EUROPEAN PATENT SPECIFICATION

(54) System for Creating a Customized Software Installation On Demand

System zur Erstellung einer individuellen Softwareinstallation auf Anfrage

Système de création d’un logiciel personnalisé sur demande

(51) Int Cl.: G06F 9/445 (2006.01)

(21) Application number: 06113569.5

(22) Date of filing: 05.05.2006

(84) Designated Contracting States: DE FR GB IE

(30) Priority: 19.05.2005 US 134541

(43) Date of publication of application: 22.11.2006 Bulletin 2006/47

(73) Proprietor: Novell, Inc.

Provo, Utah 84606-6194 (US)

(72) Inventors:

• Isaacson, Scott A.
  Woodland Hills, UT UT 84653 (US)

• Anderson, Eric W. B.
  Alpine, UT UT 84004 (US)

• Wipfel, Robert
  Draper, UT UT 84020 (US)

(74) Representative: Hanna, Peter William Derek et al

Hanna Moore & Curley

13 Lower Lad Lane

Dublin 2 (IE)

(56) References cited:


US-B1- 6 324 691

US-B1- 6 539 473

US-B1- 6 725 452

US-B1- 6 735 757

US-B1- 6 735 757

Note: Within nine months of the publication of the mention of the grant of the European patent in the European Patent Bulletin, any person may give notice to the European Patent Office of opposition to that patent, in accordance with the Implementing Regulations. Notice of opposition shall not be deemed to have been filed until the opposition fee has been paid. (Art. 99(1) European Patent Convention).
DESCRIPTION

FIELD OF THE INVENTION

[0001] This invention pertains to allowing a user to create a customized installation of software packages, and more particularly to enabling users to select and verify the interoperability, at run-time, of a software installation.

BACKGROUND OF THE INVENTION

[0002] Software distributions are typically built into an installation program and stored on a compact disc (CD), to be purchased by the customer. Such distributions usually include at least one program file and a number of other packages that work with the program file to provide additional functionality and features. These CDs are pre-packaged and designed to include features that the company selling the software thinks will be desired by customers.

[0003] Manufacturers of such software products recognize that a "one size fits all" mentality often does not work for all clients. Thus, manufacturers sometimes produce multiple different versions of a software package. For example, application suites are very popular products today. Most versions include a word processor and a spreadsheet program. But some versions might include a database package, whereas other versions might include a slideshow generating program. Still other versions might include stripped-down versions of the products, priced to sell to students and educators. By offering different versions of the product, the manufacturer hopes that as many customers as possible will be satisfied by the different versions, thereby maximizing the manufacturer's sales.

[0004] This approach to building pre-packaged software installations is used not just with application software, but also with operating systems. For example, in selecting a Linux® distribution, a customer must choose between different packages of distributions that have been released and are available off-the-shelf combinations. (Linux is a registered trademark of Linus Torvalds.) A customer typically chooses a Linux distribution by first selecting a vendor who sells Linux distributions, and then identifying a particular distribution available from the vendor that has the most features that the customer is looking for. But if a customer wants a finer level of control in selecting the structure of the Linux distribution, the customer is usually left wanting.

[0005] Accordingly, a need remains to allow a user to create a customized software installation, including only packages that the user wants, verifying that the installation will be operable at run-time, and including any required package dependencies.

SUMMARY OF THE INVENTION

[0006] The present invention provides a system, method and program for creating a user-customized software installation, in accordance with claims which follow. A customized installation is created by allowing a user to select packages that the user is interested in installing. During the process, the user is informed if two of the packages that were selected will not be interoperable with each other at run-time. The user is then given an opportunity to resolve the conflict. In cases where packages conflict or will not work together at run-time, a different package can be selected to serve the purpose. After verification that the selected packages do not conflict, the user then can then install the customized installation by installing the selected packages.

[0007] The foregoing and other features, objects, and advantages of the invention will become more readily apparent from the following detailed description, which proceeds with reference to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] FIG. 1 shows a system on a computer configured to support a distribution of Linux on demand, according to an embodiment of the invention.

FIG. 2 shows an example of the set of packages of FIG. 1 that are available for user selection.

FIG. 3 shows an example of two packages from the database of run-time conflict information of FIG. 1 that will conflict at run-time.

FIG. 4 shows an example table in the database of package run-time conflicts of FIG. 1.

FIG. 5 shows a flowchart of the procedure used by the conflict checker of FIG. 1 to resolve conflicts in packages.

FIG. 6 shows an example of dependency information that is stored in the database of FIG. 1.

FIGS. 7A-7B show a flowchart of the procedure used by the dependency validator of FIG. 1 to validate that the necessary package dependencies are included in the customized installation.

FIG. 8 shows a flowchart of the procedure used by the assembler of FIG. 1, to assemble an installation with both dependency package validation and conflict resolution for all packages in the installation.

FIG. 9 shows a table identifying what packages a particular user received in the installation of FIG. 1.

FIG. 10 shows a system where the computer of FIG. 1 is a bootstrap server capable of installing the installation on to remote servers.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

[0009] FIG. 1 shows a system on a computer with a set of available packages, a set of user requirements, a database with dependency and run-time conflict information, a conflict checker, a dependency validator, an as-
connection. Wireless (such as IEEE 802.11a/b/g/n, among others) connection.

