UNITED STATES DISTRICT COURT
DISTRICT OF MASSACHUSETTS
EASTERN DIVISION

RED BEND LTD., and RED BEND SOFTWARE INC., Plaintiffs, v. GOOGLE INC., Defendant.	Civil Action No. 09-cv-11813-DPW **DECLARATION OF JENNIFER C. TEMPESTA IN SUPPORT OF PLAINTIFFS' MOTION FOR A PRELIMINARY INJUNCTION ENJOINING GOOGLE'S INFRINGEMENT**

I, Jennifer C. Tempesta, declare as follows:

1. I am a member in good standing of the New York State bar and am an associate with the law firm of Baker Botts, L.L.P., counsel of record for Plaintiffs in the above captioned matter. I submit this declaration based upon personal knowledge. If called upon as a witness, I could, and would, competently testify to the truth of each statement herein.

2. Attached hereto as Exhibit 1 is a true and correct copy of U.S. Patent No. 6,546,552.

3. Attached hereto as Exhibit 2 is a true and correct copy of the webpage available at: http://blog.chromium.org/2009/07/smaller-is-faster-and-safer-too.html (last accessed November 17, 2009).

4. Attached hereto as Exhibit 3 is a true and correct copy of the webpage available at: http://dev.chromium.org/developers/design-documents/software-updates-courgette (last accessed November 17, 2009).

5. Attached hereto as Exhibit 4 is a true and correct copy of the webpage available at: http://www.chromium.org/developers/how-tos/get-the-code (last accessed November 17, 2009).

6. Attached hereto as Exhibit 5 is a true and correct copy of the webpage available at: http://src.chromium.org/viewvc/chrome/trunk/src/courgette/win32_x86_patcher.h?view=markup&sortdir=down (last accessed November 17, 2009).

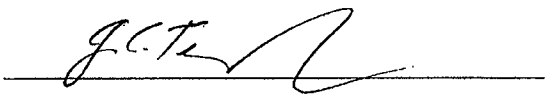
7. Attached hereto as Exhibit 6 is a true and correct copy of the webpage available at: http://src.chromium.org/viewvc/chrome/trunk/src//LICENSE?revision=1489 (last accessed November 17, 2009).

8. Attached hereto as Exhibit 7 is a true and correct copy of the webpage available at: http://android-developers.blogspot.com/2009/09/note-on-google-apps-for-android.html (last accessed November 17, 2009).

9. Attached hereto as Exhibit 8 is a true and correct copy of the webpage available at: http://www.redbend.com/pdf/CorporateProfile.pdf (last accessed on November 17, 2009).

10. Attached hereto as Exhibit 9 is a true and correct copy of correspondence from Plaintiffs to Google dated September 3, 2009.

I declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct and that this declaration was executed on November 17, 2009 at New York, New York.

By: 

Jennifer C. Tempesta

EXHIBIT 1

US006546552B1

(12) **United States Patent**
Peleg

(10) **Patent No.: US 6,546,552 B1**
(45) **Date of Patent: Apr. 8, 2003**

(54) **DIFFERENCE EXTRACTION BETWEEN TWO VERSIONS OF DATA-TABLES CONTAINING INTRA-REFERENCES**

(75) Inventor: **Sharon Peleg**, Ramat Hasharon (IL)

(73) Assignee: **Red Bend Ltd.**, Ramat Hasharon (IL)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/376,512**

(22) Filed: **Aug. 18, 1999**

(30) **Foreign Application Priority Data**

Aug. 19, 1998 (IL) 125846

(51) **Int. Cl.**[7] **G06F 9/45**
(52) **U.S. Cl.** **717/170**
(58) **Field of Search** 717/11, 170

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,155,847	A	*	10/1992	Kirouac et al.	717/11
5,359,730	A	*	10/1994	Marron	709/100
5,752,039	A	*	5/1998	Tanimura	707/203
5,761,649	A	*	6/1998	Hill	705/27
5,790,760	A	*	8/1998	Arima	706/45
5,815,704	A	*	9/1998	Shimotsuji et al.	382/190
5,832,520	A	*	11/1998	Miller	707/203
6,367,075	B1	*	4/2002	Kruger et al.	717/169

