

Method and System for Upgrading Software

The application is continuation of International Application No. PCT/CN2012/084721, filed on November 16, 2012, which claims the priority of Chinese patent application No. 201110392461.4, filed on December 1, 2011, in Chinese Patent Office, entitled “method and system for upgrading software”, the disclosure of which is incorporated herein by reference in its entirety.

Field of the Disclosure

The present disclosure relates to software processing technology field of data processing devices, and more particularly, to a method and system for updating software.

Background of the Disclosure

There are at present two methods for upgrading software of data processing devices (like computers, intelligent mobile phones and so on), one is an upgrading method based on installation package, the other is an upgrading method based on cab upgrading package.

The upgrading method based on installation package is very simple. It is necessary to download a latest installation package from a server, and run the installation package after the installation package is downloaded. The installation package will achieve the uninstallation of the old version and the installation of the new version.

Disadvantage of the upgrading method based on installation package is as follows. Bandwidth is greatly wasted. A user has to download an installation package as big as several Mega (M) bytes or even hundreds of M bytes even if a small change is made on the function, and most unchanged files will be downloaded and replaced together, thus bandwidth is greatly wasted.

The upgrading method based on cab upgrading package is more targeted. The cab upgrading package is a kind of Microsoft WINDOWS operating system package, which includes upgrading files. For the manner, difference files between software version x and version y are packaged in accordance with standard format (cab format) of the operating system to get a cab upgrading package. The upgrading program starts the cab upgrading package after the cab upgrading package is downloaded to local device of the user, and

the cab upgrading package will finish the upgrade and operation of specified files

according to the version difference, and combine the found operation instruction sets.

In the present disclosure, updating of each software version may accompany with an operation instruction set, and several such operation instructions may be used for upgrading version once, and these operation instructions will be organized in the form of set finally. Each operation instruction in the operation instruction set may include: an operation instruction name which may be identified by the local client, and a relative path and a file name of a target file of the operation corresponding to the instruction, and may further include a name of an auxiliary program for processing the target file. Table 2 is an example of an operation instruction set. In order to facilitate illustration, table 2 further includes use instruction of each operation instruction.

Instruction name	Target file	Use instruction
regsvr32	%INSTALL_DIR%/Ext/AuHelp.dll	Register a specified COM component
unregsvr32	%INSTALL_DIR%/Ext/Help.dll	Deregister a specified COM component
delete	%INSTALL_DIR%/UtilUtil.dll	Delete a specified file
execute	%INSTALL_DIR%/TEST.dll+auxiliary program name	Perform specified processing on a target file through a specified auxiliary program

Table 2

There may be one operation instruction set in the server regarding to upgrading of each software version, and each operation instruction set has a version. After receiving the upgrading request, the server finds operation instruction sets of all software each of which the version is higher than the version of the to-be-upgraded software in the upgrading request, and finally returns an operation instruction set, which is obtained by combining all found operation instruction sets, to an upgrading module. Figure 2 is a diagram illustrating the operation of combining the found operation instruction sets in the present disclosure. As shown in figure 2, suppose operation instruction sets of higher versions than version in the upgrading request are operation instruction set 201 of version 20467 and operation instruction set 202 of version 20532, when the two operation instruction sets are combined, a final instruction set 203 is obtained.

13) Take the to-be-updated file list information and the combined operation instruction set

as upgrading instruction information.

The upgrading instruction information is finally organized by the server in form of XML to obtain a datasheet, and the datasheet is sent to the upgrading module of the client. The XML datasheet mainly includes “to-be-updated file list”, “operation instruction set” and some other additional information. For example, the server may further find function description information of the latest version of the to-be-upgraded software, and add the function description information to the upgrading instruction information. The function description information is used for illustrating new functions brought to the user by the upgrading, including some description information and preview pictures, so that the user can obtain a general understanding of the upgrading before the upgrading is performed.

Content of a typical upgrading instruction table in XML format is shown by the following XML code:

```
<?xml version = "1.0" encoding="utf-8" ?>
<response>
  <features>
    <feature>
      <id>10001</id>
      <name>multiple accounts</name>
      <description> Simultaneous login function </description>
      <pic_1>http://dl_test.aaa.com/aaafile/B-001.png</pic_1>
      <pic_2> http://dl_test.aaa.com/aaafile/B-002.png </pic_2>
    </feature>
  </features>
  <files>
    <file>
      <name>StringBundle.xml</name>
      <path>%INSTALL_DIR%/AuI18N/2052</path>
      <size>16099</size>
      <md5>6bb5a6660683c3b688a3f2be2fbd6fba</md5>
      <key_file>1</key_file>
    </file>
    <file>
      <name>config.xml</name>
```

namely the old version files in the installation directory of the to-be-upgraded software are replaced with the downloaded new files.