In the art will recognize that the set of Linux packages 110 includes database 120 that stores conflict and dependency information, making it possible to verify that no packages conflict at run-time and that all required dependencies will be included.

Computer 100 also includes a conflict checker 130 to ensure that none of the selected packages will conflict at run-time, and a dependency validator 140 to validate that all dependencies of selected packages are also selected. Once the Linux packages have been selected and checked for conflicts and dependencies, an assembler 150 can then assemble the selected packages into an installation 160. In one embodiment of the invention, the assembler assembles the installation 160 as an ISO image file 170. All of these elements are discussed in greater detail below.

FIG. 2 shows an example of a set of packages 110 in FIG. 1 that are available for selection, according to an embodiment of the invention. In the example set of packages 110, there are six package categories, kernel 210, security 220, file system 230, encryption 240, firewall 250 and web server 260, as well as two to three packages for each category. While the present embodiment has only six categories and fourteen packages, a person skilled in the art will recognize that the categories are not required. In addition, the set of packages can be organized into any number of categories, and any number of packages per category (and, of course, any number of total packages).

In an embodiment of the invention, a deployment tool can be used to investigate the computer environment of the user. The selection of the packages can be done automatically for the user based on the packages that are in the user’s current Linux system. After the deployment tool has identified what packages are currently on the user’s system, an installation is built using the most recent versions of those packages (along with any other packages necessary or deemed of interest).

In another embodiment, a user of the system will be familiar with Linux and its respective packages, and will have preferences on which packages he is interested in installing. For example, a user might be particularly interested in getting the security package XYZ, as well as the encryption package XYZ. With the checkbox embodiment of FIG. 2, the user would select a kernel of his choice 211 or 212, and then also select security XYZ 223, and encryption XYZ 242. And finally in another embodiment, a user might not be interested in security, but might need a file system 230, a web server 260, and a firewall 250. Again, the user likely has particular packages in mind, and can select the appropriate packages.
But if the user has no particular preferences for certain features, the system can provide default selections, or make recommendations on those features (based, for example, on levels of compatibility among the various user-selected packages). While checkboxes are the user interface element used in the example of FIG. 2, a person skilled in the art will recognize that there are other means of identifying selected packages for an installation, e.g., list box, search boxes, etc.

[0019] In an embodiment of the invention, the packages that are built into an installation are compiled binary files that the user will be able to run immediately after installing the installation. In the prior art, customers built installations by compiling source code components from different vendors. Because compiler settings can affect the behavior of software, even technically skilled users could inadvertently build an installation that would not work as intended. By providing the user with compiled modules, embodiments of the invention avoid these problems, and save the user time (in that the user does not have to spend time building the installation from the source code).

[0020] FIG. 3 shows an example of the computer of FIG. 1 identifying two packages that will conflict at run-time using the database of run-time conflict information in FIG. 1, according to an embodiment of the invention. In FIG. 3, the installation includes packages "security ABC" 221 and "file system XYZ" 233. The conflict checker 130 obtains information about the security ABC package 221 from conflict database 120 with locator 310, and information about file system XYZ package 233 from conflict database 120 with locator 311 (which can, of course, be the same locator). In the example shown in FIG. 3, conflict checker 130 determines that the packages conflict, represented by conflict symbol 320, and presents message 330 to the user, so that the conflict can be resolved.

[0021] In another embodiment, the selected packages might be packages that do not conflict at run-time. In this case, the conflict checker 320 does not prompt the user to resolve the package conflict, and instead compares the other packages in the set of selected packages in search of conflicts. FIG. 4 shows an example database table in the database in FIG. 1 that records packages that conflict at run-time, according to an embodiment of the invention. Table 120 is a table with rows 410-414 and columns 420-424 representing the various packages, such as Kernel XYZ 410. In the example shown in FIG. 4, there is only conflict information for five packages, but a person skilled in the art will recognize that in other examples there can be any number of packages.

[0022] Conflict information is represented by an X, such as Xs 430, 431, 432, 433, 434, 435, in entries in table 200. For each pair of packages that has a conflict, table 200 stores an indication of this conflict in the appropriate table entry. For example, X 430 represents a conflict between package "security ABC" in column 221 and package "kernel XYZ" in row 210. The conflict 430 means that an installation containing both kernel XYZ and security ABC will not be interoperable at run-time. In other words, while the kernel XYZ package can operate on its own, and can interoperate with other packages, kernel XYZ does not interoperate with security ABC at run-time. (Presumably, there is some other version of the kernel that interoperates with security ABC, or else security ABC cannot be used at all.)

[0023] Although FIG. 4 shows conflict information being arranged in an N x N table, where N is the total number of packages, a person skilled in the art will recognize that there are other ways of recording conflict information. For example, database 120 includes redundant information, in that every combination of packages is represented twice, e.g., Xs 430 and 431 both represent a conflict between kernel XYZ and security ABC. Other embodiments of the conflict information can include linked lists, arrays, etc. In addition, a person skilled in the art will recognize that other conflict combinations are possible, and will recognize how to modify database 120 to store this additional information. For example, there may be three different packages, which include no pair-wise conflicts, but as a trio conflict.