FOREIGN PATENT DOCUMENTS

EP	0 472 812 A		3/1992	G06F/9/45
JP	4-242829	*	8/1992	G06F/9/06
JP	5-091550	*	4/1993	G06F/9/06
WO	WO 93/00633		1/1993	G06F/11/34
WO	WO 97/12508		4/1997	G06F/11/00

OTHER PUBLICATIONS

Horwitz, Identifying the Semantic and Textual Differences Between Two Versions of a Program, ACM, pp. 234–245.*
International Search Report for PCT/IL99/00446, dated Dec. 12, 1999 (3 pages).
Coppieters, K.: "A Cross–Platform Binary Diff", Dr. Dobb's Journal, US, San Mateo, California, pp. 32, 35–36, XP 000610668.

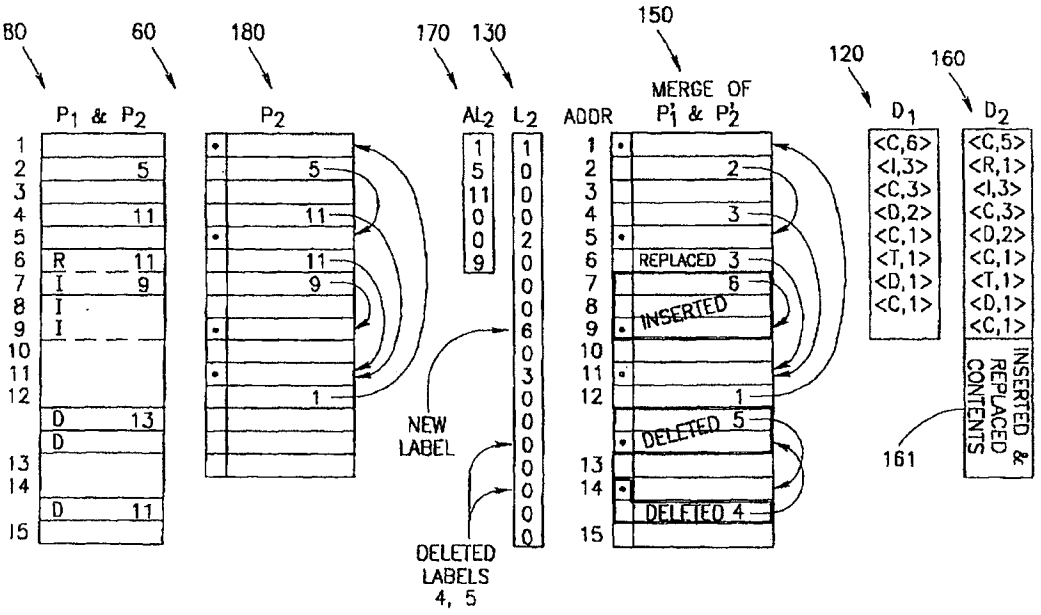
* cited by examiner

Primary Examiner—Gregory Morse
Assistant Examiner—John Q. Chavis
(74) *Attorney, Agent, or Firm*—Fitch, Even, Tabin, & Flannery

(57) **ABSTRACT**

A method for generating a compact difference result between an old program and a new program. Each program including reference entries that contain reference that refer to other entries in the program. The method includes the steps of scanning the old program and for each reference entry perform steps that include replacing the reference of the entry by a distinct label mark, whereby a modified old program is generated. There is further provided the step of scanning the new program and for each reference entry perform steps that include replacing the reference of the entry by a distinct label mark, whereby a modified new program is generated. There is still further provided the step of generating the specified difference result utilizing directly or indirectly the modified old program and modified new program.

