程序。它可包含采用有线连结或无线连结的数据接收机制，和数据查错机制及/或数据安全检查机制。在修补数据接收模块 302 正确地收到修补程序后，它将会将此修补程序传递给修补数据写入模块 304。这个修补数据写入模块 304 是用来将修补程序写入修补资料库 310 中。它会包含一个可将数据写入 FLASH, NVM, EEPROM 存储器及/或其他类型存储器中的程序。修补资料库 310 是 FLASH 存储器中用来存放修补程序的一块记忆区。而修补更正处理器 320 则是负责让软件在执行时使用修补程序而不再使用原来的错误程序。它可包含一个机制，能够让软件在执行时从原含有错误的程序码区跳入修补程序数据区。

3. 软件更新技术

3.1 将软件划分为多个软件区域

为了获得本发明应用在软件更新系统时所具备的优势，在将一个内嵌软件安装到数字产品前，首先必须在软件中选定多个位置，做为以后软件更新的起点。这些位置依照地址顺序由小到大排列，可以用 L1，L2，...，Ln 来表示，其中 n 是这些位置的总数。

位于两个连续位置之间的部分 (如 L1 与 L2 之间)，我们把它定义成内嵌软件中的一个软件区域 (如 S1)。如此一来，这一内嵌软件实际上就被分割为多个软件区域，依照地址顺序由小到大排列即 S1，S2，...，Sn，其中 n 是软件区域的总数。最后一个区域 Sn 是从 Ln 开始一直到这个内嵌软件结束为止。

如同任何其他软件，内嵌软件需要不时地加以更新或修改。比方说，当需要更新区域 S2 中的程序码时，可由位置 L2 开始将程序的执行由软件区域 S2 导向软件修补程序。于是这个修补程序在软件执行时被用来取代原 S2 中的程序码。在某些情况下，从其他的位置 (例如 L1) 开始做更新可能会更方便，此时内嵌软件的执行将由 L1 开始被导向修补程序，虽然需要被更新的程序码发生在 S2 中。换言之，在某些情况下，从较前面的区域开始将软件的执行导向修补程序是比较好的。

软件区域的大小未必是固定的，它可依据各个区域不同的需求而大小各异。决定 L1，L2 "，Ln 的位置并依此将内嵌软件区分为软件区域 S1，S2 "，Sn 的方式可能基于以下的标准，如软件程序功能的界限，软件程序的行数，CPU/DSP 的指令类型，软件的设计方式，硬件的设计方式，软件修改的需求，或是错误更正的需求等可被熟悉本领域的人所认同的方式。

在每一个位置 L1，L2, ...，Ln 置入一个更新处理程序来处理软件的更新。参考图 4，显示一个简化的软件区域分隔及更新处理程序配置的图解范例。为了调整内嵌软件 400，我们在软件 400 中每一个区域 401，402，403 一开始的地方置入一个更新处理程序 404，405，406。

"更新处理程序"为一个负责软件更新处理的程序。如果存在一个用来更新某软件区域 (例如 S2) 中一些程序码的软件修补数据，位于 L2 的更新处理程序会将 CPU/DSP 的执行导向软件修补程序开始的地址，并执行那段修补程序。这个更新处理程序在指示程序跳入修补程序的起始地址之前可能会牵涉到使用一个"修补控制程序"，我们将在图 6 做进一步说明。如果用来更新区域 S2 的修补数据不存在，位在 L2 的更新处理程序将指示 CPU/DSP 执行现有的 S2 中的程序；也就是说，CPU/DSP 将会在不影响正常执行的状况下继续执行 S2 的程序。

更新处理程序的内容可以依照位置的不同而有所不同，也可以全部设计为相同内容。

3.2 以软件区域为基础的软件更新

图 5 显示一个以软件区域为基础来做软件更新的设计与实施范例。当存在一个用来更新软件区域 K 510 的软件修补数据时，区域 K 中的更新处理程序 505 会在软件执行时将 CPU/DSP 执行的地址跳 506 到修补程序 507 的起始地址 502 来更新软件区域 K。这样一来，修补程序 507 将取代区域 K 中的原始程序而被执行。在执行完这个修补改程序 507 后，CPU/DSP 的指令指针会被指示跳 508 到一个事先决定的原始程序中的位置。这个位置通常就紧接在区域 510 中需要被更新的程序码之后。如此一来，区域 510 中的任何错误将因修补程序的执行而被绕过。

图 6 显示另一个以软件区域为基础来做软件更新的设计及实施范例。当存在一个用来更新软件区域 K 610 的软件修补数据时，区域 K 中的更新处理程序 605 会指示 CPU/DSP 将其执行跳 606 到一个修补控制程序 609。修补控制程序 609 是一个可以控制一般修补程序执行的程序，比如修补程序执行前的准备程序及/或根据如一个“修补控制表” (将说明于下) 里的修补状态及系统参数来决定修补程序 607 的起始地址 602。这个修补控制程序 609 指示 CPU/DSP 将其执行跳 611 到区域 K 610 的修补程序 607 的起始地址 602。这样一来，修补程序 607 将取代区域 K 610 中的原始程序而被执行。在执行完毕这个修补程序后，CPU/DSP 的指令指针会被指示跳 608 到一个事先决定的原始程序中的位置，这个位置通常就紧接在需要被更新的程序码之后。

前述的修补控制表是一个存有用来控制修补数据之接收，写入存储器以及修补信号处理的数据表。这个数据表可以包含已被写入数字产品的修补数据的识别码所形成的表格，及/或被这些修补数据更新过的对应区域的识别码所形成的表格。它也可以包含一个修补资料库的数据，比如可用于储存新修补数据的修补程序数据区的字节数。它还可能包含其他用来控制修补数据之接收、写入存储器，以及信号处理的数据。这个修补控制表可以位于修补控制程序中或数字产品存储器 600 中的其他任何地方。

使用如修补控制程序 609 这样的中间步骤有其优点，是会被熟悉本领域的人所认同的。软件的修补会经由这个中间步骤来指示 CPU/DSP 的指令指针跳到如图 6 所示之修补程序 607 的起始地址 602，而不直接跳到如图 5 所示的修补程序 507 的起始地址 502。如果有必要对一个已有修补数据的区域进行新的修补时，可以将对应于既有修补数据的区域识别码 从修补控制程序 609 所负责的修补控制表中删除 (例如设定成零)，而对于新的修补数据，可将原来的区域识别码配合新的修补数据识别码及/或修补程序起始地址，写入修补控制表中。如此一来，修补控制程序 609 将可以从修补控制表的内容来决定新的修补程序的起始地址，并指示 CPU/DSP 的指令指针跳到这个新的修补程序。

用来储存修补程序码的记忆区可以被安排在内嵌软件程序码范围以外的区域，也可以被放在内嵌软件程序码范围以内的地方。例如，FLASH 存储器内的一个记忆区可以被

保留并填满十六进位 "0xFF "以做为数据区或数据缓冲区，这个记忆区可以被放在内嵌软件程序码的范围内，经由更改或覆写这些十六进位 "0xFF "成为修补程序码来存放修补程序。

3.3 设计更新处理程序

以下章节将讨论更新处理程序的相关设计与实施方式。

3.3.1 使用修补检查程序来设计更新处理程序

使用修补旗标或修补检查程序是设计与实施更新处理程序的一个例子。用这样的方法，每一个软件区域将有一个对应的修补旗标及/或一个修补检查程序来标示这一区域是否有修补数据存在于修补程序数据区中。

当修补程序数据区中不存在对应于某区域的修补数据时，其更新处理程序可以用程序描述语言表示如下，这段程序的描述可以直接被翻译成 C/C++Java/Assembler 及其他程序语言或是 CPU/微处理器及 DSP 的可执行机器码：

```
(at the beginning of a software section.)
Patch-Checking-Routine (修补检查程序)
If (the Patch-Checking-Routine declares that there is a patch for updating this section)
{
Patch-Correction-Routine (修补更正程序)
}
ORIGINAL SECTION CODE.
```

在以上的程序中使用了一个修补检查程序。这个修补检查程序是一个用来检查修补程序数据区中有无更新这个区域的修补程序的软件程序。它可以检查某些修补旗标或系统参数，或使用某些软件程序来得知有无用来更新这个区域的修补程序。

修补更正程序是一个用来指示 CPU/微处理器或是 DSP 使用修补程序码而不用原区域程序码的软件程序。当有必要为跳入修补程序做准备动作时，修补更正程序可以执行比如储存一些参数或设定一些参数及程序等的前置处理。修补更正程序也包含一些指示 CPU/微处理器或 DSP 指令指针的指令，可以如图 5 将指针指向修补程序的起始地址，或如图 6 指向一个修补控制程序。使用修补控制程序的优点已在 3.2 中说明。

3.3.2 使用 Jump/Branch 指令来设计更新处理程序

另一个设计与实施更新处理程序的例子是使用 JUMP (或 ARM®语法中的 BRANCH) 指令或其他类似的 CPU/DSP 指令。请注意 JUMP 这个于程序设计中众所周知的概念在

本申请案中是被当成一般的用词，代表将 CPU/DSP 的执行从一个位置导向另一个位置。

当修补程序数据区中不存在对应于某区域的修补数据时，其更新处理程序可以用程序描述语言表示如下，这段程序的描述可以直接被翻译为 C/C++/Java/Assembler 及其他程序语言或是 CPU/DSP 的可执行机器码：

```
(At the beginning of a software section, in case of no patch programmed)
    Jump to Label_O
Label_P:  Patch-Correction-Routine (修补更正程序)
Label_O:  ORIGINAL SECTION CODE
```

当修补程序数据区中不存在对应于某区域的修补程序时，更新处理程序的第一个指令为 JUMP 到 Label_O，也就是 JUMP 到这个区域原程序码的起始地址。

当修补程序数据区中有一个对应此区域的修补程序时，更新处理程序的第一个指令将会被更改，使得 CPU/DSP 指令指针指到 Label_P 而非 Label_O。这个更新处理程序可以用程序描述语言表示如下，这段程序的描述可以直接被翻译成 C/C++/Java/Assembler 及其他程序语言，或是 CPU/微处理器及 DSP 的可执行机器码：

```
(At the beginning of a software section, in case that there is a patch for updating this section)
    Jump to Label_P
Label_P:  Patch-Correction-Routine (修补更正程序)
Label_O:  ORIGINAL SOFTWARE CODE
```

在此情况下，CPU/微处理器或是 DSP 的指令指针将被导向修补更正程序。修补更正程序的设计方式已在之前的 3.3.1 中说明过。必需注意的是 FLASH 存储器的内容只允许被从 "1 "改成 "0 "，从 "0 "改成 "1 " 是不被允许的。因此，把 jump 的位差 (offset) 初始值设定成十六进位 "0xFFF" 可以在进行软件更新时用来减少位差。

3.3.3 使用 Direct Jump/Branch 指令来设计更新处理程序

以下我们将说明另一个将本发明应用在更新处理程序之设计与实施的范例。当修补程序数据区中不存在对应于某个区域的修补数据时，其更新处理程序可以用程序描述语言表示如下，这段程序的描述可以直接被翻译为 C/C++/Java/Assembler 及其它程序语言，或是 CPU/微处理器及 DSP 的可执行机器码：

(At the beginning of a software section in case of no patch programmed)

Jump to Label_O

Label_O: ORIGINAL SECTION CODE

当修补程序数据区中不存在对应于某区域的修补程序时，其更新处理程序的第一个指令且是唯一的一个指令就是 JUMP 到 Label_O，而 Label_O 就是本区域原程序码的起始地址。