In most cases, a to-be-replaced target file may not be modifiable or replaceable (the file may be being scanned by virus killing software, or be being used by a certain process), so it is needed to make all-wave safety mechanism assurance for these cases. Since program files of the to-be-upgraded software are usually interdependent, if one file is not replaced successfully, it is highly possible that the program cannot run any longer. In view of the, two safety assurance mechanisms are made in the present disclosure.

A) Before the combination of files (namely updating files) is performed, it is determined whether the to-be-upgraded software is running, when the to-be-upgraded software is running, the user is prompted to close the software, and corresponding files are updated after the software is closed; otherwise, corresponding files are directly upgraded. In the present disclosure, the upgrading module and the to-be-upgraded software run in two processes, and the operation of closing the to-be-upgraded software will not affect the current upgrading module.

B) Before the files are combined, local to-be-updated original files are backed up, namely to-be-replaced old version files are backed up to a temporary directory, and then the file update is performed. When it appears that a file is not updated successfully during the updating process, it is determined whether the file must be updated successfully. When the file must be updated successfully, it means that the file is indispensable for normal running of the program, and it is necessary to restore upgraded files using the backup files, to guarantee that “although the upgrading is unsuccessful, the old version can still run”. When the file is not indispensable to be replaced successfully, the latest downloaded file corresponding to the file may be registered in a restart replacement service of the operating system (such as the WINDOWS system), so that the file may be updated by starting a replacement service when the system is restarted. Whether the file must be replaced successfully may be determined according to a “key file” field in the to-be-updated file list. In the case, when the server generates the to-be-updated file list, it is necessary to include a “key file” field in description information of each file to indicate whether the file is a key file, namely whether the file must be replaced successfully. For example, most underlying components are indispensable for running a program, so these components may be taken as key files, but as to extended resource packages and plugins, they will not bring great impact to normal operation of the whole system, so they may not

be taken as key files. The client checks the “key file” field of a file when the file is not updated successfully, when the file is a key file, the client determines that the file must be replaced successfully; otherwise the file is not one that must be replaced successfully.

Block 305, after the combination of files is finished, perform corresponding local upgrading operations according to the operation instruction set in the upgrading instruction information.

Each operation instruction in the operation instruction set includes: an operation instruction name which may be identified by a local client, and a relative path and a file name of a target file of the operation corresponding to the instruction. As for each operation instruction, the upgrading module may find a target file with the file name in the operation instruction from the relative path of the installation directory of the to-be-upgraded software, and perform an operation corresponding to the operation instruction name on the target file, such as registering a COM component in the file list, deleting the target file, processing the target file by adopting a program specified in the operation instruction, etc.

In a practical application, a great bandwidth may be occupied when a whole file is directly issued. It may be required for the client to download the whole file of several mega because of a little file change, which may waste a lot of time and network bandwidth, so the present disclosure also provides an embodiment that may save bandwidth and file downloading time.

In the embodiment, file information in the upgrading request sent to the server by the client further includes a HASH value of each file. After finding a modified file in the latest version software, the server may find a corresponding HASH value in the upgrading request according to a file name of the modified file, and find an original file with the same HASH value stored in the server according to the HASH value, perform a binary comparison on the original file and the latest version file with the same name as the original file, and generate a binary difference file, replace download path information of a file, which has the same name as the binary difference file and is carried in the to-be-updated file list, with download path information of the binary difference file, namely information of a to-be-updated file may include a file name, download path information of a binary difference file, relative path information and so on.

The binary difference file is usually called a diff file, the diff file is of a small size, and the diff file describes which binary-level adding, deleting or modifying operations

the to-be-upgraded software and the latest version software according to file information of the to-be-upgraded software in the upgrading request, generate upgrading instruction information according to software difference and issues the upgrading instruction information to the upgrading module 401 in the client.

Figure 5 is a diagram illustrating an upgrading service module in a server according to an embodiment of the present disclosure. With reference to figure 5, in a specific embodiment, the upgrading service module 402 in the server end includes the following modules specifically.

Version difference determining module 421, configured to compare the to-be-upgraded software with the latest version software according to file name and version information of each file, and determine file difference and version difference.