68 Claims, 4 Drawing Sheets



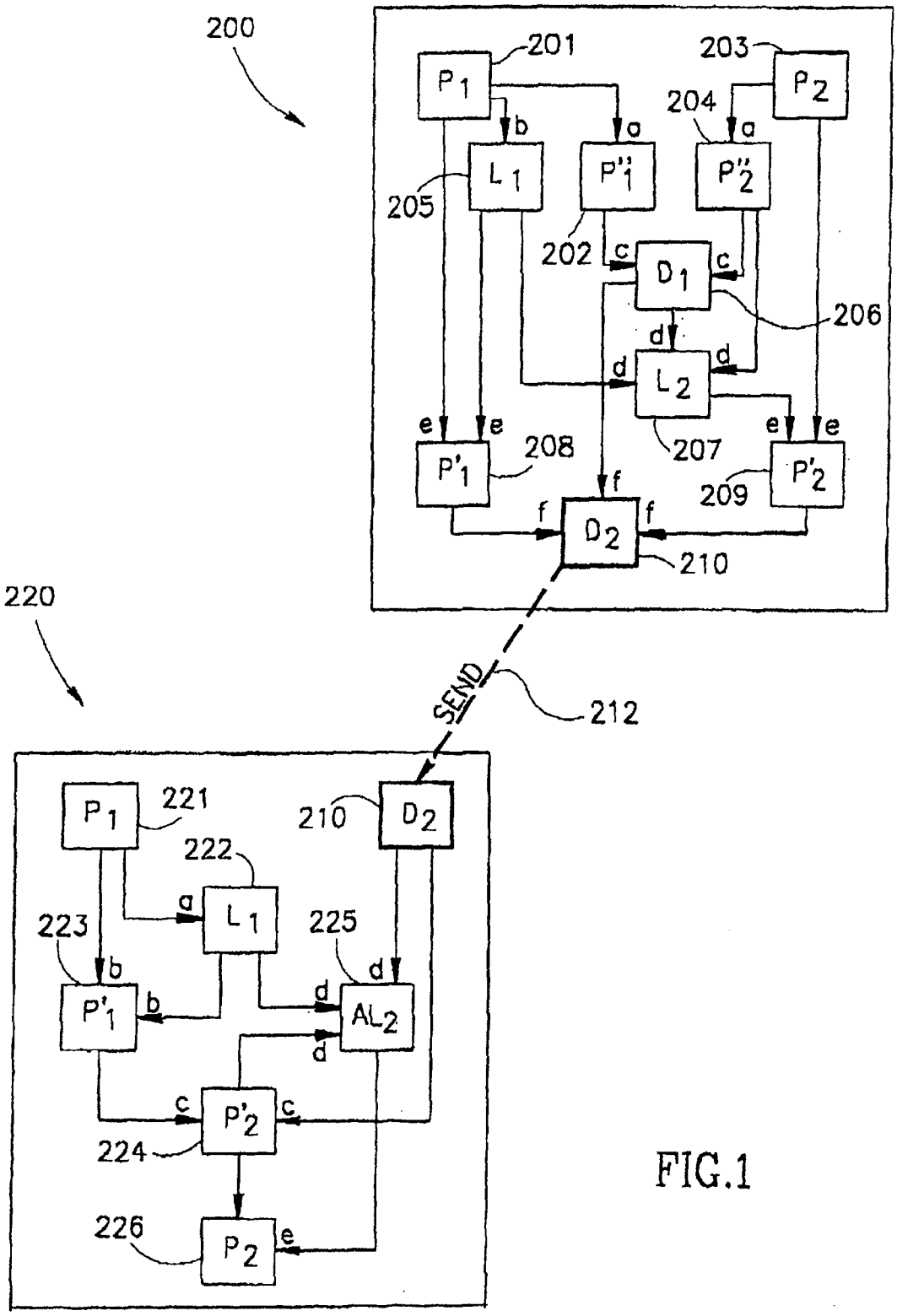

200
201
203
P1
P2
204
b
a
a
L1
P''1
P''2
205
202
c
c
D1
206
d
d
d
L2
e
e
e
e
P'1
208
207
P'2
209
f
f
f
D2
210
220
SEND
212
P1
221
210
D2
222
a
L1
223
225
b
d
d
P'1
b
AL2
d
c
c
P'2
224
226
P2
e