[0024] FIG. 5 shows a flowchart that the conflict checker 130 in FIG. 1 uses to resolve run-time conflicts in packages, according to an embodiment of the invention. In step 510, the conflict checker starts by identifying two of the packages that have been selected for the installation. In step 520, the conflict checker refers to the database to see if the packages have a run-time conflict. This can be accomplished, among other ways, by having the database store, for each package, a list of other packages with which the first package conflicts. A person skilled in the art will recognize that this information can be stored in a number of different manners: for example, by using a list for each package, or by creating a table showing pairs of packages and flagging which combinations have conflicts. (A person skilled in the art will also recognize that conflicts can extend beyond pairs of packages: for example, there can be three packages which do not pair-wise conflict, but as a trio conflict.) If there is a conflict with the packages, the user is alerted with a message at step 540. Otherwise, at step 530 the conflict checker looks to see if there is another combination of packages to check. If there are no more combinations, the conflict checker finishes, having successfully validated that no selected packages contain any run-time conflicts. If there are more combinations of selected packages, the conflict checker 130 then goes back to step 510 and begins the process all over again. FIG. 5 is described in an abstract model (for example, FIG. 5 does not specify exactly how the conflict checker selects packages in step 510). But a person skilled in the art will recognize how to adapt FIG. 5: for example, by using nested loops to select pairs of packages. A person skilled in the art will also recognize how to adapt FIG. 5 to check for conflicts among groups of packages larger than two.

[0025] In an embodiment of the invention, the conflict
In another embodiment of the invention, the conflict checker provides the user with a recommendation for resolving the current run-time conflict. Sometimes a conflict between packages might have a relatively straightforward resolution. For example, there might be one package that conflicts with several others. If those other packages do not conflict with any more packages, then the system could recommend an alternative package to the one that is causing the numerous conflicts.

In yet another embodiment, a means of resolving a package conflict might not be as straightforward. For example, it could be the case that two packages conflict with each other, but not with any other packages in the set of selected packages. In this case, it is not necessarily clear which of the two conflicting packages should be replaced with an alternative non-conflicting package. In this context, the conflict checker can alert the user to which packages are in conflict.

FIG. 6 shows an example of dependency information that is stored in database 120 in FIG. 1, according to an embodiment of the invention. In the present embodiment, two dependencies are shown. In dependency 600, Encryption XYZ 223 has a package dependency of Security XYZ 233. So if Encryption XYZ 223 is in the installation, then Security XYZ 223 should also be included in the installation for the encryption software to run.

Similarly, dependency 600 shows that Security XYZ 233 requires that Kernel XYZ 211 be selected and included in the installation. As a result, a selection of Encryption XYZ 223 will require that not only Security XYZ 233 be selected and included in the installation, but also that Kernel XYZ 211 be selected and included in the installation.

As can be seen, the example of FIG. 6 shows only immediate dependencies, under the assumption that any indirect dependencies are captured by checking the dependency information for the needed package. Thus, dependency 650 does not reflect that Encryption XYZ 223 depends (indirectly) on Kernel XYZ 211. This information is represented through dependency 600. But a person skilled in the art will recognize that database 120 can store all the dependencies for a single package, whether direct or indirect. Thus, dependency 650 can be modified to reflect that Encryption XYZ 223 is also dependent on Kernel XYZ 211.

FIGs. 7A-7B show a flowchart of the procedure used by the dependency validator 140 of FIG. 1 to validate that the necessary package dependencies are included in the customized installation, according to an embodiment of the invention. In FIG. 7A, at step 710, the dependency validator begins by identifying a package. In step 720 the dependency validator looks up that package in the dependency database 120, and checks to see if that package depends on any other packages. If a dependency does not exist, then the dependency validator goes to step 730 and checks to see if there are more packages that need dependency checking. If at step 730 there are more packages to check for dependencies, then the dependency validator returns to step 710 and identifies the next package to move through the flowchart again. However, if at step 730 there are no more packages that need to be checked for dependencies, the dependency validation is complete, and the selected packages can be built into an installation with the assurance that all required dependency packages are included.

While one embodiment of the invention alerts the user to a dependency issue as soon as a problem is identified, another embodiment can check all the packages in the selected set and identify all missing but needed packages before alerting the user of the missing packages. In yet another embodiment, the dependency checker can check for dependency packages as soon as a package is selected. While packages are being selected, it can select the dependency package and note the automatic selection of the additional package (so that the user is aware of this automatic selection). If a needed package is removed from the set of selected packages, then the original package can be removed as well (again, with the system notifying the user of this automatic action). In one embodiment, the alerts of dependency packages can be in the form of a dialog box, but a person skilled in the art will recognize that there are other ways of alerting the user of missing dependencies, such as text in the selection interface itself, log files or windows, etc.

EP 1 724 682 B1
dependency validator sees the particular security package, the dependency validator then checks for a kernel that is in the set of dependency packages that will satisfy the dependency requirement for the security package. Similarly, a selected encryption package can only require that a security package be included for the encryption package to be operable. In this case, it is not important which security package is included, only that one security package is included. A person skilled in the art will also recognize other combinations of dependencies that can be tested for.

[0035] FIG. 8 shows a flowchart of the procedure used by the assembler 150 in FIG. 1 with both dependency package validation 140 and conflict resolution 130 for all packages in the installation, according to an embodiment of the invention. In step 810, the assembler starts with the set of selected packages. In step 820 the assembler checks to see if any packages conflict, as described above with reference to FIG. 5. If no packages conflict, then the assembler goes to step 830 and validates that the necessary dependency packages are included in the set of selected packages, as described above with reference to FIGS. 7A-7B.