To-be-updated file determining module 422, configured to find newly-added files and modified files in the latest version software according to the file difference, and organize list information of the newly-added files and modified files to form to-be-updated file list information. The to-be-updated file list information at least includes download path information, relative path information of each to-be-updated file, and so on.

Operation instruction determining module 423, configured to find operation instruction sets of all software each of which the version is higher than version of the to-be-upgraded software in the upgrading request, and combine the found operation instruction sets. Each operation instruction in the operation instruction set at least includes: an operation instruction name which may be identified by a local client, and a relative path and a file name of a target file of operation corresponding to the instruction.

Upgrading instruction combining and sending module 424, configured to take the to-be-updated file list information and the combined operation instruction set as upgrading instruction information and issue the upgrading instruction information to a upgrading module which initiates the upgrading request.

Further, the upgrading service module 402 may further include:

Description information determining module 425, configured to find function description information of the latest version of the to-be-upgraded software, and send the function description information to the upgrading instruction combining and sending module 424. The upgrading instruction combining and sending module 424 may further add the function description information to the upgrading instruction information and

send the upgrading instruction information to the upgrading module of the client which initiates the upgrading request.

Figure 6 is a diagram illustrating the upgrading module in the client according to an embodiment of the present disclosure. With reference to figure 6, in a specific embodiment, the upgrading module 401 in the client includes the following modules specifically.

Self-upgrading module 411, configured to, before a software upgrading request is reported, interact with the server to determine whether the upgrading module 401 needs to be upgraded, when the upgrading module 401 needs to be upgraded, download files of a new version upgrading module, and replace files of the original upgrading module with the files of the new version upgrading module; otherwise, the software upgrading request is reported.

Upgrading inquiring module 412, configured to report the software upgrading request carrying version information of local to-be-upgraded software to the server, and receive upgrading instruction information returned by the server.

File downloading module 413, configured to download corresponding files according to file download paths in the to-be-updated file list information in the upgrading instruction information.

Difference file combining module 414, configured to, according to the to-be-updated file list information, update the downloaded files to corresponding relative paths of the installation directory of the local to-be-upgraded software.

Operation performing module 415, configured to, as for each operation instruction in the upgrading instruction information, find a target file with the file name in the operation instruction from the relative path of the installation directory of the to-be-upgraded software, and perform the operation instruction on the target file.

In an embodiment, the difference file combining module 414 needs to further implement a safety assurance, namely the difference file combining module 414 may further include:

Close prompting module, configured to, before corresponding files are updated, determine whether the to-be-upgraded software is running, when the to-be-upgraded software is running, prompt the user to close the software, and corresponding files will be updated after the software is close, otherwise, corresponding files are directly updated.

Backup operating module, configured to, before corresponding files are updated,

back up local to-be-updated original files, when a file is not updated successfully, determine whether the file must be updated successfully, when the file must be updated successfully, use the backup files to restore all upgraded files; otherwise, register the latest download file corresponding to the file in a restart replacement service of the operating system, and start the replacement service to update the file when the system is restarted.

In order to further save bandwidth resource and time for downloading files, in another embodiment, the to-be-updated file determining module 422 may further include a binary difference file generating module, which is configured to, after the to-be-updated file determining module 422 finds a modified file in the latest version software, find a corresponding HASH value in the upgrading request according to a file name of the modified file, and find an original file which is stored at the server end and has the same HASH value according to the HASH value, perform a binary comparison on the original file and the latest version file which has the same file name as the original file, generate a binary difference file, and replace download path information of a file, with the same name as the binary difference file, in the to-be-updated file list with download path information of the binary difference file.

Correspondingly, the difference file combining module 414 in the client may further include a binary difference combining module, which is configured to, after downloading the binary difference file, combine the binary difference file and a file which has the same name as the binary difference file and is located at the relative path of the installation directory of the local to-be-upgraded software.

The present disclosure also provides a storage medium, which is used for storing instructions which instructs a machine to execute the method for upgrading software as described above. Specifically, a system or device configured with the storage medium may be provided, and the storage medium stores software program code for implementing function of any embodiment described above, and a computer (or CPU or MPU) of the system or device should be able to read and execute the program code stored in the storage medium.

In such case, the program code read from the storage medium can implement function of the any embodiment of the above-mentioned embodiments, so the program code and storage medium storing the program code make a part of the present disclosure.

Embodiments of storage medium for providing program codes include floppy disk,