FIG.1

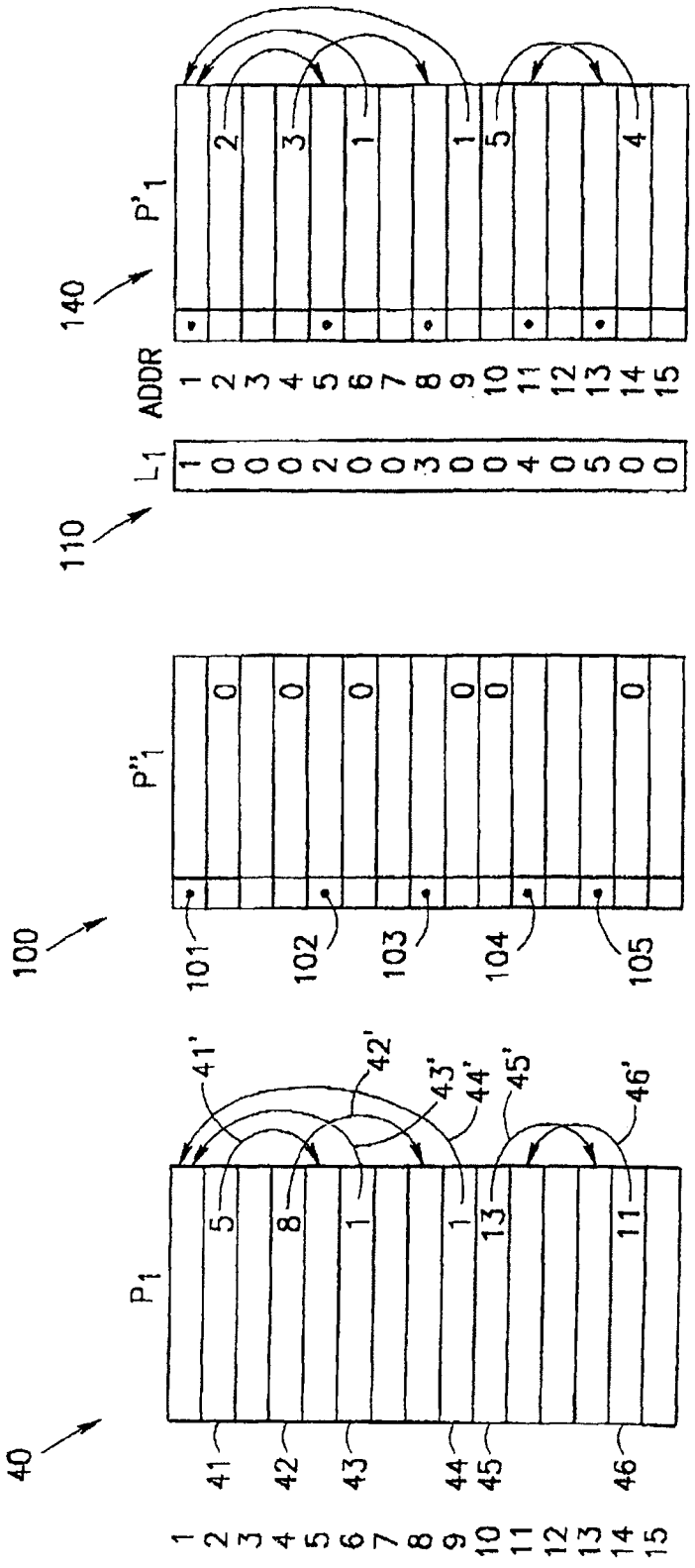

140
P'1
2
3
1
1
5
4
ADDR
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
110
L1
1 0 0 0 2 0 0 3 0 0 4 0 5 0 0
100
P''1
0 0 0 0 0 0 0
101 102 103 104 105
40
P1
41' 42' 43' 44' 45' 46'
5 8 1 1 13 11
41 42 43 44 45 46
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