[0036] If at step 820 the conflict checker 140 identifies packages that will conflict at run-time, or if at step 830 the dependency validator 130 identifies dependencies that must be selected and included in the installation, the assembler goes to step 840 where it prompts the user to resolve the issues in the selected packages. If at step 830 the dependency validator finds no missing dependencies (and no conflicts), the assembler goes to step 850 where the packages are assembled into an installation, which can then be installed on a computer.

[0037] After the conflict checker and dependency validator successfully certify the packages in the installation, the installation is ready to be installed. In one embodiment of the invention, the installation can be installed over a network to a remote server. FIG. 10 shows a system where the computer in FIG. 1 is a bootstrap server capable of installing the installation on to remote servers, according to an embodiment of the invention. Bootstrap server 1000 includes the elements of the computer in FIG. 1, with a set of packages available for user selection, a conflict checker, a dependency validator, and an assembler with an assembled installation, and an ISO image of the installation. After the installation has been assembled, bootstrap server 1000 remotely boots destination server 1010, and installs base kernels to destination server 1010. Then destination server 1010 can install the other selected packages in the customized installation.

[0038] In one embodiment of the invention, bootstrap server 1000 could save a backup of the installation, so that the installation could be replicated if necessary. In an embodiment of the invention the backup of the installation can be represented as an ISO image of the installation.

[0039] Also, in another embodiment of the invention, bootstrap server 1000 could serve as a cache of all packages in order to have dependency packages available in the future. If a user installs a package in the future that has a needed dependency package that is not included in the user customized installation, then the cache provides access to the dependency package. While this embodiment uses one destination server, it would be obvious to a person skilled in the art that any number of destination servers could be used.

[0040] In another embodiment of the invention is a way to retain information about what packages are included in a particular customer's customized Linux installation. FIG. 9 shows a database table identifying what packages a particular user received in the installation in FIG. 1, according to an embodiment of the invention. After an installation is built for a user, information is stored to identify what packages were included for the user.

[0041] In Table 900, Users in column 910 are matched with Packages in column 950. User 1 in entry 915 created an installation that included Kernel ABC 951, Security ABC 952, and Encryption ABC 953. In the event that, for example, Encryption ABC 952 is updated, this new version of the package can be added to the set of packages available for creating a customized Linux installation. However, this updated package can also be distributed to users who have the previous version of Encryption ABC 952. A query for Encryption ABC 952 in Package table 900 identifies User 1 in entry 915 as having installed Encryption ABC 952. This way, User 1 can be notified of the update, for example, by e-mail, and can install the update if desired. Similarly if Encryption JKL 956 is updated, User 2 in entry 920 can be notified of the update. Although FIG. 9 shows the package information being stored as a table, a person skilled in the art will recognize that there are other ways of recording package information, such as linked lists, arrays, etc.

[0042] While currently it is possible to notify Linux users when updates to packages are available, an embodiment of the invention makes it possible for users of a customized Linux installation to receive notifications only when a package that is in the user's own customized version is updated. In this way, the user does not get bombarded with notices of updates to packages that the user does not have.

[0043] Another embodiment of the invention includes a deployment tool that checks for updates to packages in a user's Linux environment. In one embodiment the deployment tool can be set up to automatically use the Internet to see if any patches are available to packages in the user's Linux environment. The deployment tool can be set up to check on a periodic basis, for example every month. A person skilled in the art will recognize that this periodic basis could be any set period, or that this period can be set as a preference of the user.

[0044] In addition to automatically checking for updates to packages, the deployment tool can also allow users to run query for updates on demand. Instead of having the update checking as a process that is started automatically, users can manually start the process. A
person skilled in the art will recognize that some users might prefer to always have the updates to their packages, while other users might prefer to always be notified of the package updates, in order to approve of the updates before the updates are installed.

In addition to being able to provide customized notifications of package updates, the information in table 900 in FIG. 9 can be used by a Linux vendor as a basis for a customer support agreement. That way, a customer is able to get support for packages included in the customer’s installation. Similarly, the vendor knows what packages the customer is entitled to support.

The following discussion is intended to provide a brief, general description of a suitable machine in which certain aspects of the invention may be implemented. Typically, the machine includes a system bus to which is attached processors, memory, e.g., random access memory (RAM), read-only memory (ROM), or other state preserving medium, storage devices, a video interface, and input/output interface ports. The machine may be controlled, at least in part, by input from conventional input devices, such as keyboards, mice, etc., as well as by directives received from another machine, interaction with a virtual reality (VR) environment, biometric feedback, or other input signal. As used herein, the term “machine” is intended to broadly encompass a single machine, or a system of communicatively coupled machines or devices operating together. Exemplary machines include computing devices such as personal computers, workstations, servers, portable computers, handheld devices, telephones, tablets, etc., as well as transportation devices, such as private or public transportation, e.g., automobiles, trains, cabs, etc.

