(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
8 September 2006 (08.09.2006)

PCT

(10) International Publication Number
WO 2006/093327 A1

(51) International Patent Classification:
G01R 31/319 (2006.01)

(21) International Application Number:
PCT/JP2006/304517

(22) International Filing Date: 2 March 2006 (02.03.2006)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
11/070,957 2 March 2005 (02.03.2005) US

(71) Applicant *(for all designated States except US)*: **ADVANTEST CORPORATION** [JP/JP]; 1-32-1, Asahi-cho, Nerima-ku, Tokyo, 1790071 (JP).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **PRAMANICK, Ankan** [US/US]; 5839 Capilano Drive, San Jose, California, 95318 (US). **ELSTON, Mark** [US/US]; 1777 Merlot Way, Salinas, California, 93906 (US). **ADACHI, Toshiaki** [JP/US]; 1196 Nikette Way, San Jose, California, 95120 (US).

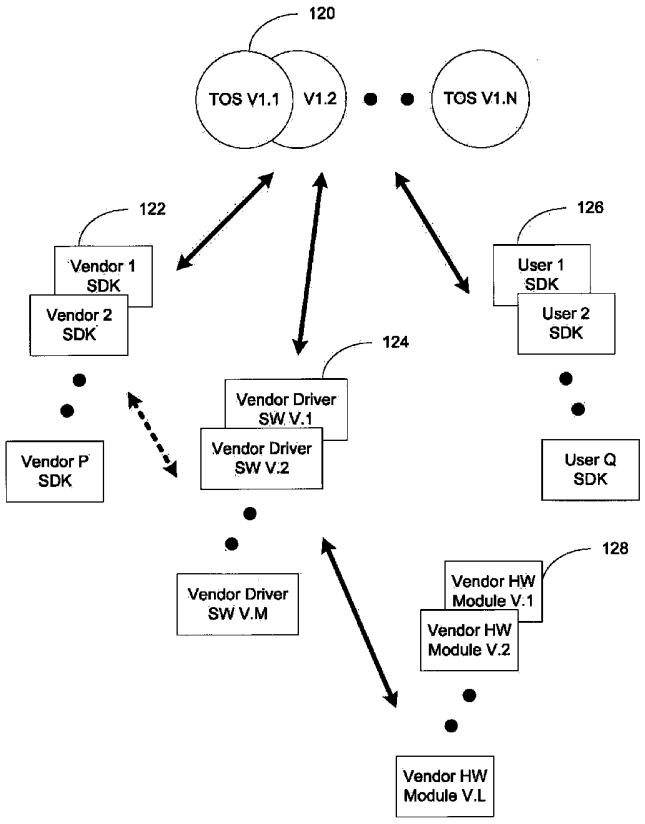
(74) Agent: **RYUKA, Akihiro**; 5F, Shinjuku Square Tower, 22-1, Nishi-Shinjuku 6-chome, Shinjuku-ku, Tokyo 163-1105 (JP).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT,

[Continued on next page]

(54) Title: COMPACT REPRESENTATION OF VENDOR HARDWARE MODULE REVISIONS IN AN OPEN ARCHITECTURE TEST SYSTEM



(57) Abstract: Method and system for associating software components with vendor hardware module versions in an open architecture test system are disclosed. The method includes receiving a set of hardware versions of a vendor hardware module, receiving a set of software components supported by the vendor hardware module, processing the set of hardware versions, where the set of hardware versions is represented as an equivalence class of hardware version numbers using a mask value, obtaining user choices of hardware versions of the vendor hardware module, validating the user choices of hardware versions of the vendor hardware module, and creating a system profile in accordance with the user choices of hardware versions.

WO 2006/093327 A1

RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

DESCRIPTION

Compact Representation of Vendor Hardware Module Revisions in an Open Architecture Test System

FIELD OF THE INVENTION

[0001] The present invention relates to the field of automated test equipment (ATE) for semiconductor testing. In particular, the present invention relates to a compact representation of vendor hardware module versions in an open architecture test system for efficient field deployment.

BACKGROUND OF THE INVENTION

[0002] In an open architecture test system, software needs to have access to vendor specifications for the list of vendor hardware module versions that are guaranteed to work with each version of the accompanying module software, for every vendor hardware module with which the user's particular system is configured, and for every different associated module software component for each such vendor hardware module. The system is then required to analyze this complex information efficiently to determine the choices for software components it should present to the user as candidates for the software content of the system.

[0003] Ideally, the presented choices for software components should comprise as large a set as possible, within the overall constraint that any complete set of software components the user picks from the given choices are guaranteed to work together, and work with the user's chosen hardware configuration. In the context of this problem, one method is to list compatible hardware versions for each software component, such as a comma-separated list of integral numbers, to represent the hardware version numbers (with range specifiers for denoting contiguous numbers). This method has a number of problems.

[0004] First, hardware version numbers are often large; requiring as many as 32 bits, where different bit-fields within the numbers may have different proprietary significance to the hardware vendor. If the vendor chooses to certify all versions with all possible values for a particular pattern of bits within the number as being compatible with a single software version, this may lead to: a) a non-contiguous sequence of version numbers, which cannot be efficiently

represented through range specifiers; and/or b) a very large number of version numbers, yielding a very long list, which may be quite impractical for storage and/or processing purposes. Second, the method is quite restrictive, especially in the case of very frequent hardware product updates, which may call for frequent additions to the list, and consequently, a tedious and frequent process of regeneration of system profiles at a user's site to incorporate the additions. Therefore, there is a need for a compact representation of vendor hardware module versions in an open architecture test system for efficient field deployment.

SUMMARY

[0005] A method is disclosed such that a hardware module vendor can specify, in a compact manner, an arbitrary sequence of hardware version numbers that are guaranteed to work with a given version of the associated module software in an open architecture test system. In addition, the hardware version numbers are processed so as to provide the user with an optimal set of software component choices for the vendor hardware modules, which lead to a valid system configuration for deployment in the field.

[0006] In one embodiment, a method for associating software components with vendor hardware module versions in an open architecture test system includes receiving a set of hardware versions of a vendor hardware module, receiving a set of software components supported by the vendor hardware module, processing the set of hardware versions, where the set of hardware versions is represented as an equivalence class of hardware version numbers using a mask value, obtaining user choices of hardware versions of the vendor hardware module, validating the user choices of hardware versions of the vendor hardware module, and creating a system profile in accordance with the user choices of hardware versions.