FIG.2A

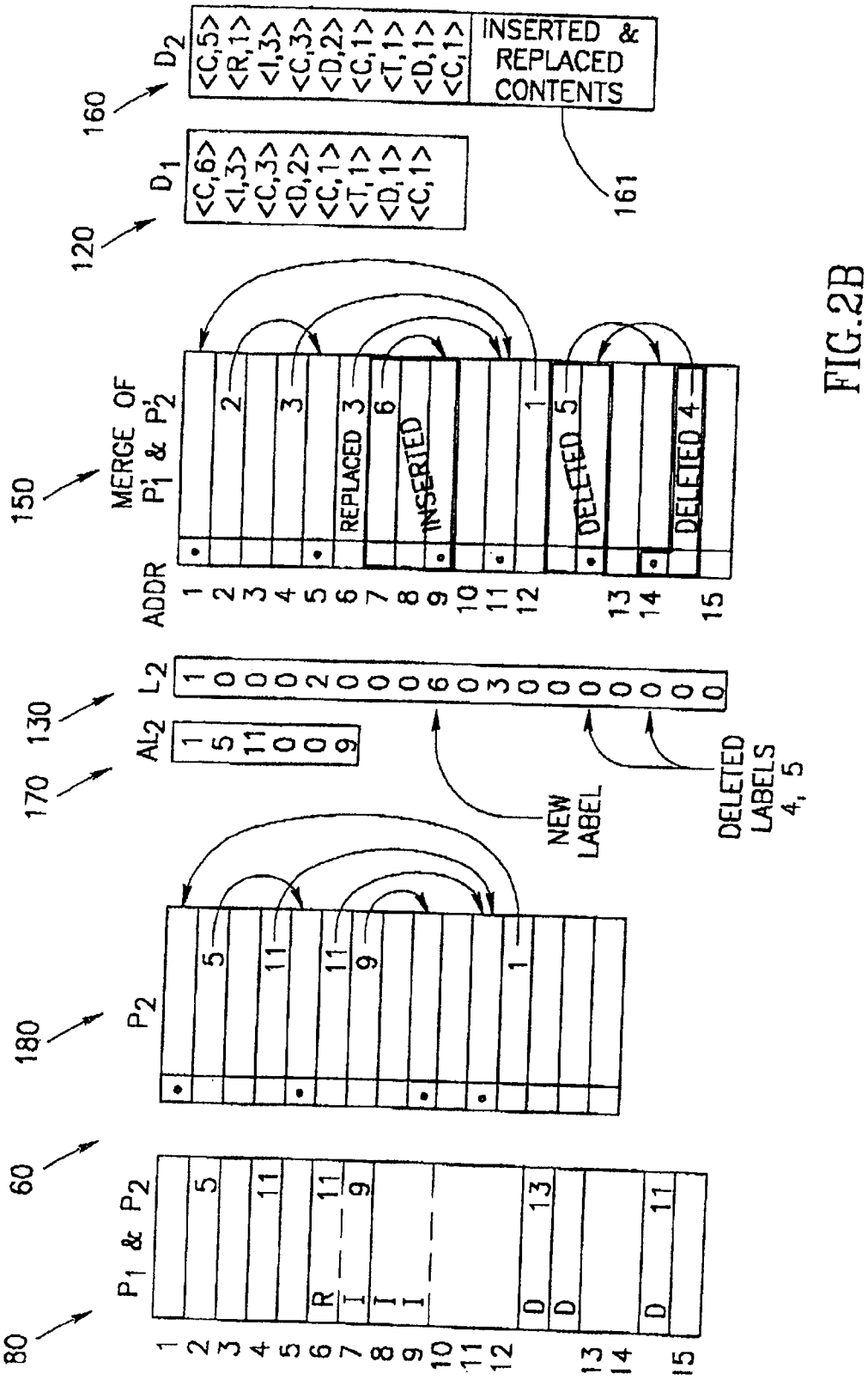

160
D2
<C,5>
<R,1>
<1,3>
<C,3>
<D,2>
<C,1>
<T,1>
<D,1>
<C,1>
INSERTED & REPLACED CONTENTS
161
120
D1
<C,6>
<1,3>
<C,3>
<D,2>
<C,1>
<T,1>
<D,1>
<C,1>
150
MERGE OF P'1 & P'2
REPLACED 3
INSERTED
DELETED 5
DELETED 4
ADDR
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
130
L2
AL2
170
NEW LABEL
DELETED LABELS 4, 5
180
P2
60
P1 & P2
80
R
I
D
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