The machine may include embedded controllers, such as programmable or non-programmable logic devices or arrays, Application Specific Integrated Circuits, embedded computers, smart cards, and the like. The machine may utilize one or more connections to one or more remote machines, such as through a network interface, modem, or other communicative coupling. Machines may be interconnected by way of a physical and/or logical network, such as an intranet, the Internet, local area networks, wide area networks, etc. One skilled in the art will appreciated that network communication may utilize various wired and/or wireless short range or long range carriers and protocols, including radio frequency (RF), satellite, microwave, Institute of Electrical and Electronics Engineers (IEEE) 802.11, Bluetooth, optical, infrared, cable, laser, etc.

The invention may be described by reference to or in conjunction with associated data including functions, procedures, data structures, application programs, etc. which when accessed by a machine results in the machine performing tasks or defining abstract data types or low-level hardware contexts. Associated data may be stored in, for example, the volatile and/or non-volatile memory, e.g., RAM, ROM, etc., or in other storage devices and their associated storage media, including hard-drives, floppy-disks, optical storage, tapes, flash memory, memory sticks, digital video disks, biological storage, etc. Associated data may be delivered over transmission environments, including the physical and/or logical network, in the form of packets, serial data, parallel data, propagated signals, etc., and may be used in a compressed or encrypted format. Associated data may be used in a distributed environment, and stored locally and/or remotely for machine access.

Having described and illustrated the principles of the invention with reference to illustrated embodiments, it will be recognized that the illustrated embodiments may be modified in arrangement and detail without departing from such principles. And although the foregoing discussion has focused on particular embodiments and examples, other configurations are contemplated. In particular, even though expressions such as “according to an embodiment of the invention” or the like are used herein, these phrases are meant to generally reference embodiment possibilities, and are not intended to limit the invention to particular embodiment configurations. As used herein, these terms may reference the same or different embodiments that are combinable into other embodiments. Descriptions of the embodiments by reference to Linux are illustrative; the invention can be used with other operating systems and software distributions.

Consequently, in view of the wide variety of permutations to the embodiments described herein, this detailed description and accompanying material is intended to be illustrative only, and should not be taken as limiting the scope of the invention.

Claims

1. A system for installing a user-customized operating system software installation created from one or more software packages, comprising:
   a server (1000) including a set of software packages (110) available for user selection; a selector to enable a user to select a subset of the set of software packages, the selected subset including at least a first package, a second package, a third package, one of which is base kernel (210) package and omitting a non-included fourth package and a non-included fifth package; an installation builder to build an installation (160) including the selected subset of the set of software packages; a database (120) of package conflicts and dependencies, a conflict including an indication that the first package and the second package do not conflict at run-time, a second indication that the first package and the third package do not conflict at run-time and a third indication that the first package and the fourth package do not conflict...
at run-time, a dependency including an indication that the first package depends on the fifth package;
a conflict checker (130) to ensure that none of the selected packages will conflict at run-time, with reference to the database (120);
a dependency validator (140) to validate that all dependencies of selected packages are also satisfied, with reference to the database (120);
a destination (1010) for a user-customized software installation; and
an installer to install (160) the user-customized software installation at said destination;
characterized in that
the server (1000) is a bootstrap server,
at least a first destination (1010) is capable of a remote boot,
wherein the bootstrap server includes the conflict checker (130), the dependency validator (140) and an assembler (150) that assembles the installation with both dependency package validation and conflict resolution for all packages in the installation by removing the third package from the selected subset of the set of available software packages and including the fourth package, in the selected subset of the set of available software packages, and
the bootstrap server (1000) remotely boots the first destination, and installs the selected base kernel packages (211,212) so that the first destination (1010) can then install the other selected packages (221-223;231-233;241-242;251-252;261-262) in the user-customized installation.

2. A system according to claim 1, further comprising a recommender to recommend the fourth package to add to the installation.

3. A system according to claim 1, further comprising:
an update to a package in the installation; and
a notifier to notify the user of the update.

4. A system according to claim 3, further comprising an installer to install the update.

5. A system according to claim 1, comprising a backup copy of the installation (160).

6. A method for installing on demand for a user a user-customized operating system software installation created from one or more software packages, comprising:
accessing a server having a set of software packages (110) available for user selection;
selecting (510) a subset of the set of software packages, the subset including least a first package, a second package, a third package, one of which is base kernel (210) package and omitting a non-included fourth package and a non-included fifth package;
building the selected subset of software packages into an installation (160);
providing a database (120) of package conflicts and dependencies, a conflict including an indication that the first package and the second package do not conflict at run-time, a second indication that the first package and the third package conflict at run-time, and a third indication that the first package and the fourth package do not conflict at run-time, a dependency including an indication that the first package depends on the fifth package;
identifying that there is a conflict (520,820) between the first package and the third package; removing the third package from the subset of the available software packages; selecting the fourth package that does not conflict with the first package at run-time; adding the fourth package to the subset of the available software packages; identifying that there is a dependency (720,830) between the first package and the fifth package; adding the fifth package to the subset of the available software packages; installing (160) the user-customized software installation at a destination;
characterized in that
the server (1000) is a bootstrap server,
at least a first destination (1010) is capable of a remote boot,
wherein the bootstrap server includes a conflict checker (130), a dependency validator (140) and an assembler (150) that assembles the installation with both dependency package validation and conflict resolution for all packages in the installation by removing the third package from the selected subset of the set of available software packages and including the fourth package, in the selected subset of the set of available software packages, the bootstrap server (1000) remotely boots the first destination, and installs the selected base kernel packages (211,212) so that the first destination (1010) can then install the other selected packages (221-223;231-233;241-242;251-252;261-262) in the user-customized installation.