[0007] In another embodiment, an open architecture test system includes a System Controller, a Site Controller coupled to the System Controller, one or more hardware modules coupled to the Site Controller, and one or more devices-under-test (DUTs) serviced in parallel by groups of the hardware modules. The system further includes means for receiving a set of hardware versions of a vendor hardware module, means for receiving a set of software components supported by the vendor hardware module, means for processing the set of hardware versions, where the set of hardware versions is represented as an equivalence class of hardware

version numbers using a mask value, means for obtaining user choices of hardware versions of the vendor hardware module, means for validating the user choices of hardware versions of the vendor hardware module, and means for creating a system profile in accordance with the user choices of hardware versions.

[0008] In yet another embodiment, a method for representing vendor hardware module versions in an open architecture test system includes receiving a set of hardware versions of a vendor hardware module and representing the set of hardware versions as an equivalence class of hardware version numbers using a mask value including a mask number and a mask. The mask is a data pattern within the range of the set of hardware versions and the data pattern includes at least a don't-care bit and a unmask bit.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] The aforementioned features and advantages of the invention as well as additional features and advantages thereof will be more clearly understandable after reading detailed descriptions of embodiments of the invention in conjunction with the following drawings.

[0010] Figure 1a illustrates an open architecture test system for parallel testing of multiple DUTs with one or more SiteCs according to an embodiment of the present invention.

[0011] Figure 1b illustrates an example of multiple vendor hardware module versions and their corresponding software component versions in an open architecture test system.

[0012] Figure 2 illustrates an application of a mask to a masked number according to an embodiment of the present invention.

[0013] Figure 3 illustrates a method for creating a system profile for testing different vendor hardware module versions in an open architecture test system according to an embodiment of the present invention.

[0014] Figure 4 illustrates a method for computing intersections of masked representations according to an embodiment of the present invention.

DESCRIPTION OF EMBODIMENTS

[0015] Methods and systems are provided for a compact representation of vendor hardware module versions in an open architecture test system. The following descriptions are presented to enable any person skilled in the art to make and use the invention. Descriptions of specific embodiments and applications are provided only as examples. Various modifications and combinations of the examples described herein will be readily apparent to those skilled in the art, and the general principles defined herein may be applied to other examples and applications. Thus, the present invention is not intended to be limited to the examples described and shown, but is to be accorded the widest scope consistent with the principles and features disclosed herein.

[0016] A re-configurable open architecture test system provides a solution under which third party software and hardware (also known as vendor hardware modules) can be developed, certified individually, and integrated reliably into the open architecture test system. Figure 1a illustrates an open architecture test system for parallel testing of multiple DUTs with one or more SiteCs according to an embodiment of the present invention. The system includes a System Controller 102, a plurality of Site Controllers 104, a plurality of vendor hardware modules 106, and the corresponding group of DUTs 110; each group of DUTs may include one or more individual DUTs 112. The test head 108 holds the group of DUTs 110. In one embodiment, the System Controller and the Site Controller(s) may be implemented by a distributed computing system with multiple computers in multiple geographical locations. In another embodiment, the System Controller and the Site Controller may be implemented by a single computer. A detailed description of an open architecture test system is provided by U.S. application no. 60/447,839, "Method and Structure to Develop a Test Program for Semiconductor Integrated Circuits," and by U.S. application no. 60/573,577, "Software Development in an Open Architecture Test System." In addition, an open architecture test system is also described in "Testing Programming Environment in a Modular Open Architecture Test System," by A. Pramanick et al., in *2004 IEEE International Test Conference*, October 2004, pp. 413-422.

[0017] In such an open architecture test system, each modular unit, both software and hardware, is replaceable by third party units, as long as the third party solutions conform to the

requirements of the integrating framework. Thus, third party hardware may be any functional unit, such as a digital pin card, an analog card, a device power supply, etc. The term vendor hardware module (or module for short) is used to refer to such a hardware functional unit, while the associated vendor software components in support of the vendor hardware module are referred to as the module software. The latter comprise several distinct software components, such as module control driver software, calibration and diagnostic software, emulation software, vendor-specific pattern compilers, etc.

[0018] An open architecture test system provides not only for the operation of the test system, but also for controlling the test environment and introduction of new vendor hardware modules, along with all their accompanying software. Figure 1b illustrates an example of multiple vendor hardware module versions 128 and their corresponding software component versions in an open architecture test system. The software components include multiple versions of the Tester Operating System (TOS) 120 and its associated versions of Vendor software development kits (SDKs) 122 and User SDKs 126, as well as multiple versions of Vendor software drivers 124. A method for managing multiple vendor hardware module versions and software component versions such as TOSs, Vendor SDKs, and User SDKs, in an open architecture test system is provided by U.S. application no. 60/635,094, "Method and System for Performing Installation and Configuration Management of Tester Instrument Modules," which is assigned to the assignee of the present application.

[0019] The test system is required to be flexible enough such that a particular version n of the module software, for a vendor hardware module M, can be verified by the test system as being compatible with a particular system configuration, as long as the vendor of module M has certified that version n of the software is compatible with certain different versions of vendor hardware modules that may co-exist in that given system configuration. The vendor is required to publish a list of hardware versions for module M that are guaranteed to work with the version n of the module software for M.

[0020] When a user configures an open architecture test system with particular versions of the vendor hardware modules and their associated module software, it is up to the system software to verify that the user's choice has actually resulted in a working system. For this

reason, such an open architecture test system needs to rely on system profile information that is created by the user, with proper guidance from the system to the user during profile creation that constrains the user to make only valid choices for system configuration, both hardware and software, and given that a large system can have scores of different vendor hardware modules, each with multiple associated software components. Thus, the task of providing proper guidance to the user to create a system profile, while at the same time not restricting the user unnecessarily, is very complicated.

[0021] In general, the method of providing a compact representation of vendor hardware module versions in an open architecture test system for efficient field deployment includes two main parts: 1) specifying the hardware versions; and 2) processing the hardware versions information and allowing the user to create valid system profiles. A person skill in the art would recognize that the method may be applied to represent software component versions, such as versions of TOSs, Vendor SDKs, and User SDKs, in an open architecture test system.

[0022] In one embodiment, compatible hardware version specification is accomplished by masking an arbitrary set of (vendor-specified) bits in the hardware version number. For instance, the syntax of a hardware version number may be represented as follows in a specification file: 1095188784, 0x01403500 / 0x0ff0ff00, 8800000111, where the numbers "1095188784" and "8800000111" denote non-masked, single hardware versions, while the number "0x01403500 / 0x0ff0ff00" denotes a masked number, which represents an entire family (class) of equivalent hardware version numbers.