FIG.2B

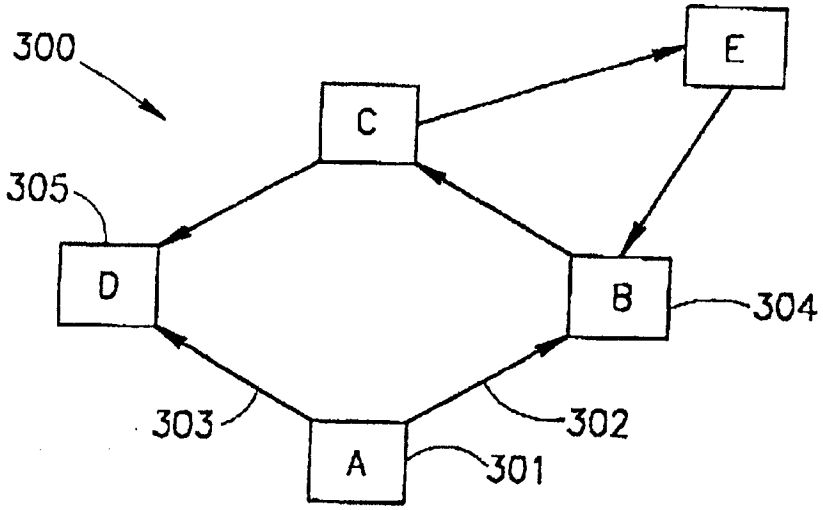


FIG.3A

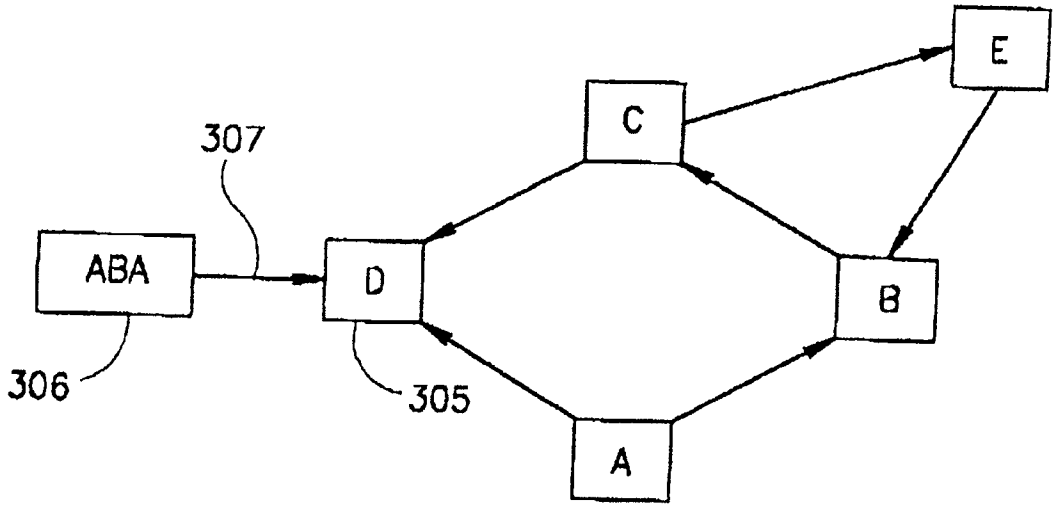


FIG.3B

DIFFERENCE EXTRACTION BETWEEN TWO VERSIONS OF DATA-TABLES CONTAINING INTRA-REFERENCES

FIELD OF THE INVENTION

This invention relates generally to updating computer programs.

BACKGROUND OF THE INVENTION

With the ever increasing use of remote communication and in particular the Internet, new applications have been introduced such as commercial trade over the Internet, electronic supermarkets, distribution of computer products over the Internet, and others.

Both the popularity and availability of access to the Internet for common users have encouraged not only the distribution of products, but also the upgrade and update of the product under question from a remote site, using, to this end, the Internet infra-structure.

Turning to a specific example of computer programs, an old program is installed at a remote client site and is subject to be upgraded to a new program, where the latter includes some modifications as compared to the old program.