当修补程序数据区中有一个对应此区域的修补程序时，更新处理程序的第一个并且唯一的指令将会被更改，使得 CPU/DSP 的执行被导向一个修补更正程序的起始地址，或是修补程序的起始地址。这个更新处理程序可以用程序描述语言表示如下，该程序的描述可以直接被翻译为 C/C++/Java/Assembler 及其它程序语言，或是 CPU/微处理器及 DSP 的可执行机器码：

(At the beginning of a software section, in case that there is a patch for updating this section)

Jump to Patch-Correction-Routine (or Patch Start Address)

Label_O: ORIGINAL SECION CODE

修补更正程序的设计方式已在之前的 3.3.1 中说明过。

以下我们说明一个使用 ARM® (Advanced RISC Machines) CPU 汇编语言的例子，这个例子是特别针对使用 32 比特指令组的 ARM 模式而做的。

(At the beginning of a software section, in case of no patch programmed)

B Label_O

Label_O: ORIGINAL SECTION CODE

在此范例中，ARM CPU 的 Branch 指令"B"被用来当作更新处理程序。因为它跳到下一行，它的位差应是"0xFFFFFF"。由于在将数据写入 FLASH 存储器时一个比特可以从"1"被改成"0"，这个位差值 "0xFFFFFF"可轻易地被变更为任何其他 24 比特值。

在收到用来更新此区域的修补程序后，修补程序就会被写入修补程序数据区中。这个 "B "指令的位差也将从原本对应于 Label_O 的 "0xFFFFFF "被改为对应到修补程序的位差值。

```
(At the beginning of a software section, in case that there is a patch
for updating this section)
    B Patch_Start_Address
Label_O: ORIGINAL SECTION CODE
```

3.4 以修补处理区为基础之软件更新

本节将介绍在软件中配置或保留用来操作修补之修补处理区的技术。修补处理区是一个可用以操作修补程序的记忆区或数据区块。举例来说，它可以是包含前述的更新处理程序的记忆区，储存 JUMP 指令来指示 CPU/DSP 指令指指针跳到修补程序起始地址的记忆区、储存修补控制程序的记忆区、以及包含修补程序的记忆区。

类似于 3.1 节，首先应先决定地址由小到大依序为 L1，L2，“，Ln 的一组位置。如此一来，这一内嵌软件实际上就被分隔为多个软件区域，即 S1, S2, ..., Sn。我们可在 L1,L2, ..., Ln 中的每一个位置，或至少某些或大部份位置，配置或保留一个修补处理区。某些修补处理区也可以被插入或保留在内嵌软件的最尾端，或在内嵌软件程序码范围以外的一个记忆区域内。

一种实施方式是直接在原来的内嵌软件中保留一些记忆区，并用这些记忆区作为更新内嵌软件用的修补处理区。在这样的情况下，上述所使用的“配置”一词将会更近似于“保留”的意思，就是使用内嵌软件的某些指令来进行记忆区的保留。

当定义了一个修补处理区并且把它用做更新处理程序时，它可以如上一节所示地被放在内嵌软件的每一个软件区域中。

以下我们用一个例子来说明如何在汇编语言程序中配置或保留修补处理区。在汇编语言程序中，我们通常可以在汇编语言指令中宣告某些数据区并且利用这些数据区来容纳某些固定参数或一整块的数据。基于此一特性，我们可以在程序中宣告某些数据区并把它们专门作修补处理之用。当 FLASH 存储器被数字产品用来存放软件程序时，我们也可以将某些数据区宣告成填满十六进位的 "0xFF“。如此一来，以后修补程序就可以轻易地将这些“0xFF“十六进位码改成修补程序码，比如修补更正程序的程序码，修补控制程序的程序码，或是用来更新软件区域或修复错误的修补程序码。这是建立在 FLASH 存储器的内容只可以从 “1” 被改成 “0” 而不可以从 “0” 被改成 “1” 的事实上。我们也可以一开始就在修补处理区中定义某些专门处理修补工作的程序。

当 CPU 执行到的软件区域在修补程序数据区存在更新此区域的修补程序时，CPU 的指令指针应被指向修补程序的起始地址，比如图 5 中的修补程序起始地址 502 及图 6 中的修补程序起始地址 602。然而，汇编语言的 jump/branch 指令通常有最大的跳入范围限制。例如，当一个 ARM CPU 使用 THUMB 模式时，branch 指令 “B” 的最大跳跃范

围仅有"2048 字节。如果 CPU 的指令指针不能用一个 jump/branch 指令跳到修补程序的起始地址，多个 jump/branch 指令可被用来做长距离的跳入。例如，当 ARM CPU 使用 THUMB 模式时，我们可以用 branch 指令 "B" 及 Long Branch with Link 指令"BL"来达到比±2048 字节更远的跳入。而为此我们需要为这些 jump/branch 指令保留修补处理区。以下我们将说明一个可以在配置这些修补处理区时节省记忆空间的技术。

首先，我们在每一个软件区域中插入一个 jump/branch 指令。例如，当 ARM CPU 使用 THUMB 模式时，我们插入这个具备最大跳入范围为±2048 字节的 jump/branch 指令 "B"。再把它的位差设定成跳到下一行命令。如此一来，CPU 将执行原来的程序码，这个 jump/branch 指令此时并没有被用到。这个方法已经在 3.3.3 中说明。

一个修补处理区可包含一个修补控制区块 (Patch-Control-Block) 。修补控制区块的功能与修补控制程序相同，并且可以有第二级 (secondary) 的 jump/branch 指令。当使用一个 FLASH 存储器时，修补控制区块可以被事先宣告成填满十六进位的 "0xFF"。如此一来，在修补程序中，将数据写入 FLASH 时可以轻易地将这些十六进位的 "0xFF" 码改成任何程序码，例如第二级的 jump/branch 指令码。而多个软件区域可以共用一个修补控制区块。

图 7 是一个修补控制区块如何执行跳入修补程序数据区的例子。用一个短距离的 jump706 将程序的执行从软件区域 710 带入修补控制区块 709，这个控制区块被用作一个修补控制程序。再用第二个 jump711 将程序的执行带到修补程序 707 的起始地址 702。在执行完毕修补程序 707 后，它跳回到软件区域 710 的最末端。

下列的程序范例是以程序描述语言写成的，它显示一个被四个软件区域 (即区域 A，区域 B，区域 C 以及区域 D) 共用的修补控制区块。在此例中，区域 A 的修补处理区被定义为一个更新处理程序"Jump to Label_O-A"；区域 B 的修补处理区被定义为一个更新处理程序"Jump to Label_O_B" ；区域 C 的修补处理区被定义为一个填满十六进位 0xFF 的 N 个字节的数据区块以作为修补控制区块，及一个更新处理程序"Jump to Label_O_C" ；区域 D 的修补处理区被定义为一个更新处理程序"Jump to Label_O_D "。

```
(At the beginning of Section-A, when there are no patches for updating
Section-A)
  Jump to Label_O_A
Label_O_A:  ORIGINAL SECTION-A CODE
(At the beginning of Section-B, when there are no patches for updating
Section-B)
  Jump to Label_O_B
```

Label_O_B: ORIGINAL SECTION-B CODE

(At the beginning of Section-C, when there are no patches for updating Section-C)

Jump to Label_O_C

Declare a data block area of N bytes being filled with hexadecimal 0xFF, which is used as a Patch-Control-Block

(0xFF,0xFF, ., 0xFF)

Label_O_C: ORIGINAL SECTION-C CODE

(At the beginning of Section-D, when there are no patches for updating Section-D)

Jump to Label_O_D

Label_O_D: ORIGINAL SECTION-D CODE

当收到一个软件修补数据时，修补程序就启动 FLASH 数据写入的程序去改变相对应的软件区域中的 jump/branch 位差，并且也把某些在相对应的修补控制区块中的字节从 0xFF 变更为第二级的 jump/branch 指令，或是变更为某些 DSP/CPU 指令码以执行如 3.3.1 中所讨论的修补更正程序的工作。例如，在收到一个软件区域 A 的修补数据时，修补控制区块中的前几个字节将被改成区域 A 的修补更正程序。在收到软件区域 D 的修补数据时，修补控制区块中的另些字节将被改成区域 D 的修补更正程序。这些修补更正程序将指示 CPU/DSP 的指令指针转向到修补程序的起始地址。

(At the beginning of Section-A, When there is a patch for updating Section-A)

Jump to Label_P_A

Label_O_A: ORIGINAL SECTION-A CODE

(At the beginning of Section-B, When there are no patches for updating Section-B)

Jump to Label_O_B

Label_O_B: ORIGINAL SECTION-B CODE

(At the beginning of Section-C, When there are no patches for updating Section-C)

Jump to Label_O_C

Data block area of N bytes, which is used as Patch-Control-Block

(

Label_P_A: Patch-Correction-Routine of Section-A

Label_P_D: Patch-Correction-Routine of Section-D

Other remaining bytes 0xFF, 0xFF, ., 0xFF

)

Label_O_C: ORIGINAL SECTION-C CODE

(At the beginning of Section-D, When there is a patch for updating Section-D)

Jump to Label_P_D

Label_O_D: ORIGINAL SECTION-D CODE

以下我们将介绍一种减小修补控制区块大小的技术。在某些情况下，节省 FLASH 存储器空间相当重要，或是有使用存储器空间的限制，那么尽可能减少 FLASH 存储器的使用就变成必要的。减少修补处理时使用的 FLASH 存储器的方法之一，就是减小修补控制区块的大小。在前面的范例中，修补控制区块是被四个软件区域所共用，即区域 A、区域 B、区域 C 与区域 D。然而如果这四个区域同时都有其相对修补程序的机率非常低，那么修补控制区块就可以被设计成仅够处理三个软件区域或甚至两个软件区域所需的空间。

3.5 配置修补处理程序时的软件编译

在原始的内嵌软件中配置更新处理程序或修补处理区后，进行软件编译时可能会出现某些复杂的情况。

以下我们介绍一种能够避免软件编译时无法对齐程序字组的技术。如同前述，每一个软件区域都可以被配置一个更新处理程序来检查该区域在修补程序数据区中是否有修补数据。我们也可以藉由在组合程序码中宣告多个存储器区块为参数数据区来配置修补控制区块，而该区块可以用来容纳第二级的 jump/branch 指令。然而，这样地插入程序码可能会造成软件编译时的复杂状况。例如在软件编译中某些 CPU 可能要求程序指令要对齐到字组。例如，当 ARM CPU 使用 THUMB 模式时，汇编命令 LDR 可能需要一个可以被 4 整除的地址位差 (offset address) 。因此，当我们插入 jump/branch 指令及修补控制区块以供修补处理时，需要调整我们插入的字节使得 LDR 的地址位差能够被 4 整除。也就是说，藉由调整用作修补处理的字节，我们可以解决在编译过程中需要对齐字组的问题。

以下我们介绍一种可以避免在编译过程中发生位差超出容许范围的技术。例如，当 ARM CPU 使用 THUMB 模式时，汇编命令 LDR 可以用来把一些位差少于 2048 字节的数据放入暂存器中。如果为了修补处理而插入的程序码是在 LDR 与位差地址之间，而位差地址处含有一些要被放入暂存器的数据，则插入程序码后这个 LDR 所需的位差值将有可能大于 2048 字节，如此就会产生编译问题。一项可以避免此种问题的技术就是将这