[0023] In this example, a masked value is represented in hexadecimal format, in the syntax "n / m", where 'n' represents a masked number, and 'm' represents a mask. The mask m for a masked number n, in the expression "n / m", is defined as a bit-pattern that specifies an equivalence class (EQC) of numbers based on n, denoted "EQC(n / m)", such that 1) for every bit that is a don't-care (represented by a "0" bit in this example) in m, the corresponding bit in n is masked, indicating that all members of the equivalence class can have either a 1 or a 0 in that bit position; and 2) for every bit that is a unmasked bit (represented by a "1" bit in this example) in m, the corresponding bit in any number in the equivalence class is identical to the corresponding bit in n.

[0024] Figure 2 illustrates an application of a mask to a masked number according to an embodiment of the present invention. Consider the tuple "010/110" (expressed in binary format instead of hexadecimal format for ease of exposition) in Figure 2. A mask m 202 is applied to the number n 204 to create an equivalence class of version numbers 206.

[0025] Applying the definition of a mask given above, a number x, such that x is a member of EQC(010/110), has '1' and '0' in bit positions 2 and 3 respectively (counting the three bits from right, or least significant, to left, or most significant). This is because bit positions 2 and 3 in the mask "110" are 1s (i.e., not masked), which implies that in x, these two bits have the exact values of the corresponding bits in n. However, x can have either a '1' or a '0' in bit position 1, since the value of bit-1 in the mask "110" is '0' (i.e., masked). Thus, EQC(010/110) is the set {010, 011}, i.e., the set {2, 3}, and hence, x can be either 2 or 3.

[0026] In an open architecture test system, a commonly performed operation is to determine whether a hardware version number x provided by the user is a member of the masked equivalence class. One way to determine membership in a masked equivalence class is to show that the number x is a member of EQC(n / m) if and only if x & m = n, where "&" is the bitwise-AND operator. This property provides a convenient way of performing the required determination.

[0027] Next, a method for generating a system profile with a masked representation of the equivalence class of a large number of non-contiguous version numbers is described. In general, the method includes 1) processing the set of masked hardware versions; 2) providing the user with a choice of compatible software component versions to run the user's chosen version of the vendor hardware module; and 3) computing the compact representation of the set of hardware versions for each vendor hardware module that results from the user's selection of compatible software component versions for the modules; and 4) storing the compact representation in the user's system profile.

[0028] For processing the set of masked hardware versions, the notation used for representing the versions of vendor hardware modules and their associated software components is described as follows. First, let a particular vendor hardware module be denoted as M, and let the software components that are required for controlling the hardware module M in the ATE be

denoted by the set {MX}, where X can be A, B, C, ... etc. Thus, for example, if M has a total of four software components, they are represented as MA, MB, MC and MD. Next, let a particular version k of a software component X for the vendor hardware module M be denoted as MX.k. For example, version 1 of the component A for module M is denoted as MA.1. Moreover, the set of hardware versions supported by the k^{th} version of the software component X for the vendor hardware module M is denoted as HWR(MX.k). For example, HWR(MA.1) represents the set {3, 5, 010/011}. Note that for simplicity, the masked numbers are represented in binary format. A person of ordinary skill in the art would recognize that the masked numbers may be represented in hexadecimal or other formats.

[0029] Figure 3 illustrates a method for creating a system profile for testing different vendor hardware module versions in an open architecture test system according to an embodiment of the present invention. The method starts in the Begin state (302) and thereafter moves to Step 1 (304) where the method queries the user for choice(s) of hardware version numbers for module M in the test system. As discussed above, the use of arbitrarily masked values for hardware versions and the resulting families of hardware version numbers, create a challenge in representing the equivalence class EQC(n/m) to the user. A complete enumeration (and subsequent maintenance) of all possible values x, such that x is a member of EQC(n/m), can be exponentially complex, especially in view of the number of masked bits specified in m. For even moderately large values of m (e.g. for m is less than or equal to 32), the method of complete enumeration is clearly inefficient and impractical. Hence, the user is prompted to enter his choice of hardware version value(s).

[0030] In Step 2 (306), the method determines whether each user entry is a valid choice. The determination is performed using the method described above in determining whether the hardware version number is a member of the EQC. If the user input(s) are valid (306_yes), the method continues to Step 3. If not (306_no), the method repeats Step 1 and queries the user to re-enter a hardware version number.

[0031] In Step 3 (308), upon a successful user entry of the hardware version number(s), the method determines a set of compatible versions of software components k of X compatible with HWR(MX.k) (the user's choices of hardware version(s) for M). In each step, the method

discussed above for testing the hardware version number(s) against masked values is used. For each of the user's choices, if it is covered by a masked value, the entire equivalence class corresponding to that masked value is deemed to have been chosen. This property is important for achieving effective compact representation of the hardware version numbers.

[0032] In Step 4 (310), the method queries the user for a choice of one version of software component from the set of compatible software components determined above.

[0033] In Step 5 (312), the method computes the intersection, across all X, of HWR(MX.p), where p denotes the chosen version of software component from X, for each hardware version choice from Step 1. A detailed discussion of computation of intersections is given below in association with the discussion of Figure 4. This step evaluates the overall list of all compatible hardware versions by computing the intersection of all the sets HWR(MX.p), for each user-chosen version p of the software component X. This step repeats this computation for each of the user's hardware version choices (with each user choice resulting in a single value or single EQC), and collects the results.

[0034] In step 6 (314), the method computes the final list of hardware versions for M, which are compatible with the user's software version choices, using the collection of the results from Step 5 across all chosen hardware versions.

[0035] In Step 7 (316), the method stores the final list computed above in the user's profile. The method ends in the End state (318).

[0036] Figure 4 illustrates a method for computing intersections of masked representations according to an embodiment of the present invention. As shown in Figure 4, the intersection of two components, m_I and m_J, assuming that each choice is a masked one (i.e., an entire equivalence class), is computed. A person of ordinary skill in the art would recognize that it is possible to extend the method for computing intersections to non-masked vs. masked or non-masked vs. non-masked choices of hardware version numbers. As shown in Figure 4, the user choice for m_I is represented by EQC(n_I / m_I), and the user choice for m_J is represented by EQC(n_J / m_J). Let the result of the intersection be EQC(n / m).

[0037] The method starts in the Begin state (402), and thereafter moves to Step 1 (404), where the method computes the resulting mask. The symbol "|" represents a bitwise-OR operation. The result of a bitwise-OR of m_I and m_J is indeed the resulting mask, since it is, by definition of the bitwise-OR operation and the definition of masks, the only mask that is satisfied by both m_I and m_J. Note that if m is empty (i.e., all-1s), then EQC(n / m) consists of a single value n. In Step 2 (406), the method computes the resulting masked number, n. The method ends in the End state (408). A person of ordinary skill in the art would recognize that to extend the above computation of intersections to more than two components, one may use the values of n and m computed in Step 2 as the starting values for the next iteration in the computation, and so on.