7. A method according to claim 6, further comprising recommending (540) the third package to be removed from the installation (160).

8. A method according to claim 7, further comprising
9. A method according to claim 6, further comprising notifying the user when an update to a package in the installation (160) is available.

10. A method according to claim 9, further comprising installing the update.

11. A method according to claim 6, further comprising storing a backup copy of the installation.

12. A computer program which when executing on a machine performs the steps of any one of claims 6 to 11.

Patentansprüche

1. System zum Installieren einer Benutzer-angepassten Betriebssystem-Softwareinstallation, die aus einem oder mehreren Softwarepaketen erzeugt wird, umfassend:
   - einen Server (1000), der einen Satz von Softwarepaketen (110) beinhaltet, die für eine Benutzerauswahl verfügbar sind;
   - einen Auswähler, um es einem Benutzer zu ermöglichen, einen Teilsatz des Satzes von Softwarepaketen auszuwählen, wobei der ausgewählte Teilsatz mindestens beinhaltet ein erstes Paket, ein zweites Paket, ein drittes Paket, wovon eines ein Basiskernelpaket (210) ist und auslassend ein nicht beinhaltetes viertes Paket und ein nicht beinhaltetes fünftes Paket;
   - einen Installationsbuilder, um eine Installation (160) zu erstellen, die den ausgewählten Teilsatz des Satzes von Softwarepaketen beinhaltet;
   - eine Datenbank (120) von Paketkonflikten und Abhängigkeiten, wobei ein Konflikt eine Markierung beinhaltet, dass das erste Paket und das zweite Paket nicht zur Laufzeit kollidieren, eine zweite Markierung, dass das erste Paket und das dritte Paket zur Laufzeit kollidieren, und eine dritte Markierung, dass das erste Paket und das vierte Paket nicht zur Laufzeit kollidieren;
   - einen Konfliktprüfer (130), um zu gewährleisten, dass keines der ausgewählten Pakete zur Laufzeit kollidieren wird, mit Bezug auf die Datenbank (120);
   - einen Abhängigkeitsvalidierer (41), um zu validieren, dass alle Abhängigkeiten von ausgewählten Paketen befriedigt sind, mit Bezug auf die Datenbank (120);
   - ein Ziel (1010) für eine Benutzer-angepasste Softwareinstallation; und
   - einen Installierer, um die Benutzer-angepasste Softwareinstallation an einem Ziel zu installieren (160);

gekennzeichnet dadurch, dass
   - der Bootstrap-Server (1000) das erste Ziel fernbootet, und die ausgewählten Basiskernelpakete (211, 212) installiert, so dass das erste Ziel (1010) dann die anderen ausgewählte Pakete (221-223; 231-233; 241-242; 251-252; 261-262) in der Benutzer-angepassten Installation installieren kann.

2. System nach Anspruch 1, ferner umfassend einen Empfehler, um das vierte Paket zu empfehlen, zu der Installation hinzuzufügen.

3. System nach Anspruch 1, ferner umfassend:
   - eine Aktualisierung zu einem Paket in der Installation; und
   - einen Benachrichtiger, um den Benutzer über die Aktualisierung zu benachrichtigen.

4. System nach Anspruch 3, ferner umfassend einen Installierer, um die Aktualisierung zu installieren.

5. System nach Anspruch 1, umfassend eine Backup-Kopie der Installation (160).

6. Verfahren zum Installieren auf Anforderung für einen Benutzer, eine Benutzer-angepasste Betriebssystem-Softwareinstallation, die aus einem oder mehreren Softwarepaketen erzeugt ist, umfassend:
   - Zugreifen auf einen Server mit einem Satz von Softwarepaketen (110), die für Benutzerauswahl verfügbar sind;
   - Auswählen (510) eines Teilsatzes des Satzes von Softwarepaketen, wobei der Teilsatz mindestens beinhaltet ein erstes Paket, ein zweites Paket, ein drittes Paket, wovon eines ein Basis-
kernel-(210)-Paket ist und Auslassend ein nicht beinhaltetes viertes Paket und ein nicht beinhal- tetes fünftes Paket;

Einbauen des ausgewählten Teilsatzes von Softwarepaketen in eine Installation (160);

Bereitstellen einer Datenbank (120) von Paket-konflikten und Abhängigkeiten, wobei ein Kon- flikt beinhaltet eine Markierung, dass das erste Paket und das zweite Paket nicht zur Laufzeit kollidieren, eine zweite Markierung, dass das er- ste Paket und das dritte Paket zur Laufzeit kol- lidieren, und eine dritte Markierung, dass das erste Paket und das vierte Paket nicht zur Lauf- zeit kollidieren, wobei eine Abhängigkeit eine Markierung beinhaltet, dass das erste Paket von dem fünften Paket abhängt;

Identifizieren, dass ein Konflikt (520, 820) be- steht zwischen dem ersten Paket und dem drit- ten Paket;

Entfernen des dritten Pakets von dem Teilsatz von verfügbaren Softwarepaketen;

Auswählen des vierten Pakets, das nicht mit dem ersten Paket zur Laufzeit kollidiert;

Hinzufügen des vierten Pakets zu dem Teilsatz von verfügbaren Softwarepaketen;

Identifizieren, dass eine Abhängigkeit (720, 830) besteht zwischen dem ersten Paket und dem drit- ten Paket;

Hinzufügen des fünften Pakets zu dem Teilsatz von verfügbaren Softwarepaketen;

Installieren (160) der Benutzer-anangepassten Softwareinstallation an einem Ziel;


7. Verfahren nach Anspruch 6, ferner umfassend Emp- fehlen (540) des dritten Pakets, von der Installation (160) entfernt zu werden.