些必须被载入的数据移到一个比较接近某些汇编命令 (比如 LDR 码) 的地方，如此一来，位差值就可以少于 2048 字节。也就是说，藉由把这些必须以某些汇编命令 (比如 LDR) 载入暂存器的数据移到或复制到较接近这些汇编命令 (比如 LDR) 的地方，就可避免在编译过程中某些汇编命令发生位差超出范围的问题。

同样地，在将某些与修补程序相关的字节放到内嵌软件中后，某些原来的程序码 (比如 ARM 汇编命令的 B 指令) 的位差可能会超出范围，亦即 B 指令要跳入的目标位置过远。在 B 与目标位置之间插入另一个 jump/branch 指令可防范在编译过程中发生超出范围的问题。也就是说，将长距离的 jump 分成两次来跳。

3.6 以新的修补程序来更新已有的修补程序

一个数字产品在成功地写入一个修补程序后，就会开始使用它来取代软件区域中原有的程序码。然而，即使是已经写入的修补程序有时还是有可能会有某些问题，或是这个修补程序在一段时间后必须加以更新。下列说明即介绍一个更新修补程序的技术。

为了对数字产品中已经写入的修补程序进行更新，我们可以做一个新的修补程序来更新它。为了支持这个功能，当设计一修补程序时，应保留一些码的空间给未来新的修补程序做修补处理之用；或类似前述的更新软件区域的方式，在修补程序中插入一个更新处理程序。因此，当 CPU/DSP 执行到一个已有旧修补程序的软件区域却发现它有一个新修补程序时，CPU/DSP 可以经由这个更新处理程序将指针指向新的修补程序而非旧的修补程序。

于本申请书中所展示的所有用来设计修补程序以更新软件区域的技术，均可以被应用于设计修补程序来更新另一个已存在的修补程序。

3.7 避免编译器最佳化处理所生效应

内嵌软件可以以 C/C++/JAVA/Assembler 或其他程序语言撰写。如果一个软件程序并非以汇编命令所撰写，而是以 C/C++/JAVA/或其他程序语言所写时，这个程序可以透过编译工具加以翻译成汇编命令。软件编译工具可能对软件程序进行最佳化处理，在将 C/C++/JAVA/或其他程序语言所撰写的程序转换成目的码或汇编命令的编译过程中，可能对原本的程序进行更改/最佳化。由于最佳化的处理，某些程序码可能消失。因此，如果我们把更新处理程序直接放在以 C/C++/JAVA 撰写的程序中，这个更新处理程序可能在编译器执行最佳化的过程中被改变或被移除。然而，在完成编译后，将汇编命令转换成目的码的阶段中，一般并不会再做程序码最佳化的动作。而且在使用软件连结工具把目的码连结成最后的可执行的机器码的阶段，也不会再做程序码最佳化的动作。

一个以 C/C++/JAVA 以及其他程序语言所撰写的软件程序可以以下列两种方式加以编译。

编译方法 1：使用软件编译工具将以 C/C++/JAVA 程序语言所写的程序码转换为目的码 (可能并用最佳化程序) 。然后，透过软件连结工具将目的码连结以产生可执行机器码。

编译方法 2：使用软件编译工具将以 C/C++/JAVA 程序语言所写的程序码转换为汇编命令 (可能并用最佳化程序) 。然后，利用软件组合工具将汇编命令转换为目的码。最后，透过软件连结工具将目的码连结以产生可执行机器码。

正常情形下，由编译方法 1 所产生的最后可执行的机器码与编译方法 2 所产生的最后可执行的机器码是相同的。如果一个软件程序是直接以汇编语言撰写，我们可以透过使用软件组合工具将之转换为目的码。然后，透过软件连结工具将目的码连结以产生可执行的机器码。

在此说明一个利用编译方法 2 将更新处理程序与修补处理区放入软件程序中的技术。于编译方法 2 中，在利用软件编译工具将一个以 C/C++/JAVA 程序语言所撰写的程序码转换为汇编命令码 (可能并用最佳化程序) 之后，我们可将更新处理程序、修补控制程序与修补处理区插入这个汇编命令中。因为在将汇编命令转换为目的码以及将目的码连结以产生可执行机器码的编译过程中，是不会执行最佳化程序而改变程序码或将之最佳化的，所以修补程序及修补程序所在的空间将不受影响。

另一个方法是，将一个用 C/C++/JAVA 程序语言所撰写的程序码转换为目的码，然后把更新处理程序、修补控制程序或修补处理区插入目的码中。之后再透过软件连结工具将目的码连结以产生可执行的机器码。

3.8 以软件工具配置修补处理程序

本发明另一方面指出了在数字产品 (例如行动电话) 的内嵌软件中配置或插入前述的更新处理程序以及修补处理区的方法。由于数字产品是在生产线上完成制造，因此必须先写入内嵌软件使其得以处理之后才能推出到市场上。而在内嵌软件被写入数字产品之前，更新处理程序及/或修补处理区应当已被放在其中。

为了有效率地在数字产品的内嵌软件中配置更新处理程序及修补处理区，本发明另一方面提出了一个配置工具来完成自动配置。

配置工具的一个实际应用就是将内嵌软件的编译程序及配置程序整合起来。例如，在对原始码进行编译前，配置工具可以自动地起动来完成配置的工作。在这样的方式下，软件开发者将只需要考虑自己的程序撰写工作，而无须涉及或担忧配置的处理。

例如，配置处理的一个应用可被实行在汇编命令而非 C/C++/JAVA 码之上。在此情况下，C/C++/JAVA 码将先被编译为相对应的汇编命令，然后，配置程序再被用来在汇编命令中插入更新处理程序与修补处理区。之后汇编命令再进一步被编译为目的码。在此方式下，任何对 C/C++/JAVA 码的编译最佳化处理将不会影响更新处理程序以及修补处理区。

虽然娴熟相关领域技术的人透过了解本发明内容就可以轻易研发自己的配置工具，我们仍将介绍以下两个配置软件工具的例子。

[范例 A]

对于以 C/C++/Java/Assembler 或其他程序语言，或 CPU/微处理器及 DSP 的可执行机器码撰写的软件程序码，配置工具可以每隔 N 行或 N 个字节 (N 是一个事先定义好的数字) 就插入一个更新处理程序及/或修补处理区。配置工具在必要时也可依据相对应的区域识别码来变更更新处理程序中的参数与标签。配置处理是直接在含有内嵌软件原始码的原始文件中进行的。

[范例 B]

在某些应用中，数字产品中的内嵌软件的原始码可能包含在一个以上的文件中。对于每一个文件，配置工具将产生另一个与原来文件含有相同程序码的文件 (亦即对该文件做一个拷贝) 。然后，配置工具把更新处理程序及/或修补处理区插入复制的文件中，而非原来的文件中。也就是说，对于以 C/C++/Java/Assembler 及其他程序语言或是 CPU/微处理器及 DSP 的可执行机器码撰写的软件程序码，配置工具可以在复制的文件中每隔 N 行或 N 个字节 (N 是一个事先定义好的数字) 就插入一个更新处理程序及/或修补处理区。配置工具在必要时也可依据相对应的区域识别码来变更更新处理程序中的参数及标签。

4. 修补数据的传送与接收

4.1 修补数据的传送

首先我们将说明如何把软件修补数据从修补服务器传送到数字产品中。图 8 显示一个使用某些通讯协定层来传送修补程序的范例。依据传输系统的设计方式及需求的不同，有些实际应用可能仅包含支持控制修补程序传输的三层协定中的一层或两层。某些其他的应用可能利用数据传输通道或传输管道，比如线路数据传输或数据包数据传输。也有某些其他的应用可能使用声音或影像传输通道，把修补程序放在这些通道或传输管道中来传送修补程序到数字产品中。

所谓修补服务器，对于熟知此一领域技术之人来说，应是一个可以将软件修补数据分送或传输到一个或多个数字产品中的系统，且最好是透过现存的通讯设备或通道，比如无线连接，或是有线连接。修补服务器在处理时可有某种程度的自动化。如图 3 所示在传输时，一个数字产品透过它的修补数据接收模块 302 来接收修补程序。

在修补程序传输的过程中或传输完成后，数字产品可传送信号 (或信息) 到修补服务器中来做确认。另一个例子是数字产品可传送信号 (或信息) 到一个修补回复信息收集器而非传送到一个修补服务器。然后，这个修补回复信息收集器会传送一个相对应的信号与数据到修补服务器中。这个修补回复信息收集器是一个用来收集来自数字产品的回复信息的信息接受器。它可以被放在与修补服务器不同的地点。

对于使用以 TIA/EIA-95 及 CDMA2000 标准为基础的 CDMA 技术的数字行动电话，可以使用 Data Burst Messages 来携带修补程序。一个使用 Data Burst Messages 的例子就是使用以 TIA/EIA-637 标准为基础的短信 (SMS) 传输。以下我们将说明一个使用短信来传送修补程序的范例，不过熟悉数据传输领域的人可以使用其它的信号或信息来设计自己的传输方法。由于短信传输的通讯协定 (TIA/EIA-637) 已包含了查错的程序，因此使用短信作为修补程序传输的低层协定，数字产品将可接收到几乎无错误的修补程序数据包。

参考图 8，我们说明一个依据本发明而实施的修补程序传输协定的连结层 820 范例。连结层 820 的工作是通过检测传输错误及重传有错的数据来可靠地传送修补信号信息 800 及修补程序 810。查错可以使用以 CRC 或 checksum 为基础的技术，或其他被熟悉此一领域者所惯用的技术。连结层 820 也可包含安全检查机制。例如，在传送修补程序前，修补服务器可以使用某些加密演算法对一修补程序进行加密后再将该修补程序送到数字产品中。当数字产品接收到一个修补程序时，它将对这个修补程序进行解密运算来检查所收之修补程序的正确性。

在使用短信的范例中，由短信所携带的修补程序数据包被收到后，这个短信会被传到解密模块去进行 CRC 检查及解密程序来检查这个修补程序数据包的正确性。

在一个数字产品不能正确地接收修补程序时，修补服务器应该再传送修补程序到这个数字产品。例如，修补程序的重新传输可以建立在一个以时间为基础的机制之上：这个修补服务器具有一个支持修补程序传输的计时器。修补服务器在送出一个修补程序给数字产品后就会启动计时器。如果修补服务器在其计时器计时终了时尚未收到任何从数字产品所传送来的相对应的回复信息 (例如修改状态报告)，修补服务器将会重新传送这个修补程序数据包给数字产品。在某些特别状况下，当服务器在执行预设次数的修补程序重新传送后，仍未收到任何来自于数字产品所传出的回复信号时，服务器将会暂停重新传输的工作。对于熟悉此一领域技术之人，可以透过了解本发明而运用其他的计时方式来做修补程序的传输及重新传输。

在之前使用短信的范例中，有一个名为修补状态报告的修补信号信息，它是一个由数字产品传送到修补服务器的信息，其中包含了该数字产品当前的修补状态数据。如果这个数字产品接收到一个无错误的修补程序，在下列情况下它将会利用一个反向的短信传送出修补状态报告给修补服务器：

(a)当这个数字产品发现所收到的修补程序曾被成功地写入 (programmed)，或

(b) 当这个数字产品成功地完成修补程序的写入工作。

4.2 修补程序的编写

参考图 9，在此将进一步说明修补程序 940 的编写。最好将修补程序 940 划分成数个修补程序数据包，每一个修补程序数据包可以由传递层 820 的一个信息来携带。