[0038] The following example applies the methods described in Figures 3 and 4 above. Assume that a particular vendor hardware module M has three software components: component MA, component MB and component MC. Also assume that the set of choices a particular user has for available software versions of these components, and their corresponding supported hardware versions, is given in Table 1, where the notation "MX.n" signifies the n^{th} version of the software component MX. For ease of exposition, the masked numbers are represented in binary instead of hexadecimal. In addition, it is assumed that the set of all possible hardware versions can be represented by three bits.

SW Component for Module *M*	**Hardware Versions Supported by SW Component**
MA.1	3, 5, 010/011
MA.2	5, 000/100
MA.3	000/110
MB.1	010/010, 5
MB.2	000/000
MC.1	5, 010/110

Table 1

[0039] Note that the ultimate goal is to allow the user to choose *exactly one* version of *each* software component available (i.e., one version each of *MA*, *MB* and *MC*) for working with the choice of hardware version(s) for the module *M*.

Applying the method of Figure 3 to the above example:

1. The user choices are tested against the entire set $MX.n$, i.e.,

   {3, 5, 010/011, 000/100, 000/110, 010/010, 000/000, 010/110}

   Suppose the user chooses the hardware versions '2' and '5' for *M*. This results in the following associations:

   a. For hardware version '2', the associations are (i.e., the sets $HWR(MX.n)_2$ are):

      $MA.1 \Leftrightarrow$ 010/011

      $MA.2 \Leftrightarrow$ 000/100

      $MA.3 \Leftrightarrow \emptyset$ ('2' is not supported by *MA*.3)

      $MB.1 \Leftrightarrow$ 010/010

      $MB.2 \Leftrightarrow$ 000/000

      $MC.1 \Leftrightarrow$ 010/110

   b. For hardware version '5', the associations are (i.e., the sets $HWR(MX.n)_5$ are):

      $MA.1 \Leftrightarrow$ 5

      $MA.2 \Leftrightarrow$ 5

      $MA.3 \Leftrightarrow \emptyset$ ('5' is not supported by *MA*.3)

      $MB.1 \Leftrightarrow$ 5

      $MB.2 \Leftrightarrow$ 000/000

      $MC.1 \Leftrightarrow$ 5

2. The above associations dictate that the available software component choices for the user are

   one of {*MA*.1, *MA*.2},

   one of {*MB*.1, *MB*.2}, and

one of {*MC*.1}.

3. Suppose the user (selecting one from each component group in 2 above) chooses the following: *MA*.2, *MB*.2 and *MC*.1. Then,

   a. For hardware version choice '2', the list of supported hardware versions is given by the *intersection* of the sets $HWR(MA.2)_2$, $HWR(MB.2)_2$ and $HWR(MC.1)_2$ according to the following steps:

      i. compute the intersection of $HWR(MA.2)_2$ with $HWR(MB.2)_2$, using the method of Figure 4 as follows: 000/100 ∩ 000/000 ⇒ 000/100. (Where 000 & 100 = 000, and 000 & 000 = 000.)

      ii. compute the intersection of 000/100 with $HWR(MC.1)_2$: 000/100 ∩ 010/110 ⇒ 010/110. (Where 010 & 100 = 000, and 010 & 110 = 010.)

      iii. thus, the final equivalence class for the choice of hardware version '2' is 010/110, i.e., {2, 3}.

   b. For hardware version choice '5', the list of supported hardware versions is given by the intersection of the sets $HWR(MA.2)_5$, $HWR(MB.2)_5$ and $HW(MC.1)_5$:

      i. First, compute the intersection of $HWR(MA.2)_5$ with $HWR(MB.2)_5$, using the method of Figure 3 as follows: 5 ∩ 000/000 ⇒ 101/111 ∩ 000/000 ⇒ 101/111.

      ii. Then, compute the intersection of 101/111 with $HWR(MC.1)_5$: 101/111 ∩ 5 ⇒ 101/111 ∩ 101/111 ⇒ 101/111 ⇒ 5.

      iii. Thus, the final value for the choice of hardware version '5' is 5.

4. From 3a and 3b above, applying Step 6 in Figure 3, the list of supported hardware versions to be included in the user's profile for its choice of software components *MA*.2, *MB*.2 and *MC*.1 is therefore {010/110, 5}, i.e., {2, 3, 5}.

[0040] Therefore, a hardware module vendor can specify an arbitrary sequence of hardware version numbers that are guaranteed to work with a given version of the associated module software in an open architecture test system. In addition, the hardware version numbers are processed so as to provide the user with an optimal set of software component choices for the

vendor hardware module, which leads to a valid system configuration for deployment in the field.

[0041] The present invention provides at least the following benefits. First, a compact representation of the supported module hardware versions is employed, representing an entire family of supported actual product versions for the vendor hardware module. Second, the disclosed method allows the vendor to publish a hardware version specification for a compatible software component that simultaneously supports all the hardware versions implicitly indicated by the specified value. Third, an arbitrary bit pattern can be specified by the vendor, so that the vendor is not restricted to choosing just contiguous version numbers. Fourth, the vendor-specified value can be treated by the open architecture test system software as a compact collection of all the individual hardware version numbers that make up the set. Fifth, the disclosed method efficiently analyzes the multiple compatible hardware version lists for the numerous software modules that are associated with the user's chosen vendor hardware modules, and presents the user with an optimal set of compatible software components to choose from while guaranteeing correct system operation. Finally, the disclosed method computes and stores, in the user's system profile, the optimally compact representation of the entire family of equivalent hardware versions for each chosen module. Thus, replacing a vendor hardware module with another from the same family requires neither a re-release of the hardware version list, nor a re-generation of the system profile.

[0042] It will be appreciated that the above description for clarity has described embodiments of the invention with reference to different functional units and processors. However, it will be apparent that any suitable distribution of functionality between different functional units or processors may be used without detracting from the invention. For example, functionality illustrated to be performed by separate processors or controllers may be performed by the same processor or controllers. Hence, references to specific functional units are only to be seen as references to suitable means for providing the described functionality rather than indicative of a strict logical or physical structure or organization.