8. Verfahren nach Anspruch 7, ferner umfassend Emp- fehlen des vierten Pakets, zu der Installation (160) hinzugefügt zu werden.


10. Verfahren nach Anspruch 9, ferner umfassend In- stallieren der Aktualisierung.


12. Computerprogramm, welches, bei Ausführung auf einer Maschine, die Schritte nach einem der Ansprü- che 6 bis 11 durchführt.

Revendications

1. Système pour installer une installation logicielle de système d’exploitation personnalisée par un utilisa- teur, créée à partir d’un ou plusieurs paquets logi- ciels, comprenant :

un serveur (1000) incluant un ensemble de pa- quets logiciels (110) disponibles pour une sé- lection utilisateur ;

un sélecteur pour permettre à un utilisateur de sélectionner un sous-ensemble dans l’ensem- ble de paquets logiciels, le sous-ensemble sé- lectionné incluant au moins un premier paquet, un deuxième paquet, un troisième paquet, dont l’un est un paquet de noyau de base (210) et omettant un quatrième paquet non inclus et un cinquième paquet non inclus ;

un constructeur d’installation pour construire une installation (160) incluant le sous-ensemble sélectionné parmi l’ensemble de paquets logiciels ;

une base de données (120) de conflits et de dé- pendencies de paquets, un conflit incluant une indication que le premier paquet et le deuxième paquet ne sont pas en conflit au moment de l’exécution, une deuxième indication que le pre- mier paquet et le troisième paquet sont en conflit au moment de l’exécution, et une troisième in- dication que le premier paquet et le quatrième paquet ne sont pas en conflit au moment de l’exécution, une dépendance incluant une indi- cation que le premier paquet dépend du cinquiè- me paquet ;

un vérificateur de conflit (130) pour assurer qu’aucun des paquets sélectionnés ne sera en conflit au moment de l’exécution, en référence à la base de données (120) ;
un validateur de dépendance (140) pour valider que toutes les dépendances de paquets sélectionnés sont également satisfaites, en référence à la base de données (120) ; une destination (1010) pour une installation logicielle personnalisée par l’utilisateur ; et un installateur pour installer (160) l’installation logicielle personnalisée par l’utilisateur sur l’ordinateur ;

**caractérisé en ce que**

le serveur (1000) est un serveur d’amorce, au moins une première destination (110) est capable d’amorce à distance, dans lequel le serveur d’amorce inclut le vérificateur de conflit (130), le validateur de dépendance (140) et un assembleur (150) qui assemble l’installation à la fois avec la validation de dépendance de paquet et la résolution de conflit pour tous les paquets dans l’installation en retirant le troisième paquet du sous-ensemble sélectionné dans l’ensemble de paquets logiciels disponibles et incluant le quatrième paquet, et en ajoutant le cinquième paquet, dans le sous-ensemble sélectionné dans l’ensemble de paquets logiciels disponibles, et le serveur d’amorce (1000) amorce à distance la première destination, et installe les paquets de noyau de base (211, 212) de sorte que la première destination (1010) peut alors installer les autres paquets sélectionnés (221 à 223 ; 231 à 233 ; 241 à 242 ; 251 à 252 ; 261 à 262) dans l’installation personnalisée par l’utilisateur.

2. Système selon la revendication 1, comprenant en outre un recommandeur pour recommander le quatrième paquet à ajouter à l’installation.

3. Système selon la revendication 1, comprenant également :

- une mise à jour d’un paquet logiciel dans l’installation ; et
- un notificateur pour notifier la mise à jour à l’utilisateur.

4. Système selon la revendication 3, comprenant en outre un installateur pour installer la mise à jour.

5. Système selon la revendication 1, comprenant une copie de sauvegarde de l’installation (160).

6. Procédé d’installation à la demande d’une installation logicielle de système d’exploitation personnalisée par un utilisateur, créée à partir d’un ou plusieurs paquets logiciels, comprenant :