在使用短信的范例中，一个修补程序数据包可以用一个短信信息来携带。为了与一般的短信信息区别并辨识载有修补程序的短信信息，修补服务器在送出修补程序之前可先将修补程序短信中的使用者数据 (User Data) 的前 K 个字节 (K 是一个事先定义好的数字，可以是任何一个从零开始的正数) 设定成一个 K 字节密语 (K-character Magic-word)。在该密语之后，插入一个修补程序数据包到短信信息中。

图 9 显示修补程序与修补程序数据包之间的关系。修补程序数据包 950 可以被划分为数据包前置数据组 945 以及负载信息数据组 947。一个数据包前置数据组 945 的设计可包含以下的数据：

(a) 修补程序识别码

(b) 当前数据包号码

(c) 最后数据包号码

熟悉此一领域之人可在了解本发明内容后以其他方式来设计数据包前置数据组。

在修补程序数据包 950 中的负载信息数据 947，955，965 可以包含全部或一部份的修补程序 940。

修补程序数据包 950 可以由修补服务器在传输前，先以某些加密演算方法予以加密。修补服务器可以利用静态及/或动态的钥匙来对软件修补程序进行加密：前者可以是一个 CPU 识别码，MAC 层地址及其他.电信服务相关的参数，比如电话号码, ESN 及 TMSI 临时辨识玛;后者是以位置或时间或使用者档案数据为基础的可变的钥匙。

4.3 修补程序处理

在接收到一个可能含有修补程序数据包 950 的信息后，数字产品将会检查这个信息是否包含一个真正的修补程序数据包。数字产品也可以对这个收到的信息进行查错及/或数据解密。在成功地接收到所有的必要的修补程序数据包后，这个数字产品将会把数据包中的负载信息数据 947,955,965 组合成相对的修补程序。

在使用短信信息的范例中，当数字产品接收到一个短信信息时，它将会检查信息中的前 k 个字节是否与密语相同。如果不吻合的话，这个短信信息将会被当成一个一般的短信信息而非一个修补程序数据包。

如果吻合的话，数字产品将会进一步对这个信息数据执行解密以检查这个信息数据的正确性。如果解密程序显示这个信息数据是正确的，这个信息数据将被当成一个修补程序数据包，并且会用一个缓冲区来暂存这个数据。否则，这个短信信息将被当作一个一般的短信信息。

在根据修补程序识别码数据 (即修补程序数据包中的当前数据包号码及最后数据包号码) 成功地收到所有必要的修补程序数据包后，这个数字产品将会把这些数据包组合成相对应的修补程序。

修补程序 940 可以包含某些下列的参数项目，比如：

(a) 确认所收到的修补程序是否是给这个数字产品的参数项目，例如

- ESN (Electronically Serial Number) ;

- MIN (Mobile Identification Number) 或 IMSI (International Mobile Station Identity) ;

- 制造识别码 (Manufacture Identifier) ;

- 产品型号 (Product model number) ;

- 软件版本号码 (Software version number) ;

- CPU 识别码 (CPU Identifier) ;

- MAC (Medium Access Control) 层地址;

- 其他可以作为识别一个数字产品的参数

- 其他可以作为识别用户的参数

(b) 修补数据写入 (patch programming) 所需的参数项目，例如，

-控制数据，用来指出某些修补程序必须被写入 FLASH 存储器，及/或某些修补程序必须被写入 NVM (Non Volatile Memory) 存储器及/或 EEPROM (Electrically Erasable Programmable Read-Only Memory) ;

-修补程序的地址数据，这可能是存储器地址，区域配置数据或其他在数字产品的 FLASH/NVM/EEROM 存储器的地址数据;

-可以被用来做软件更新的修补程序码;

-如何解译修补程序码的数据，例如在该修补程序中被使用的程序语言数据 (C/C++/Java/Assembler 及其他程序语言，或 CPU/微处理器或 DSP 的可执行机器码)

-可被用来更新在 FLASH 存储器，NVM，及/或 EEPROM 存储器中某些数据区或数据表的修补数据;

-被这个修补程序所更新的软件区域的识别码;

-其他用来描述修补程序与数据区更新的数据。

(c) 用以决定修补数据写入后的后续程序之参数项目，例如，

-重新起动旗标 (Reset Flag) 。

-传送回复信息到修补服务器的控制旗标。

-其他用以决定修补数据写入后的后续程序之控制信号与数据。

在收到的修补程序被组合后，必要时该数字产品最好对合成的修补程序进行解密。这是为了加强对修补程序的安全检查。当然，修补服务器端应在将修补程序分成修补程序数据包之前对该修补程序进行加密。

修补程序码可以以 C/C++/Java/Assembler 及其他程序语言或 CPU/微处理器或 DSP 的可执行机器码撰写。这个修补程序可包含某些如何转译修补程序码的转译数据。如果这个修补程序码可以根据这个转译数据来加以转译，某些做法是用一个软件翻译模

块来把收到的修补程序从一个格式翻译成另一个格式。在某些应用之中 CPU/DSP 需要用到这个修补程序，那么修补程序也可以被转译或翻译成 CPU/DSP 的可执行机器码。

修补程序中含有软件更新的数据，该更新数据是用来取代内嵌软件中某一部份原来的程序。此修补程序可以被做成从修补起始地址开始。而且，举例说，在一个修补程序的结尾处，可以用一个指令来让程序跳回一个在内嵌软件中预先定好的位置，而该位置是在被跳过的过时/错误原始码之后，或是过时/错误原始码所在的区域的末尾处。

修补程序也可以包含用以更新 FLASH 存储器，NVM，及/或 EEPROM 存储器中的数据区的数据。例如，一个修补程序可以包含一个新的数据表，用来取代在数字产品的 NVM 存储器中的旧数据表。

数字产品可检查修补程序中的某些数据项目 (例如，ESN，制造识别码，产品型号，以及软件版本号码等) 是否与储存在该数字产品中的参数值相同。如果不吻合的话，这个数字产品将丢弃这个修补程序 (或是如在使用短信信息的例子中将那些信息当成一般的短信) 。如果吻合，这个数字产品将进一步检查这个收到的修补程序是否曾经被写入到它的存储器中了。如果还没有被写入过，这个数字产品将开始把收到的修补程序写入存储器。否则，它会送出一个修补回复信息给修补服务器，并且完成修补程序的处理。

5. 修补数据写入

修补数据写入的程序牵涉到把修补程序写入 FLASH 存储器中的相对应的修补程序数据区，及/或把新的参数写入 NVM/EEPROM 存储器之中。修补数据写入的程序最好是在这个数字产品处于待机 (idle) 状态 (即未被使用状态) 时进行。将新参数写到 NVM/EEPROM 存储器中的工作可使用一般的数据复写程序来进行。然而，要将数据写入 FLASH 存储器中，在某些情况下需要额外的处理工作，特别是当这个 FLASH 存储器正在被这个数字产品读取时。为了将修补程序写入正在运作中的数字产品中，我们将介绍一种在 RAM 存储器中执行 FLASH 数据写入程序的技术，此一技术亦是本发明的一种应用。

5.1 利用 RAM 存储器执行 FLASH 数据写入的程序

用来执行将数据写入 FLASH 存储器中的软件程序，可以被放在 RAM 存储器之中。熟悉此领域技术的人已有许多方式可将软件程序放到 RAM 存储器中。例如，可以在将数据写入 FLASH 存储器之前，将一个 FLASH 数据写入的程序用一个软件程序复制到 RAM 存储器中。另一个例子是，我们可在编译内嵌软件时指示编译器把 FLASH 数据写入的程序放在 RAM 存储器中，如此一来，编译器就会自动地将这个程序配置到 RAM 存储器中。RAM 存储器可以是与 CPU/DSP 核心位在同一晶片上的 RAM/SRAM 存储器，或是位在包含 CPU/DSP 的晶片之外的 RAM 存储器。当有必要将修补相关数据写到 FLASH 存储器中时，可以在 RAM 中执行 FLASH 数据写入的程序。这个程序会将数

据写入相对应的 FLASH 地址中、读取 FLASH 的状态值，并且确认数据写入的正确性以及工作是否完成。这项将数据写入 FLASH 的工作最好被设定成具有最高的优先执行顺序，并且当它开始执行时，最好先停止所有的 CPU/DSP 中断服务的功能，如此当它在执行时，所有其他软件的工作以及所有的 CPU/DSP 的中断服务将处于停止并等待的状态中。

另一个方式是在一个中断处理程序中执行 FLASH 数据写入的程序，并且当这个程序开始执行时，最好先停止所有其他的中断服务，如此当它在执行时，所有的软件工作以及 CPU/DSP 的中断服务程序将处于停止并等待的状态。

另一个确保将数据写入 FLASH 的过程中不受来自其他软件工作及中断服务程序的影响的方法，是在开始执行 FLASH 数据写入的程序之前停止与 FLASH 数据写入的程序或软件修补处理无关的软件的工作以及中断服务程序。

5.2 修补数据写入的程序

修补数据写入的程序可被分为数个子状态，图 10 即显示一个这样的范例。在成功地接收到修补程序后，这个数字产品将会把修补程序写入 FLASH 存储器，NVM 及/或 EEPROM 存储器之中。

如果某些修补程序是用来更新在 FLASH 存储器，NVM 及/或 EEPROM 存储器中的的数据区或数据表，此数字产品应将这些数据写入 FLASH 存储器，NVM 或 EEPROM 存储器中的相对应地址。

如果某些修补程序是用来更新某些软件区域中的程序码，此数字产品应将修补程序码写入 FLASH 存储器中的相对应地址。

然后，这个数字产品可以开始将修补识别数据写入修补控制表 1020 或某些修补资料库中。这些修补识别数据可以包含修补数据地址，修补数据识别码以及软件区域识别码等。

然后，此数字产品将会进行下一个步骤，即区域状态写入 1030。根据前面介绍过的更新处理程序的设计方式，一个软件区域的更新处理程序可以包含一个或两个 JUMP 指令，或包含一个可以检查修补旗标的修补检查程序。在修补程序成功地写入后，如果更新处理程序使用修补标旗，那个相对应的修补标旗将被变更以反映修补程序存在的事实;如果更新处理程序是使用 JUMP 指令，相对应的 JUMP 位差也必须被更改，使得 CPU/DSP 指令指针跳到相对应的修补程序起始地址。某些在修补控制区块及/或修补处理区中的数据字节有可能必须被 FLASH 数据写入的程序变更。在前述的说明中已详细描述从无修补程序的状态变到有修补程序的状态时，更新处理程序及修补控制区块的改变。

在更新处理程序中，改变关键性的 JUMP 位差或修补旗标的动作最好是放在最后进行，这样如果修补数据写入的程序在中途停止时，该数字产品不致于无法正常运作。

例如，如要更改如本说明第 3.4 节所描述软件区域 A 中的更新处理程序时，此数字产品应首先将修补控制区块中的数据字节在 Label_P_A 处变更为实施 Patch-Correction-Routine of Section-A 的指令。然后再修改 JUMP 指令的位差来把 Jump to Label_O_A 改成 Jump to Label_P_A。用这样的方法，从使用原来的程序码转换成使用修补程序码的动作，只有在所有的写入处理都完成后才会发生。

当修补数据写入成功地完成时，此数字产品将会进行某些后续处理，比如发送修补状态信息到修补服务器，及/或在必要时执行系统重新起动。在使用短信信息的例子中，数字产品可能执行以下的后续处理：

-此数字产品将会以短信信息的方法发送一个修补状态报告给修补服务器。这个修补状态报告包含用户终端装置的数据以及当前的修补状态。

-然后，如果在修补程序中的重置标旗已被设定，此数字产品将进行系统重新起动。

6. 修补服务器

参考图 11，显示一个修补服务器 1100 的范例之简化图。一个修补服务器 1100 最好拥有一个修补资料库 1130，一个修补数据生成器 1110，以及一个修补数据传输控制器 1120。