[0043] The invention can be implemented in any suitable form including hardware, software, firmware or any combination of these. The invention may optionally be implemented

partly as computer software running on one or more data processors and/or digital signal processors. The elements and components of an embodiment of the invention may be physically, functionally, and logically implemented in any suitable way. Indeed the functionality may be implemented in a single unit, in a plurality of units or as part of other functional units. As such, the invention may be implemented in a single unit or may be physically and functionally distributed between different units and processors.

[0044] One skilled in the relevant art will recognize that many possible modifications and combinations of the disclosed embodiments may be used, while still employing the same basic underlying mechanisms and methodologies. The foregoing description, for purposes of explanation, has been written with references to specific embodiments. However, the illustrative discussions above are not intended to be exhaustive or to limit the invention to the precise forms disclosed. Many modifications and variations are possible in view of the above teachings. The embodiments were chosen and described to explain the principles of the invention and their practical applications, and to enable others skilled in the art to best utilize the invention and various embodiments with various modifications as are suited to the particular use contemplated.

CLAIMS

1. A method for associating software components with vendor hardware module versions in an open architecture test system, comprising:

receiving a set of hardware versions of a vendor hardware module;

receiving a set of software components supported by the vendor hardware module;

processing the set of hardware versions, wherein the set of hardware versions is represented as an equivalence class of hardware version numbers using a mask value;

obtaining user choices of hardware versions of the vendor hardware module;

validating the user choices of hardware versions of the vendor hardware module; and

creating a system profile in accordance with the user choices of hardware versions.

2. The method of claim 1, wherein the set of software components comprise:

one or more vender software development kits;

one or more user software development kits; and

one or more vendor driver software versions corresponding to the set of hardware versions of the vendor hardware module.

3. The method of claim 1, wherein processing the set of hardware versions comprises:

defining a mask value, wherein the mask value comprises a masked number and a mask, wherein the mask is data pattern within the range of the set of hardware versions, and wherein the data pattern comprises at least a don't-care bit and a unmasked bits; and

specifying an equivalence class of hardware version numbers in accordance with the mask and the masked number.

4. The method of claim 3, wherein specifying the equivalence class of hardware version numbers comprises:

for each don't care bit in the mask, masking the corresponding bit in the masked number, thereby indicating all members of the of the equivalence class have either a 1 or 0 in the corresponding bit position; and

for each unmasked bit in the mask, using the corresponding bit of the masked number in the equivalence class, thereby indicating the corresponding bit in the masked number is not masked.

5. The method of claim 3, wherein validating the user choices of hardware versions of the vendor hardware module comprises:

if a user choice is covered by the mask value, identifying an entire equivalence class corresponding to the mask value as being chosen.

6. The method of claim 5, wherein validating the user choices of hardware versions of the vendor hardware module further comprises:

for each user choice of hardware version, representing the user choice of hardware version as valid if and only if a bitwise-AND of the user choice of hardware version and the mask equals to the masked number.

7. The method of claim 1, wherein creating a system profile comprises:

determining a set of compatible software components in accordance with the user choices of hardware versions of the vendor hardware module;

selecting a version of software component for each user choice of hardware versions of the vendor hardware module;

creating a list of user selected software components for each user choice of hardware versions of the vendor hardware module; and

storing the list of user selected software components.

8. The method of claim 7, wherein selecting a version of software component comprises:

querying user to choose a version of software component from the set of compatible software components; and

for each user choice of the version of software component, computing an intersection, across the user choice of hardware versions, for generating a list of supported hardware versions.

9. The method of claim 7, wherein computing an intersection comprises:

computing resulting mask using the user choice of hardware versions; and

computing resulting masked number using the user choice of hardware versions.

10. The method of claim 1 further comprising:

providing an open architecture test system, wherein the open architecture test system comprises a System Controller, one or more Site Controllers, one or more vendor hardware modules, and one or more devices-under-test (DUTs);

receiving a new hardware version of a vendor hardware module, wherein the new hardware version is a member of the equivalence class of hardware version numbers; and

testing the new hardware version of the vendor hardware module using the system profile.

11. An open architecture test system, comprising:

a System Controller;

a Site Controller coupled to the System Controller;

one or more hardware modules coupled to the Site Controller;

one or more devices-under-test (DUTs) coupled to a hardware module in parallel;

means for receiving a set of hardware versions of a vendor hardware module;

means for receiving a set of software components supported by the vendor hardware module;

means for processing the set of hardware versions, wherein the set of hardware versions is represented as an equivalence class of hardware version numbers using a mask value;

means for obtaining user choices of hardware versions of the vendor hardware module;

means for validating the user choices of hardware versions of the vendor hardware module; and

means for creating a system profile in accordance with the user choices of hardware versions.

12. The system of claim 11, wherein the set of software components comprise:

one or more vender software development kits;

one or more user software development kits; and

one or more vendor driver software versions corresponding to the set of hardware versions of the vendor hardware module.

13. The system of claim 11, wherein the means for processing the set of hardware versions comprise:

means for defining a mask value, wherein the mask value comprises a masked number and a mask, wherein the mask is data pattern within the range of the set of hardware versions, and wherein the data pattern comprises at least a don't-care bit and a unmasked bits; and

means for specifying an equivalence class of hardware version numbers in accordance with the mask and the masked number.

14. The system of claim 13, wherein the means for specifying the equivalence class of hardware version numbers comprise:

for each don't care bit in the mask, means for masking the corresponding bit in the masked number, thereby indicating all members of the of the equivalence class have either a 1 or 0 in the corresponding bit position; and

for each unmasked bit in the mask, means for using the corresponding bit of the masked number in the equivalence class, thereby indicating the corresponding bit in the masked number is not masked.

15. The system of claim 13, wherein the means for validating the user choices of hardware versions of the vendor hardware module comprise:

if a user choice is covered by the mask value, means for identifying an entire equivalence class corresponding to the mask value as being chosen.

16. The system of claim 15, wherein the means for validating the user choices of hardware versions of the vendor hardware module further comprise:

for each user choice of hardware version, means for representing the user choice of hardware version as valid if and only if a bitwise-AND of the user choice of hardware version and the mask equals to the masked number.

17. The system of claim 11, wherein the means for creating a system profile comprise:

means for determining a set of compatible software components in accordance with the user choices of hardware versions of the vendor hardware module;

means for selecting a version of software component for each user choice of hardware versions of the vendor hardware module;

means for creating a list of user selected software components for each user choice of hardware versions of the vendor hardware module; and

means for storing the list of user selected software components.

18. The system of claim 17, wherein the means for selecting a version of software component comprise:

means for querying user to choose a version of software component from the set of compatible software components; and

for each user choice of the version of software component, means for computing an intersection, across the user choice of hardware versions, for generating a list of supported hardware versions.