- accéder à un serveur ayant un ensemble de paquets logiciels (110) disponibles pour une sélection de l’utilisateur ;
- sélectionner (510) un sous-ensemble dans l’ensemble de paquets logiciels, le sous-ensemble incluant au moins un premier paquet, un deuxième paquet, un troisième paquet, dont l’un est un paquet de noyau de base (210) et omettant un quatrième paquet non inclus et un cinquième paquet non inclus ;
- construire le sous-ensemble sélectionné de paquets logiciels dans une installation (160) ;
- fournir une base de données (120) de conflits et de dépendances de paquets, un conflit incluant une indication que le premier paquet et le deuxième paquet ne sont pas en conflit au moment de l’exécution, une deuxième indication que le premier paquet et le troisième paquet sont en conflit au moment de l’exécution, et une troisième indication que le premier paquet et le quatrième paquet ne sont pas en conflit au moment de l’exécution, une dépendance incluant une indication que le premier paquet dépend du cinquième paquet ;
- identifier qu’il existe un conflit (520, 820) entre le premier paquet et le troisième paquet ;
- retirer le troisième paquet du sous-ensemble de paquets logiciels disponibles ;
- sélectionner le quatrième paquet qui n’est pas en conflit avec le premier paquet au moment de l’exécution ;
- ajouter le quatrième paquet au sous-ensemble des paquets logiciels disponibles ;
- identifier qu’il existe une dépendance (720, 830) entre le premier paquet et le cinquième paquet ;
- ajouter le cinquième paquet au sous-ensemble des paquets logiciels disponibles ;
- installer (160) l’installation logicielle personnalisée par l’utilisateur sur une destination ;

**caractérisé en ce que**

le serveur (1000) est un serveur d’amorce, au moins une première destination (1010) est capable d’amorce à distance, dans lequel le serveur d’amorce inclut le vérificateur de conflit (130), le validateur de dépendance (140) et un assembleur (150) qui assemble l’installation à la fois avec la validation de dépendance de paquet et la résolution de conflit pour tous les paquets dans l’installation en retirant le troisième paquet du sous-ensemble sélectionné dans l’ensemble de paquets logiciels disponibles et incluant le quatrième paquet, et en ajoutant le cinquième paquet, dans le sous-ensemble sélectionné dans l’ensemble de paquets logiciels disponibles, et construise le sous-ensemble sélectionné de paquets logiciels dans une installation (160) ;

fournir une base de données (120) de conflits et de dépendances de paquets, un conflit incluant une indication que le premier paquet et le deuxième paquet ne sont pas en conflit au moment de l’exécution, une deuxième indication que le premier paquet et le troisième paquet sont en conflit au moment de l’exécution, et une troisième indication que le premier paquet et le quatrième paquet ne sont pas en conflit au moment de l’exécution, une dépendance incluant une indication que le premier paquet dépend du cinquième paquet ;

identifier qu’il existe un conflit (520, 820) entre le premier paquet et le deuxième paquet ;

retrait le troisième paquet du sous-ensemble de paquets logiciels disponibles ;

sélectionner le quatrième paquet qui n’est pas en conflit avec le premier paquet au moment de l’exécution ;

ajouter le quatrième paquet au sous-ensemble des paquets logiciels disponibles ;

identifier qu’il existe une dépendance (720, 830) entre le premier paquet et le cinquième paquet ;

ajouter le cinquième paquet au sous-ensemble des paquets logiciels disponibles ;

installer (160) l’installation logicielle personnalisée par l’utilisateur sur une destination ;
sélectionnés (221 à 223 ; 231 à 233 ; 241 à 242 ;
251 à 252 ; 261 à 262) dans l'installation per-
sonnalisée par l’utilisateur.

7. Procédé selon la revendication 6, comprenant en
outre de recommander (540) le troisième paquet à
retirer de l’installation (160).

8. Procédé selon la revendication 7, comprenant éga-
lement de recommander le quatrième paquet à ajou-
ter à l’installation (160).

9. Procédé selon la revendication 6, comprenant en
outre de notifier à l’utilisateur lorsqu’une mise à jour
d’un paquet dans l’installation (160) est disponible.

10. Procédé selon la revendication 9, comprenant éga-
lement d’installer la mise à jour.

11. Procédé selon la revendication 6, comprenant en
outre de stocker une copie de sauvegarde de l’ins-
tallation.

12. Programme informatique qui, lorsqu’il s’exécute sur
une machine, met en oeuvre les étapes selon l’une
quelconque des revendications 6 à 11.
<table>
<thead>
<tr>
<th></th>
<th>Kernel XYZ</th>
<th>Security ABC</th>
<th>File System XYZ</th>
<th>Encryption MNO</th>
<th>Mail Server ABC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel XYZ</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security ABC</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>File System XYZ</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encryption MNO</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Mail Server ABC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

FIG. 4
Start

Identify two packages

Do the two packages conflict?

Any other combinations to check?

Inform the user of the conflict

End

FIG. 5
FIG. 6
Start

Identify a package

Does a dependency exist?

Any more packages to check?

No

Yes

A

B

C

Yes

No

End

FIG. 7A
Is required package(s) included?

No

Alert the user about the dependency

Yes

B

A

C

FIG. 7B
Start

810
Take the selected packages

820
Do any packages conflict?

Yes
840
Have the user resolve the problem

No

830
Are any dependencies missing?

Yes
850
Assemble the packages into an installation

No

End

FIG. 8
<table>
<thead>
<tr>
<th>User</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>Kernel ABC</td>
</tr>
<tr>
<td>User 1</td>
<td>Security ABC</td>
</tr>
<tr>
<td>User 1</td>
<td>Encryption ABC</td>
</tr>
<tr>
<td>User 2</td>
<td>Kernel XYZ</td>
</tr>
<tr>
<td>User 2</td>
<td>File System XYZ</td>
</tr>
<tr>
<td>User 2</td>
<td>Encryption JKL</td>
</tr>
</tbody>
</table>

**FIG. 9**