修补资料库 1130 储存与每一个数字产品有关的修补数据。它可以储存如下的项目：

(a) 制造识别码;

(b) 产品型号；

(c) 软件版本号码；

(d) 软件区域识别码；

(e) 修补数据识别码；

(f) CPU 识别码；

(g) MAC (Medium Access Control) 层地址;

(h) 其他可以作为识别一个数字产品的参数项目；

(i) 其他可以作为识别用户的参数项目；

(j) 修补数据的地址数据；

(k) 重置旗标设定数据 (Reset Flag Information)；

(l) 修补程序码。

(m) 有关如何转译修补程序码的数据，例如在该修补程序中被使用的程序语言数据 (C/C++/Java/Assembler 及其他程序语言，或 CPU/微处理器或 DSP 的可执行机器码)。

(n) 用来更新此数字产品的数据记忆区 (NVM/EEPROM) 的数据参数

当修补服务器要送一个修补数据给一个数字产品时，修补数据生成器 1110 就会根据修补资料库 1130 中的数据来产生一个相对应的修补程序。当修补程序无法用一个修补程序数据包来载运时，修补数据生成器 1110 可进一步将修补程序分为数个区域，并把每一个区域放入一个修补数据数据包中来传输。

修补数据传输控制器 1120 是负责将修补程序数据包可靠地传送到数字产品中的装置。一个设计的范例就是用一个计时器来控制修补程序的重新传输，其详细的设计内容将在以下说明。

修补服务器在发送一个修补数据给数字产品后就会启动一个计时器。如果修补服务器在这个计时器所设定的时间经过后，并未收到任何从该数字产品所回应的回复信息(例如修补状态报告)，修补服务器将会再发送这个修补程序数据包给这个数字产品。在某些特别的情况下，修补服务器已经执行了一个预定次数的重新传输，却没有收到任何从数字产品发送回来的回复信息，那么这个修补服务器可以暂停重新传输的动作一阵子，或是完全停止使用这个计时器并且在一个记录档案中注明未收到回复信息。

在收到一个来自于数字产品的回复信息 (例如修补状态报告) 时，修补服务器将会停止控制修补数据重新传输的计时器，并且在修补资料库中更新这个数字产品的修补状态。

以下我们用一个设计范例来说明一个软件修补数据如何经由短信信息从修补服务器被传到数字产品中。这个设计范例的说明将参考图 12。

在这个设计范例中，含有内嵌软件的数字产品是数字行动电话 1250。修补服务器的硬件是由一台电脑 1200 和一个能够接收短信信息的无线数据机 1210 所组成。这台电脑将产生修补数据及相对应的修补短信信息，之后经由网际网络 1205 将此信息传到一个通讯服务提供者的信息服务器 1240 中。而该通讯服务提供者则提供无线服务给数字产品 1250。将这个信息传到信息服务器的传输方式可以采用一般电子邮件所使用的 SMTP (Simple Mail Transfer Protocol) 及 TCP/IP。通讯服务提供者的信息服务器 1240 将把修补短信信息当成一般的短信信息 1225 传给基地台 1220。最后，基地台 1220 经由无线介面信息 (比如载送短信信息 1225 的 Data Burst Message) 将这个信息 1225 传到相对应的行动电话 1250 中。

修补服务器的无线数据机 1210 是接在修补服务器的电脑 1200 上。这个无线数据机是用来接收来自于数字产品 1250 的回复信息 1215 并将这个回复信息 1215 转送到电脑 1200 中。电脑 1200 将根据这个回复信息 1215 中所含的数据来确认修补的状态。

虽然本发明在本申请书中是以我们所喜好的应用为参考来加以说明，但一个熟悉同一领域的人已经可以在不偏离本发明的范围的情况下，用其他应用方法来取代前述的各项实施说明。因此，本发明所包含的范围当仅受限于以下的专利要求项目。

缩写词汇

ARQ. Automatic Repeat reQuest. 自动再送信要求

CPU. Central Processing Unit. 中央处理器

CRC. Cyclic Redundancy Codes. 循环冗长码

DSP. Digital Signal Processor. 数字信号处理器

EEPROM. Electrically Erasable Programmable Read-Only Memory. 可删除可编辑只读存储器

ESN. Electronic Serial Number. 电子型连续码

FLASH. 一种持续有电的不变性存储器（constantly-powered nonvolatile memory），可以区块（block）为单位被删除及重写.

MAC. Medium Access Control. 媒介操作控制层

MCU. Micro Processor Unit. 微处理器

NVM. Non Volatile Memory. 不挥发性存储器

Patch data. 所有由修补服务器（patch server） 送给数字产品的修补数据所合成的数据.

Patch data packet. 修补程序数据包，即修补程序传输的单元. 一个修补数据可以被分成数个修补程序数据包来传输. 一个修补程序数据包可由一条信息来载送.

Patch program. 用来更新软件的软件程序, 它是由一或多个 CPU/DSP 指令所组成, 可用来更新内嵌软件中的一或多个程序码.

Patch programming. 将修补程序写入存储器的过程.

Patch server. 修补服务器。一或多个用来传送修补程序给数字产品的服务器.