19. The system of claim 17, wherein the means for computing an intersection comprise:

means for computing resulting mask using the user choice of hardware versions; and

means for computing resulting masked number using the user choice of hardware versions.

20. The system of claim 11 further comprising:

means for providing an open architecture test system, wherein the open architecture test system comprises a System Controller, one or more Site Controllers, one or more vendor hardware modules, and one or more devices-under-test (DUTs);

means for receiving a new hardware version of a vendor hardware module, wherein the new hardware version is a member of the equivalence class of hardware version numbers; and

means for testing the new hardware version of the vendor hardware module using the system profile.

21. A method for representing vendor hardware module versions in an open architecture test system, wherein the open architecture test system comprises a System Controller, a Site Controller, one or more vendor hardware module, and one or more devices-under-test (DUTs), comprising:

receiving a set of hardware versions of a vendor hardware module;

representing the set of hardware versions as an equivalence class of hardware version numbers using a mask value comprising a mask number and a mask; and

wherein the mask is a data pattern within the range of the set of hardware versions; and

wherein the data pattern comprises at least a don't-care bit and a unmask bit.

22. The method of claim 21, wherein representing the hardware versions further comprises:

for each don't-care bit in the mask, masking the corresponding bit in the masked number, thereby indicating all members of the of the equivalence class have either a 1 or 0 in the corresponding bit position; and

for each unmasked bit in the mask, using the corresponding bit of the masked number in the equivalence class, thereby indicating the corresponding bit in the masked number is not masked.

23. The method of claim 21 further comprising:

obtaining user choices of hardware versions of the vendor hardware module; and

validating the user choices of hardware versions of the vendor hardware module.

24. The method of claim 23, wherein validating the user choices of hardware versions of the vendor hardware module further comprises:

for each user choice of hardware version, representing the user choice of hardware version as valid if and only if a bitwise-AND of the user choice of hardware version and the mask equals to the masked number; and

if a user choice is determined to be valid, identifying an entire equivalence class corresponding to the mask value as being chosen.

25. The method of claim 21 further comprising:

receiving a new hardware version of the vendor hardware module, wherein the new hardware version is a member of the equivalence class of hardware version numbers; and

testing the new hardware version of the vendor hardware module using the equivalence class of hardware version numbers.

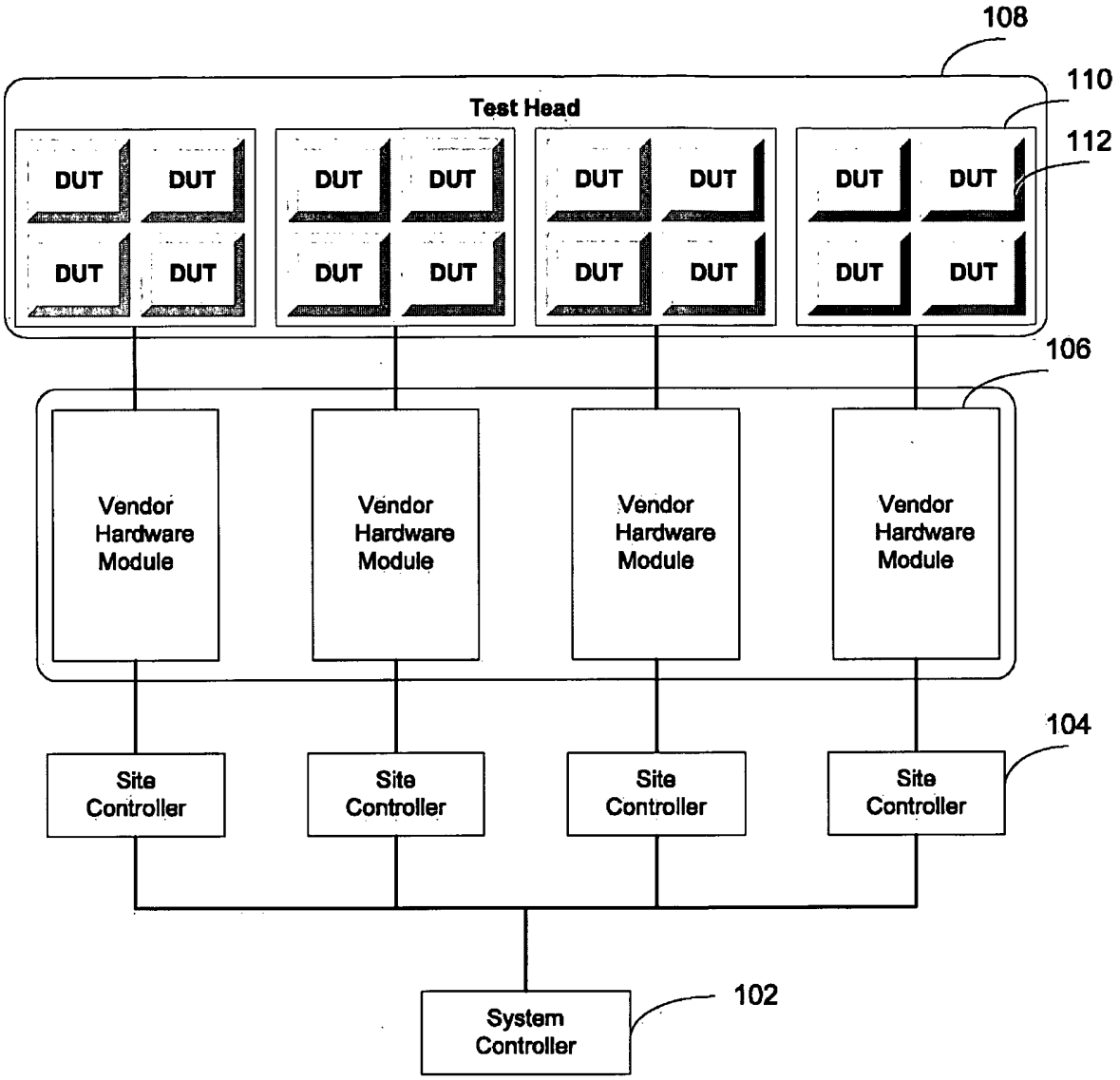

108
110
112
Test Head
DUT
DUT
DUT
DUT
DUT
DUT
DUT
DUT
DUT
DUT
DUT
DUT
DUT
DUT
DUT
DUT
106
Vendor Hardware Module
Vendor Hardware Module
Vendor Hardware Module
Vendor Hardware Module
104
Site Controller
Site Controller
Site Controller
Site Controller
102
System Controller

FIG. 1a

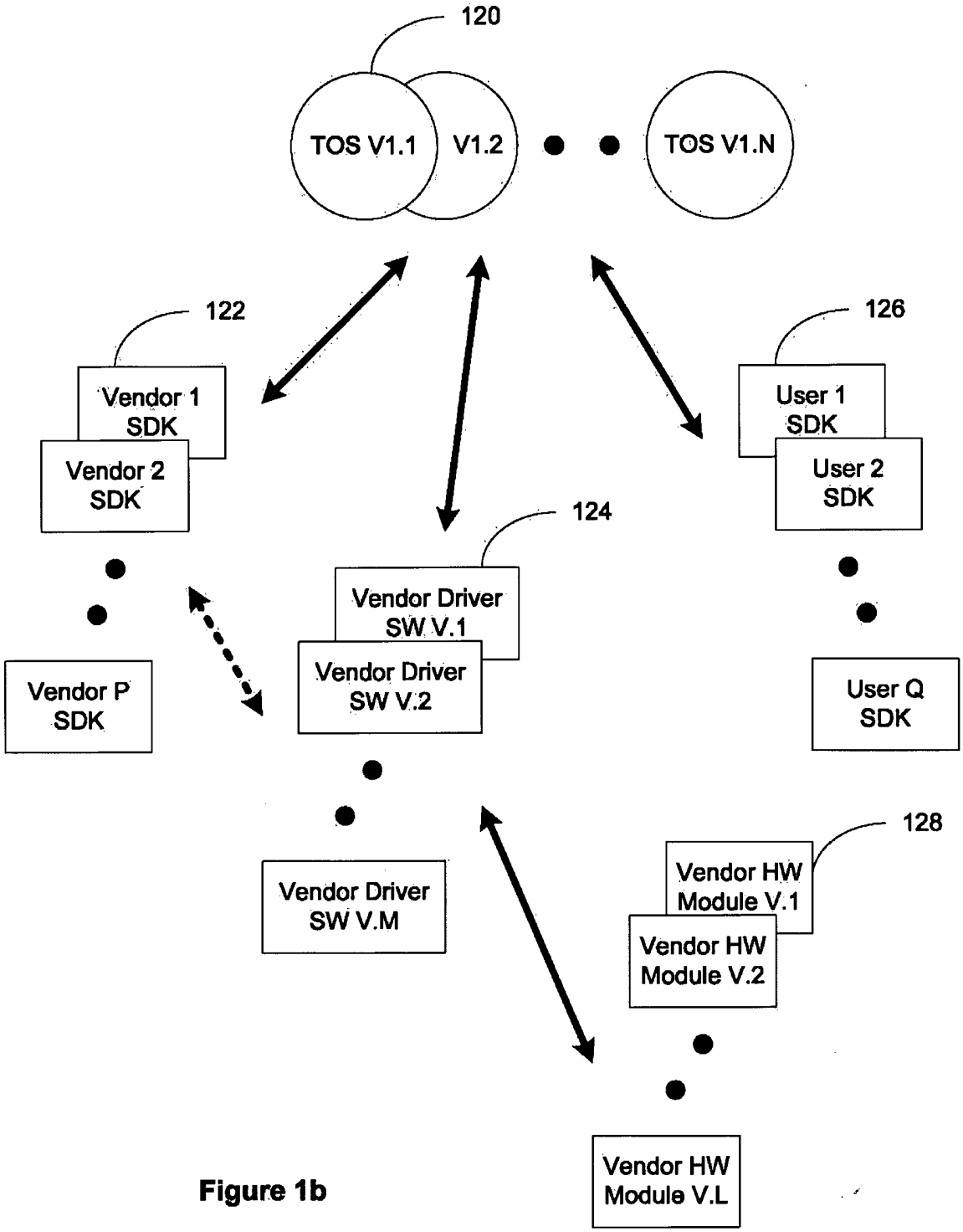


Figure 1b

SUBSTITUTE SHEET (RULE 26)