PCMCIA. Personal Computer Memory Card International Association. 个人电脑存储卡国际协会

PDA. Personal Digital Assistant. 个人数字助理

RAM. Random Access Memory. 随机存取存储器

ROM. Read-Only Memory. 只读存储器

SMS. Short Message Service. 短信服务

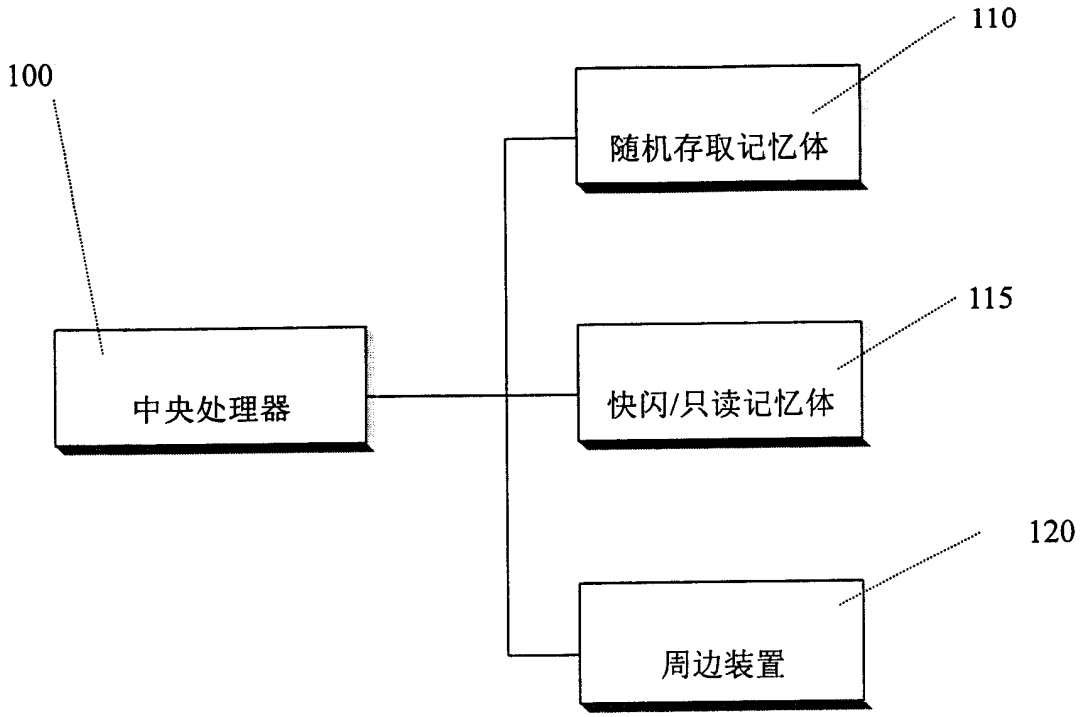

100
110
随机存取记忆体
115
中央处理器
快闪/只读记忆体
120
周边装置

图 1

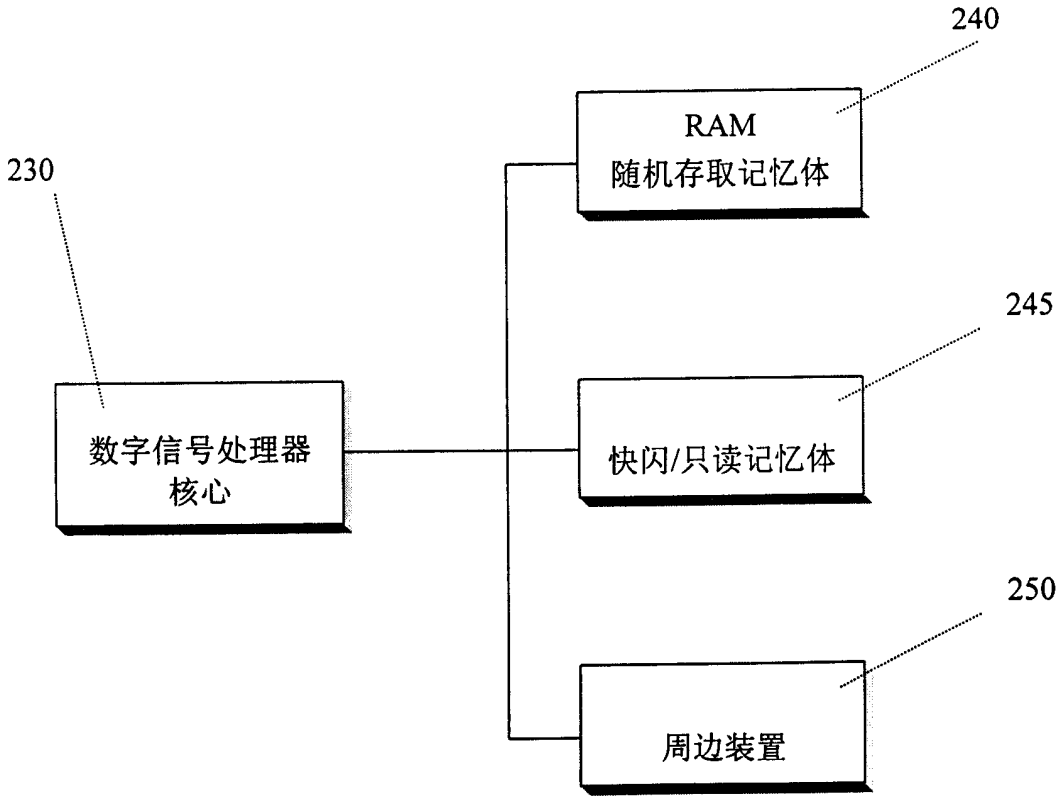

240
RAM
随机存取记忆体
230
245
数字信号处理器
核心
快闪/只读记忆体
250
周边装置

图 2

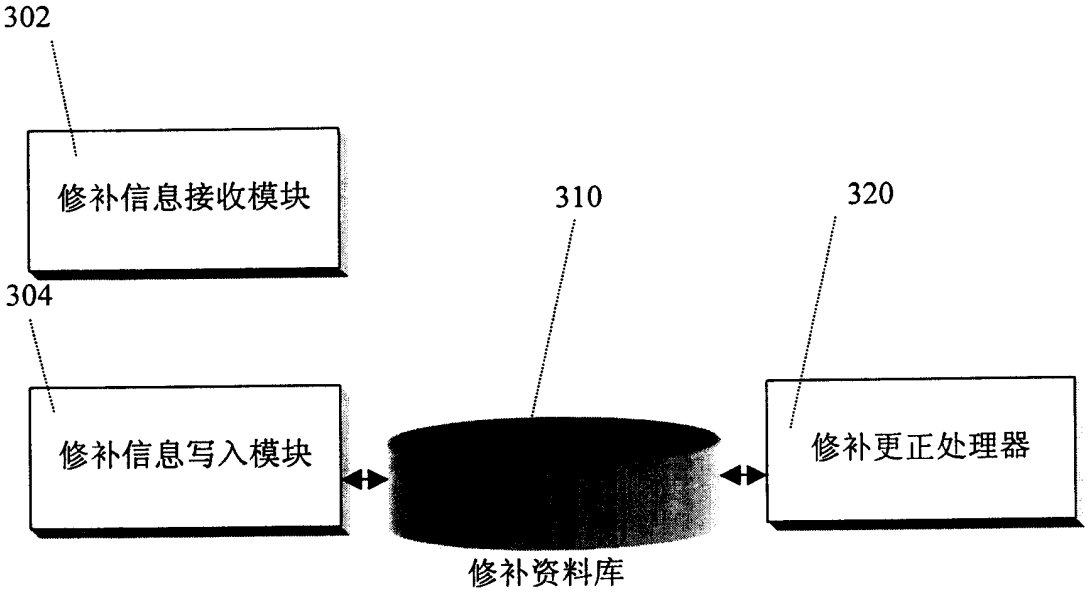

302
修补信息接收模块
310
320
304
修补信息写入模块
修补更正处理器
修补资料库

图 3

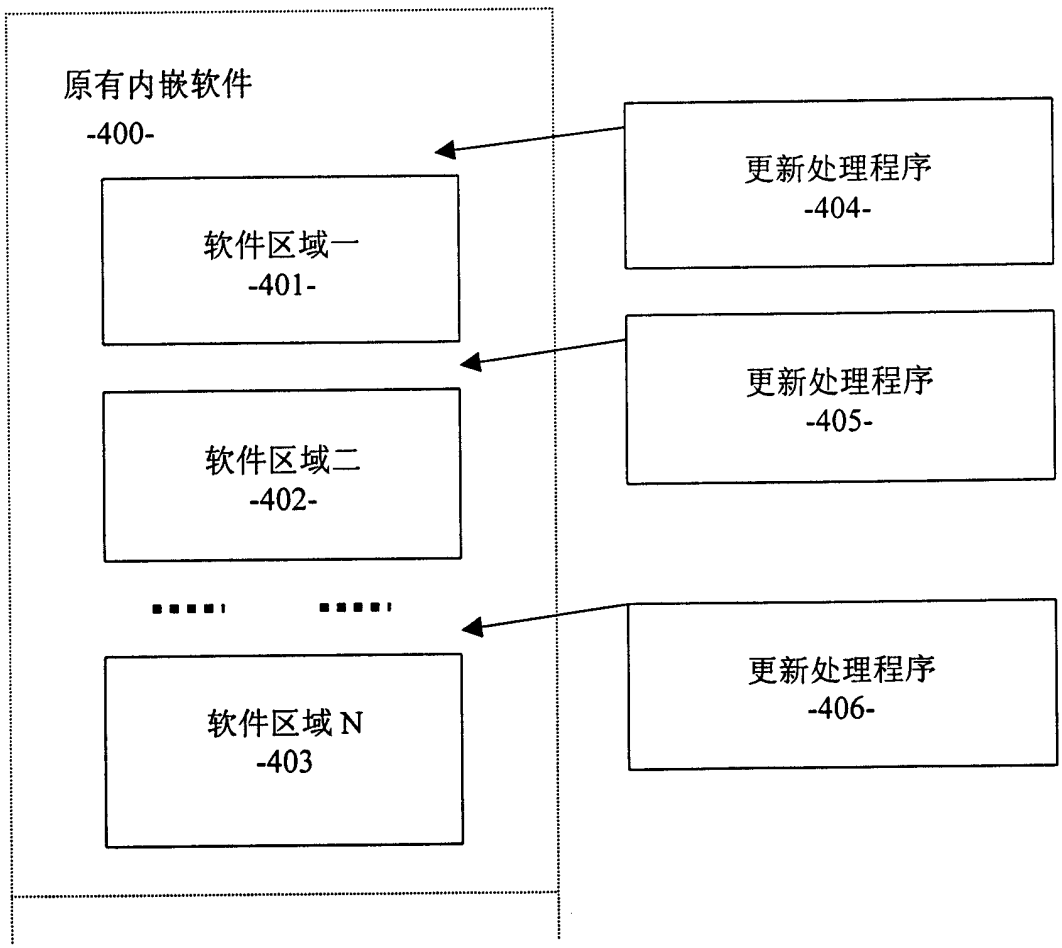

原有内嵌软件
-400-
软件区域一
-401-
软件区域二
-402-
软件区域 N
-403
更新处理程序
-404-
更新处理程序
-405-
更新处理程序
-406-

图 4

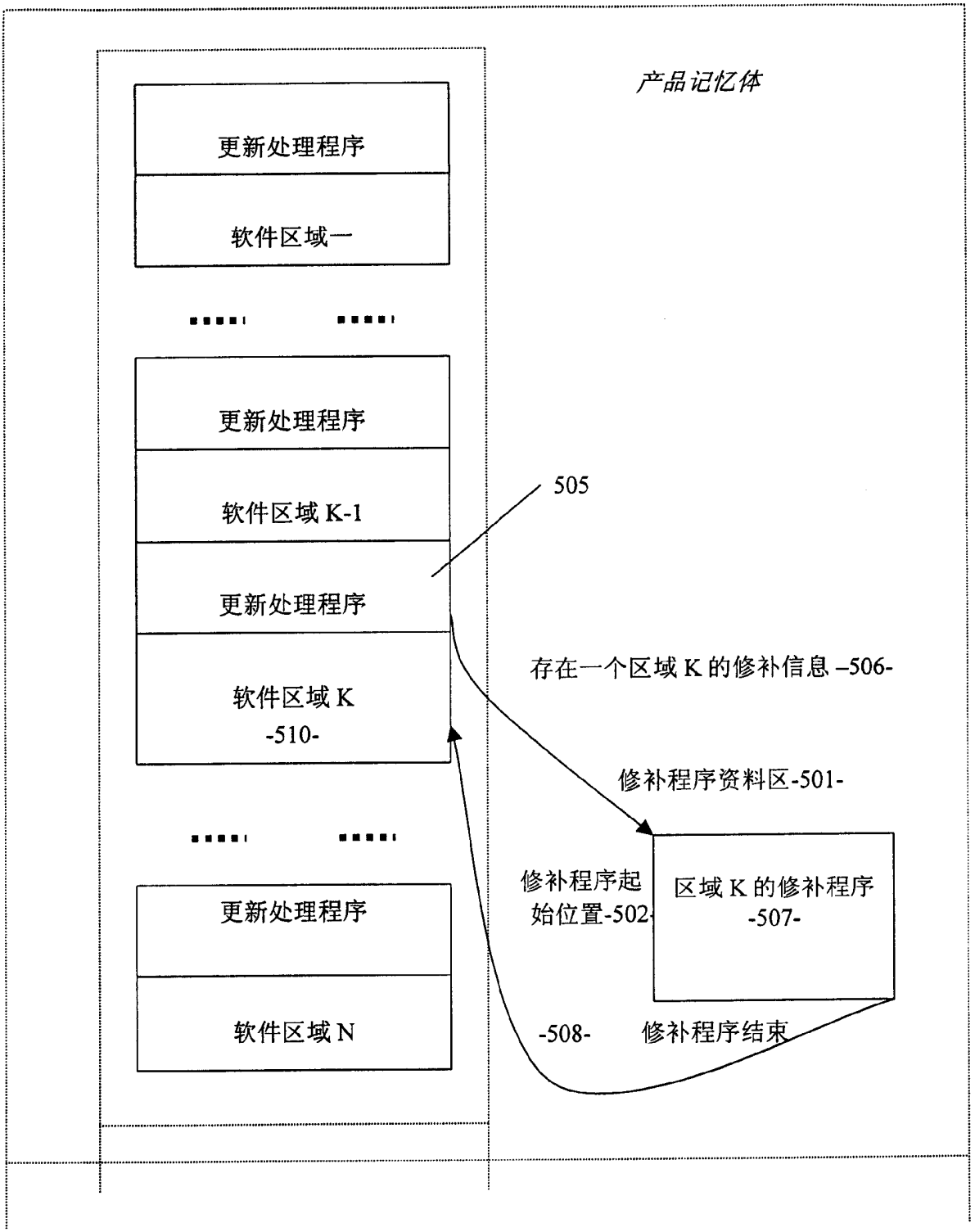

产品记忆体
更新处理程序
软件区域一
更新处理程序
软件区域 K-1
更新处理程序
软件区域 K
-510-
更新处理程序
软件区域 N
505
存在一个区域 K 的修补信息 –506-
修补程序资料区-501-
修补程序起始位置-502-
区域 K 的修补程序
-507-
-508-
修补程序结束