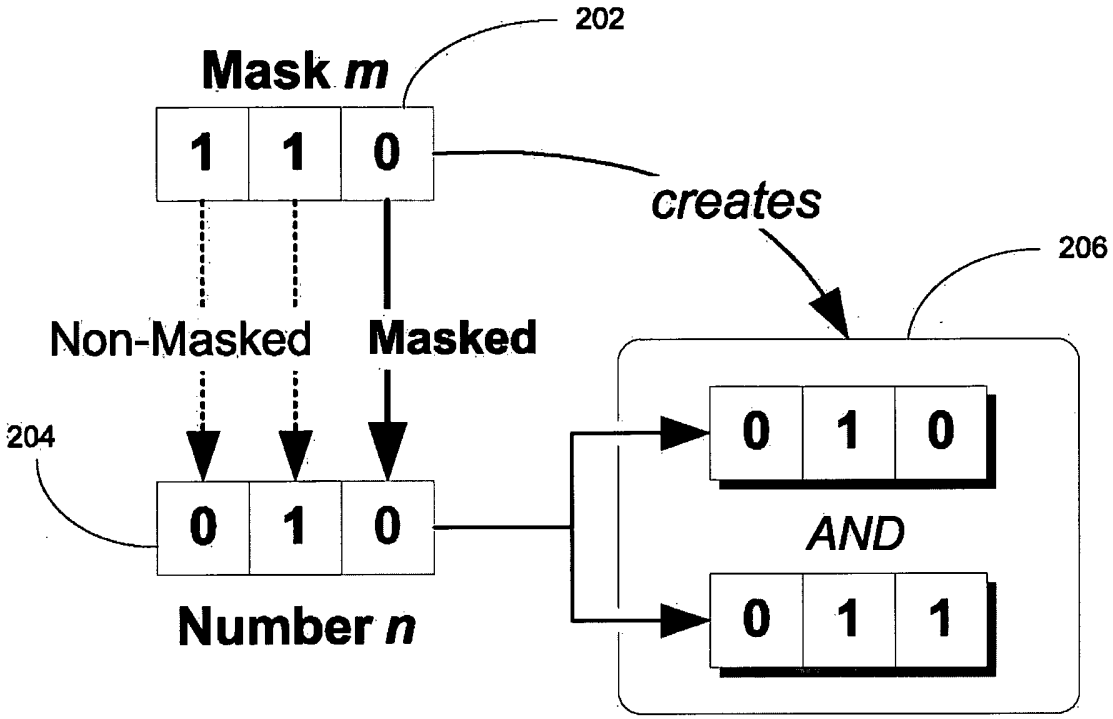

202
Mask m
1 1 0
creates
206
Non-Masked
Masked
204
0 1 0
Number n
0 1 0
AND
0 1 1

FIG. 2

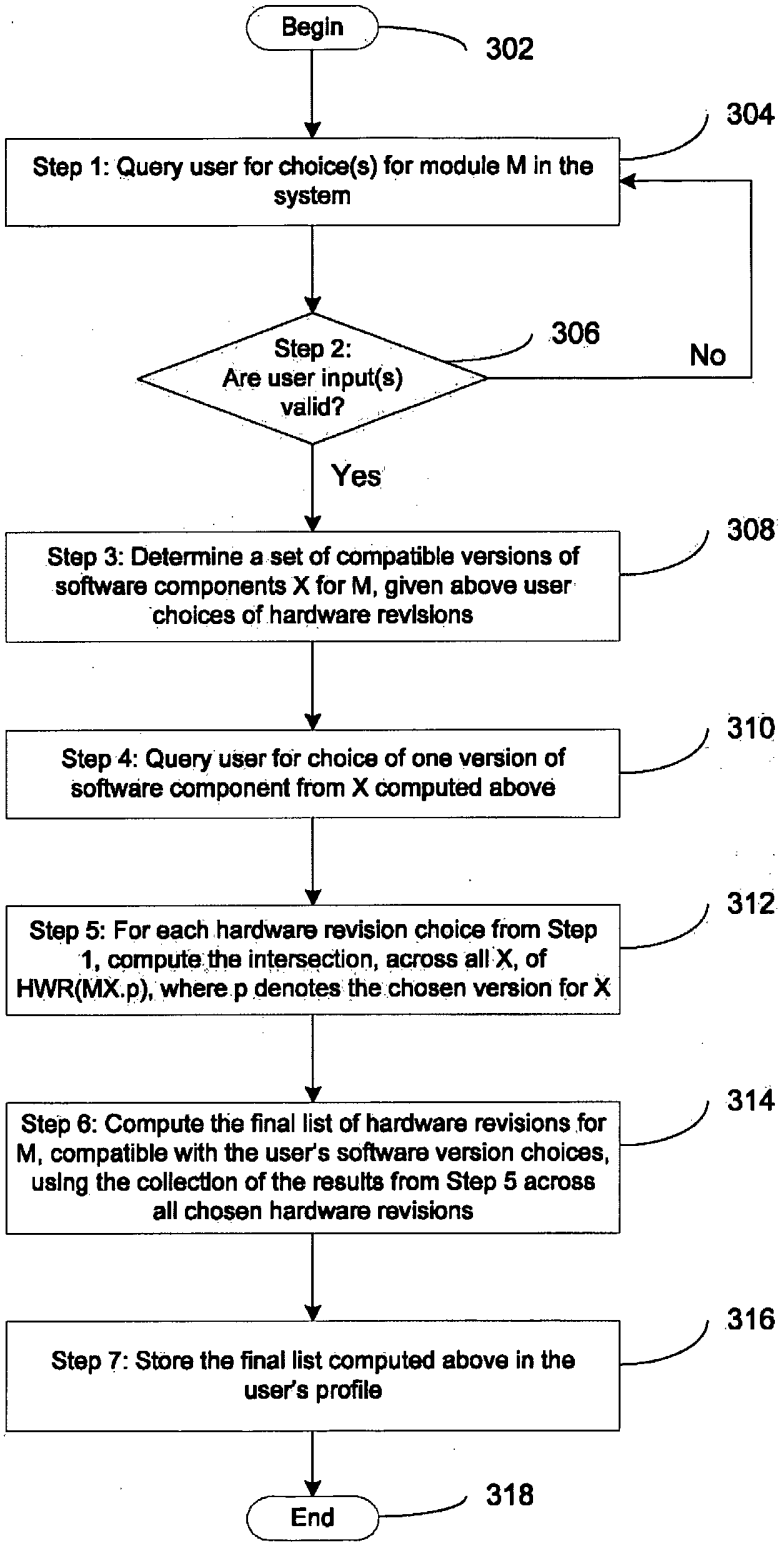


FIG. 3

SUBSTITUTE SHEET (RULE 26)

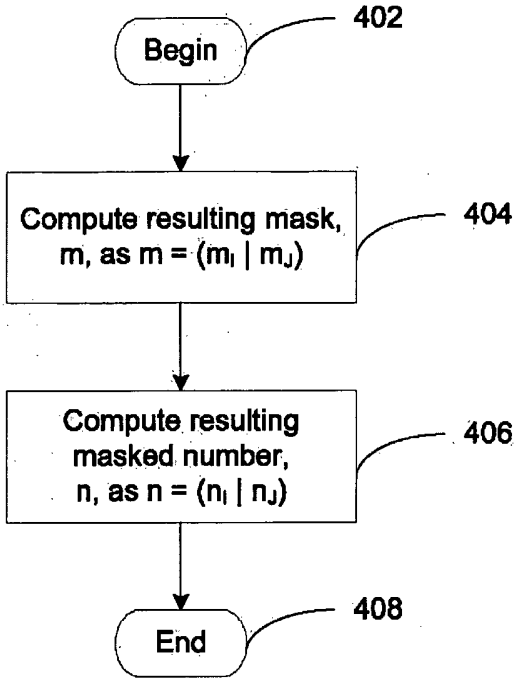

Begin
402
Compute resulting mask, m, as m = (m_I | m_J)
404
Compute resulting masked number, n, as n = (n_I | n_J)
406
End
408

FIG. 4

INTERNATIONAL SEARCH REPORT

[redacted]national application No
PCT/JP2006/304517

A. CLASSIFICATION OF SUBJECT MATTER
INV. G01R31/319

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G01R G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, INSPEC, WPI Data, IBM-TDB

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2004/225459 A1 (KRISHNASWAMY RAMACHANDRAN ET AL) 11 November 2004 (2004-11-11) abstract; figures 1,2,310 paragraphs [0007] - [0017] paragraph [0955] paragraph [0990] - paragraph [0998] paragraph [0089] paragraph [0098] paragraph [0135] paragraph [0156]	1,2, 10-12, 20,21, 23,25
A	EP 1 237 073 A (MICROSOFT CORPORATION) 4 September 2002 (2002-09-04) abstract; figures 2,3 paragraph [0040] - paragraph [0045]	1-25

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
28 June 2006	06/07/2006

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Böhm-Pélissier, A

INTERNATIONAL SEARCH REPORT

[redacted]national application No
PCT/JP2006/304517

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2004225459 A1	11-11-2004	NONE	
EP 1237073 A	04-09-2002	JP 2002229834 A	16-08-2002
		US 2002123992 A1	05-09-2002