图 5

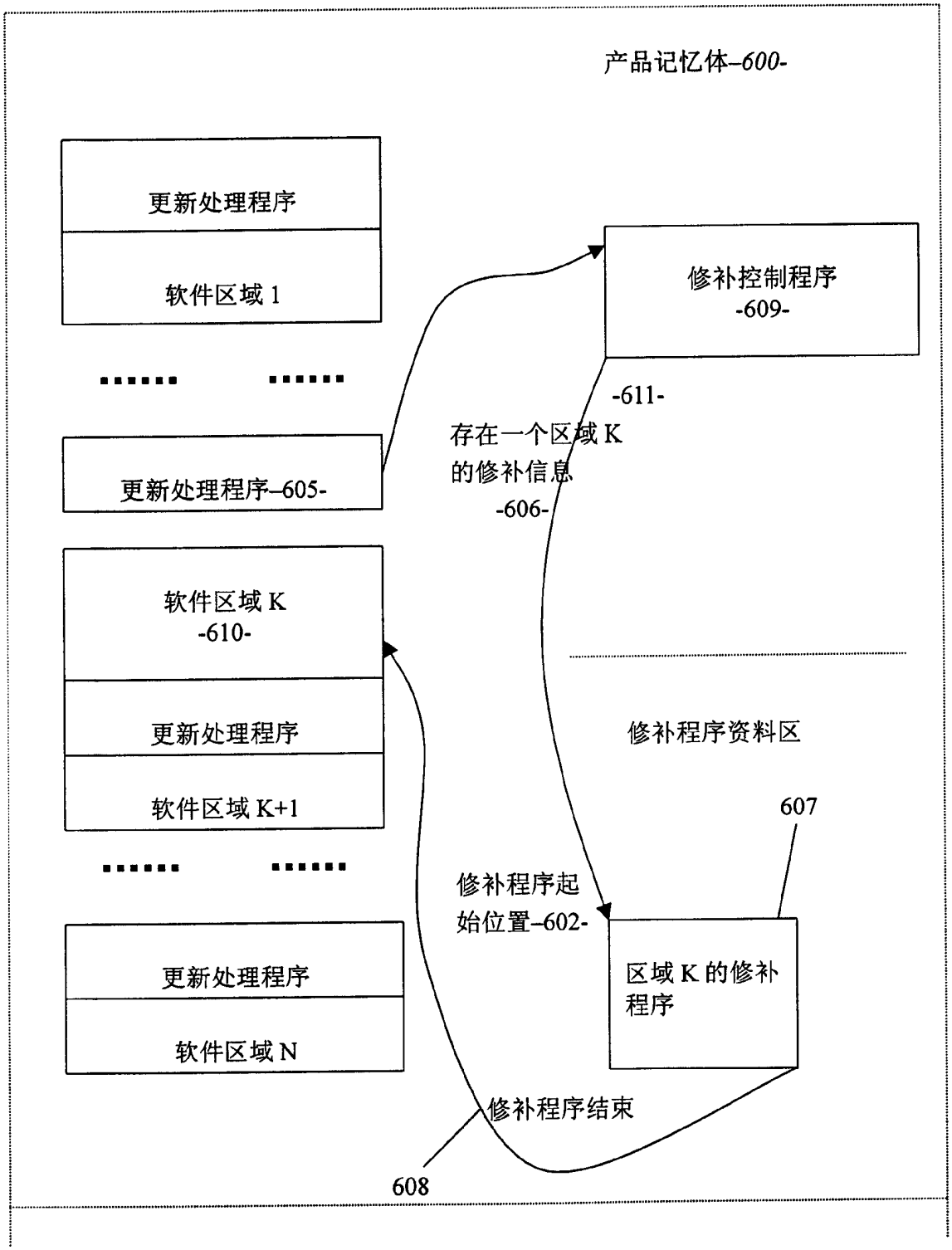

产品记忆体-600-
更新处理程序
软件区域 1
更新处理程序-605-
修补控制程序
-609-
-611-
存在一个区域 K
的修补信息
-606-
软件区域 K
-610-
更新处理程序
软件区域 K+1
修补程序资料区
607
修补程序起
始位置-602-
区域 K 的修补
程序
更新处理程序
软件区域 N
修补程序结束
608

图 6

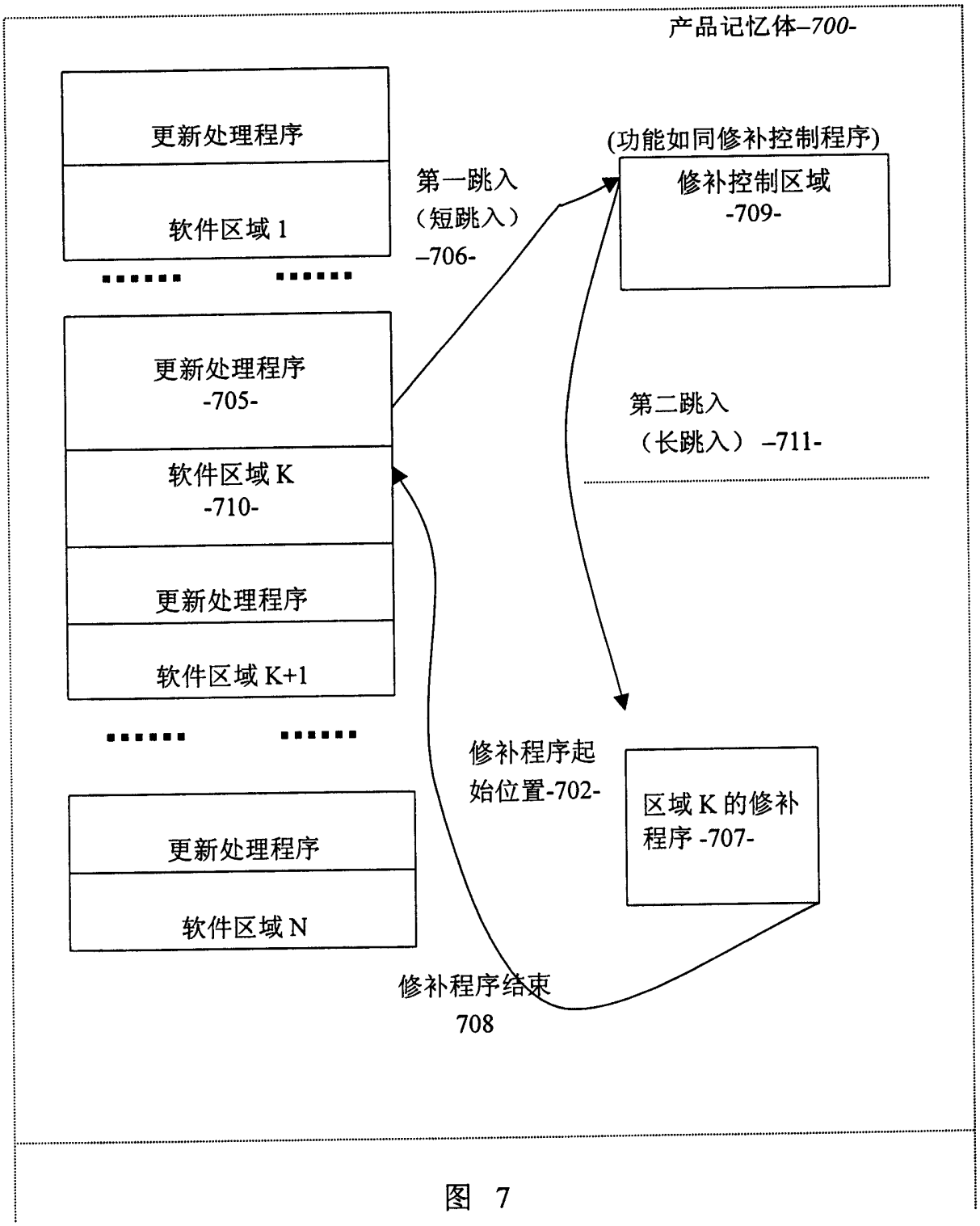

产品记忆体–700-
更新处理程序
软件区域 1
(功能如同修补控制程序)
第一跳入
（短跳入）
–706-
修补控制区域
-709-
更新处理程序
-705-
第二跳入
（长跳入） –711-
软件区域 K
-710-
更新处理程序
软件区域 K+1
修补程序起
始位置-702-
区域 K 的修补
程序 -707-
更新处理程序
软件区域 N
修补程序结束
708

图 7

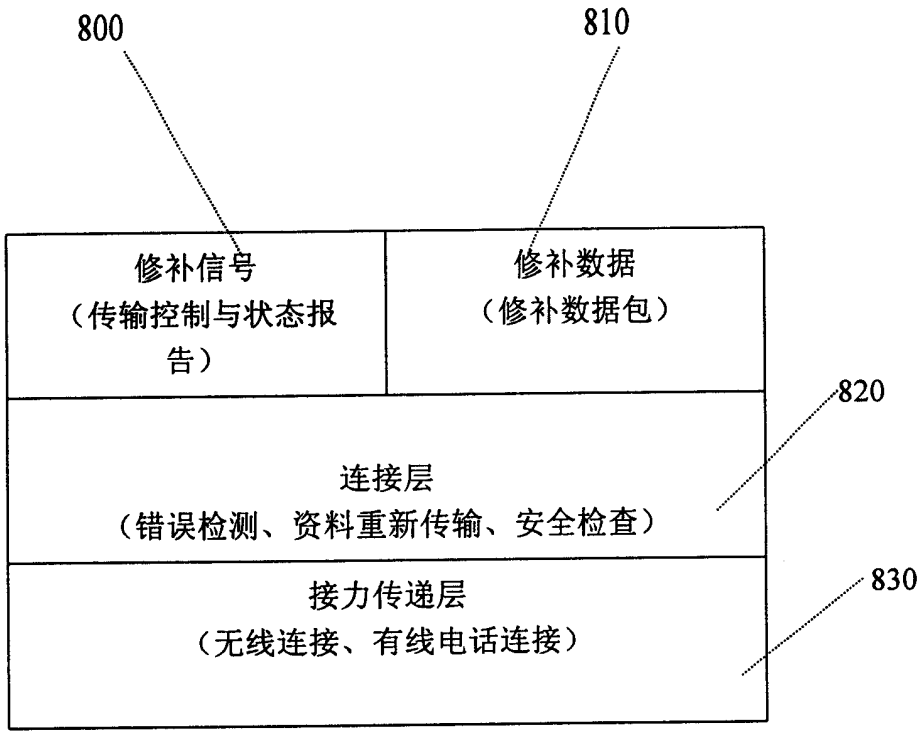

800
810
修补信号
（传输控制与状态报告）
修补数据
（修补数据包）
820
连接层
（错误检测、资料重新传输、安全检查）
830
接力传递层
（无线连接、有线电话连接）

图 8

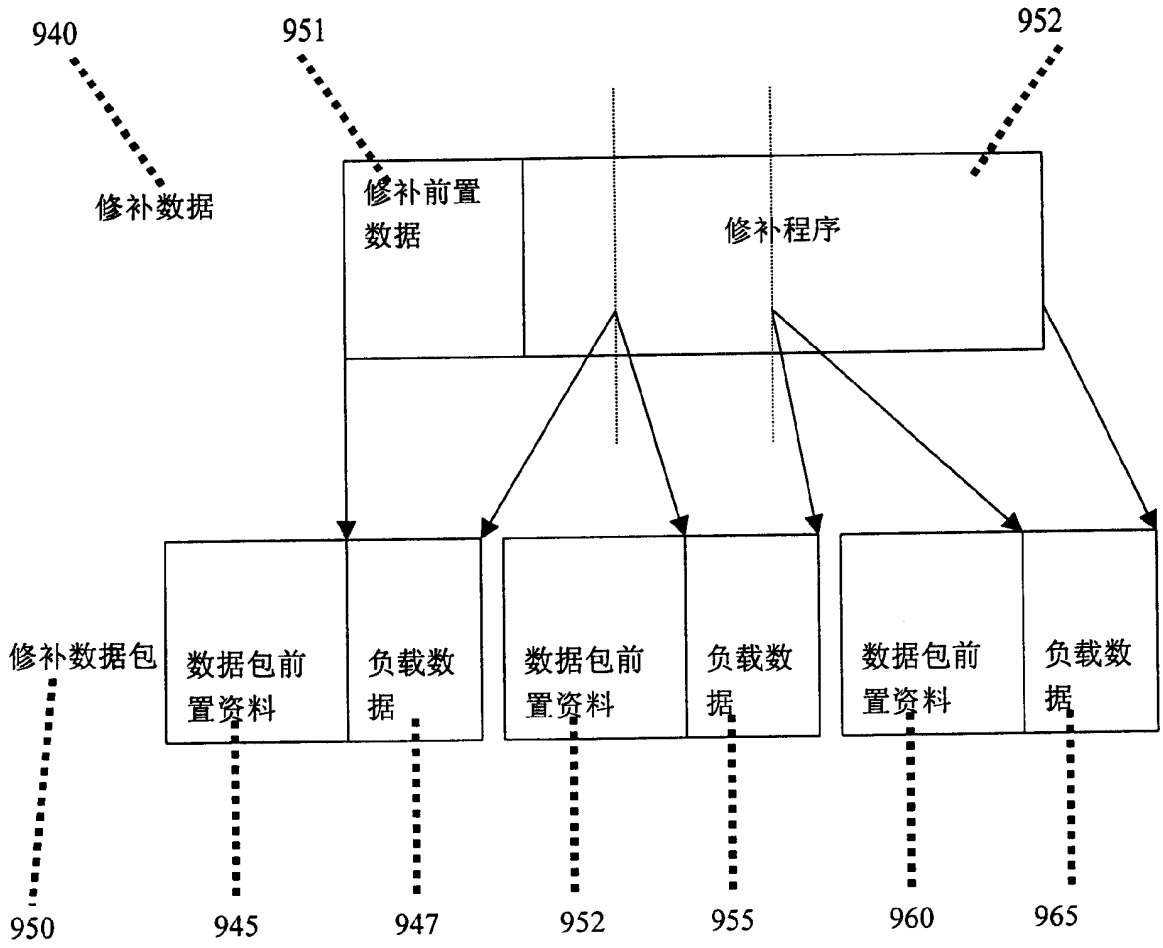

940
951
952
修补数据
修补前置数据
修补程序
修补数据包
数据包前置资料
负载数据
数据包前置资料
负载数据
数据包前置资料
负载数据
950
945
947
952
955
960
965

图 9

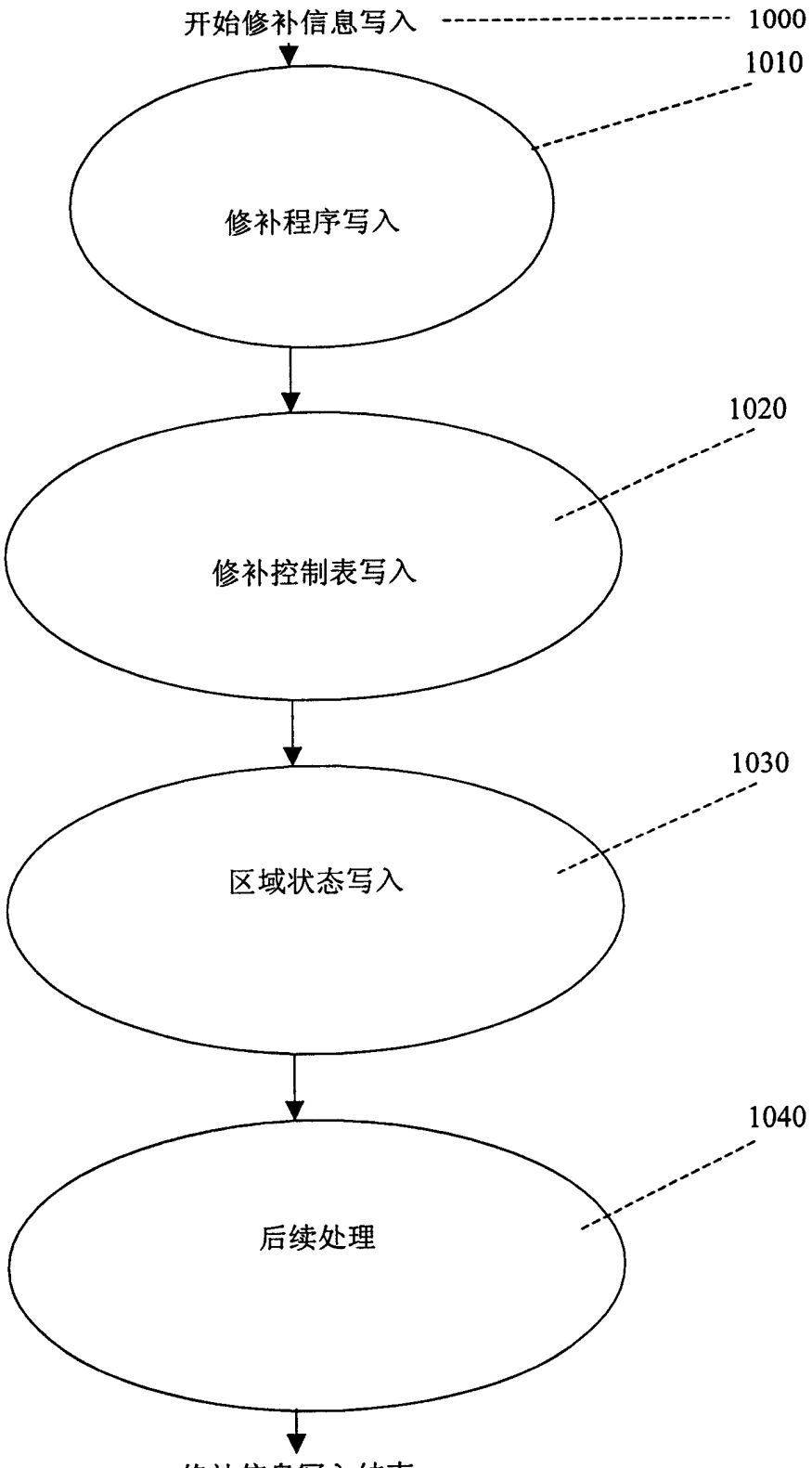

开始修补信息写入
1000
1010
修补程序写入
1020
修补控制表写入
1030
区域状态写入
1040
后续处理
修补信息写入结束

图 10

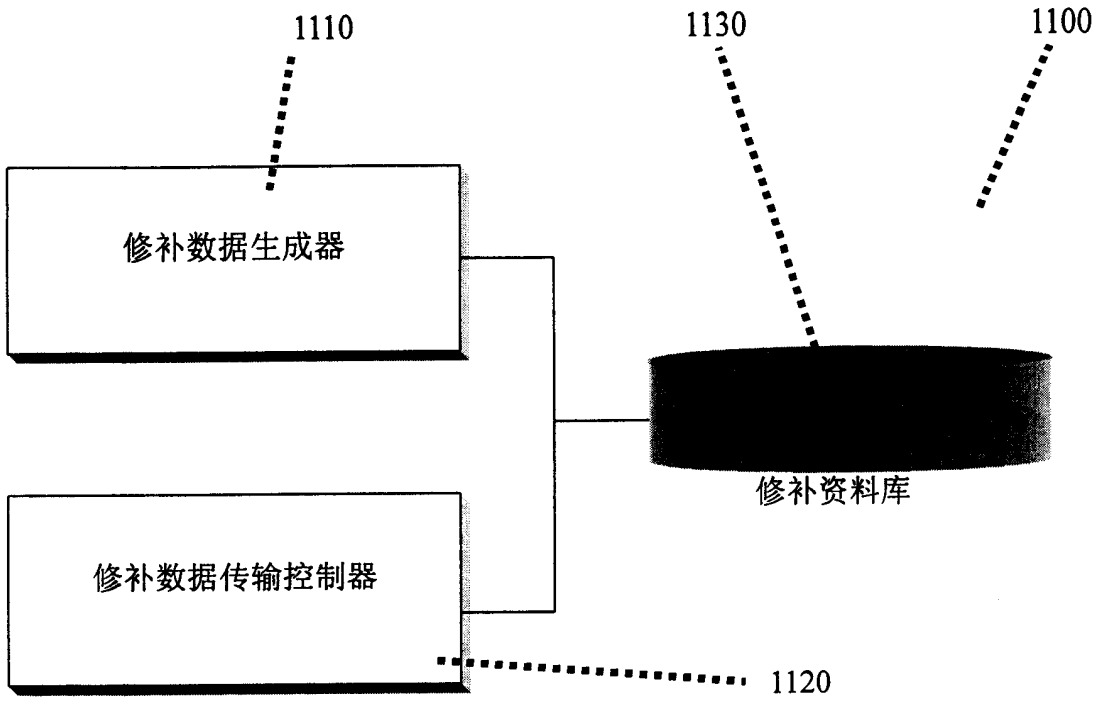


图 11

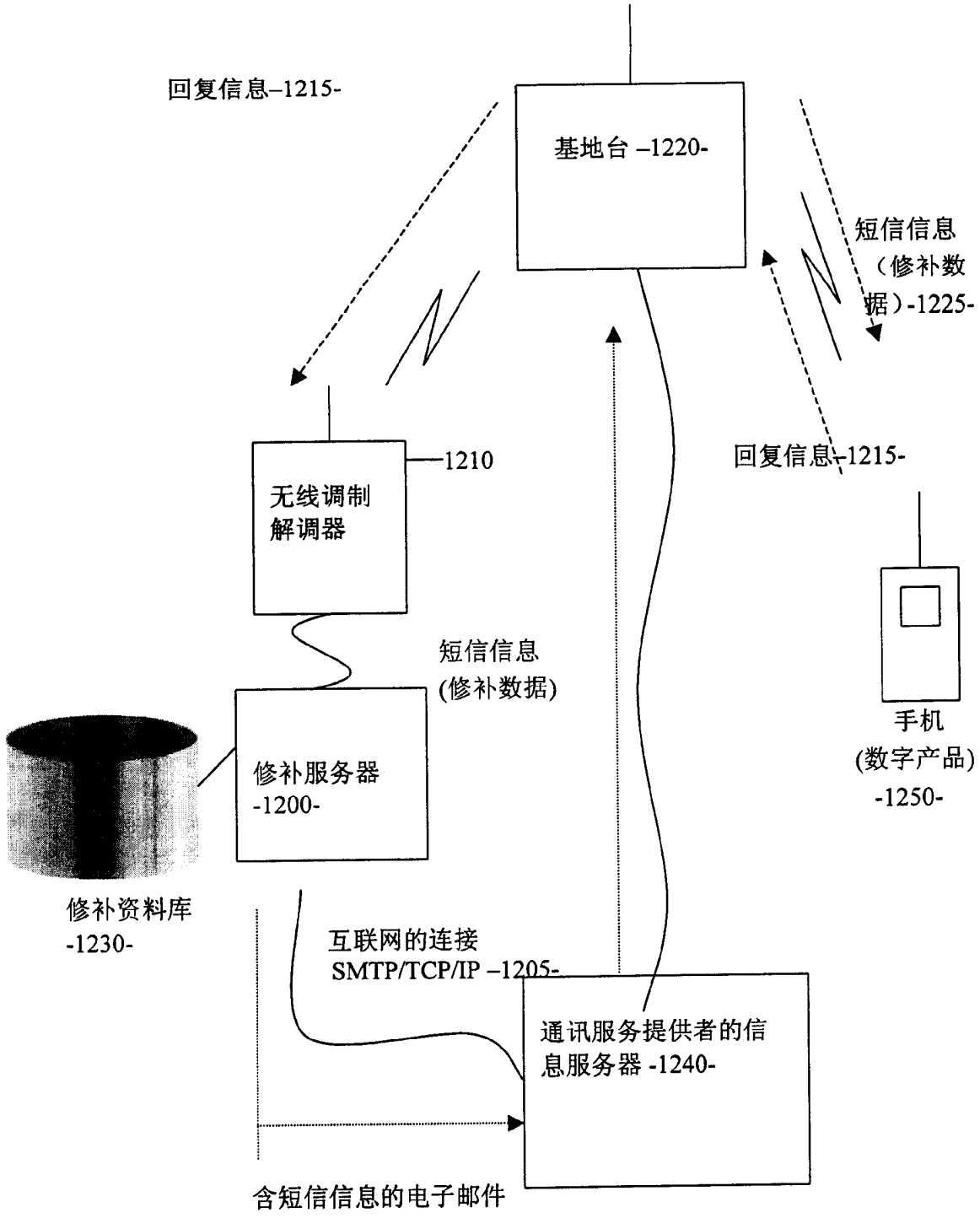

回复信息–1215-
基地台 –1220-
短信信息
（修补数
据）-1225-
回复信息–1215-
1210
无线调制
解调器
短信信息
(修补数据)
手机
(数字产品)
-1250-
修补服务器
-1200-
修补资料库
-1230-
互联网的连接
SMTP/TCP/IP –1205-
通讯服务提供者的信
息服务器 -1240-
含短信信息的电子邮件

图 12