In order to catty out the update at the remote client site (through the network), the provider should, preferably, generate a difference result representative of the difference between the old program and the new program, and send the resulting file through the Internet to the remote client site. The client, in turn, invokes appropriate utility, which incorporates the differences in the old program, thereby generating the desired new program at the client site, The specified procedure caries the obvious advantages in that on the one band, the provider does not need to be present at the client site and, on the other hand, only the difference result and not the entire new program is sent to the client. Assuming, for example, that a modified Office '97 package (commercially available from Microsoft Inc. USA) should be sent to clients, since the compressed size of programs of the package occupies ten of Mega-bytes, and, further considering the relatively low throughput of the Internet and the bottleneck of the modem throughput at the client end (say an average of 33,600 bps), it is easy to understand that transmitting the entire new package through the network is practically infeasible.

Normally, the volume of the difference result is significantly smaller than that of the raw new program and, accordingly, sending only the difference result data rather than the entire new program, is more efficient. This notwithstanding, and as will be explained in greater detail below, applying known per se file difference applications (such as techniques utilized by diff utilities of the UNIX operating systems or a similar diff utility of the GNU project from FSF) in order to generate a difference result between the old program and the new program, normally results in a relatively large amount of data, even if the modifications that were introduced to the old program (in order to generate the new program) are very few. Thus, consider, for example, an old program where few new instructions are inserted and few others are deleted in order to bring about the new program. The difference result between, the old program and the new program will not only reveal the inserted and deleted instructions, but also all those entries that jump, jump on condition, call functions, reference to data and possibly others (referred to, collectively, as reference entries—see glossary below) which, by nature, specify a

target address (reference) as an integral part of the command. The latter addresses may have been changed due to the fact that some instructions were added and others deleted. It is important to note that the reference entries that are modified are not those that were inserted, and obviously not those that were deleted. In fact, insertion of only one new entry may result in the plurality of altered reference entries which will naturally be reflected in the difference result and obviously will inflate its volume.

It is accordingly appreciated that despite the fact that the actual change between the old and new program is very limited, the resulting file difference is relatively large. The same problem is encountered in other applications, which employ data tables (see Glossary below), that are structured like a program, and are subject to updates in the manner specified.

There is accordingly a need in the art to provide for an efficient tool which will result in significantly smaller volumes of difference results between old programs and new programs, as compared to hitherto known techniques for accomplishing difference result. The proposed tool is useful for various applications including, but not limited to, incremental software updates and version control.

There is yet another need in the art to provide for an efficient tool which will result in significantly smaller volumes of difference results between old data tables and new data tables.

GLOSSARY

There follows a glossary of terms, some of which are conventional and others have been coined:

Data Table—a table of entries, each may have a different size;

Entry—a data fable includes entries, each of which is an addressable unit that contains data;

Address—a number which is uniquely assigned to a single entry by which that entry is accessed; In the following description, the terms entry and address are occasionally used interchangeably.

Reference—a part of the data appearing in an entry in the data table which is used to refer to some other entry from the same data table. A reference can be either an address or a number used to compute an address. Entries that include references are designated also as reference entries.

Label—an abstract notation of an entry which is referred by another entry of the same data table through a reference.

Old Data Table—a data table (or portion of a data table) that is to be updated (possibly from remote site) so as to generate a new data table (or portion of a new data table). Insofar as remote update is concerned, it is normally, although not necessarily, transmitted through a communication network such as the Internet. It should be noted that whilst for convenience of explanation only, the description focuses predominately on the Internet, the invention is by no means bound by this specific example.

As an example, a data table can be an executable program either as a loaded program in machine-memory or as an executable-file. In this example, entries are individual machine instructions of the program or the individual data elements used by the program.

Instructions and data elements of a program may contain addresses to other instructions or data elements and are

regarded as references. Such references can be detected by a process of disassembly applied on the program or, if given, by analyzing a relocation table attached to executable programs by link-editors that create them.

Another example of a data table is a group of inter-linked data records stored in an array of bytes where records contain addresses of other data records. The format of the records and the way they are laid out in the array are known, and the analysis and decomposition of such array is possible.

Old program—an example of old data table: a program (or portion of a program) that is to be updated so as to generate a new program (or portion of a program).

It should be further noted that reference to the old program and the new program is made for convenience of explanation only, and encompasses inter alia the upgrade of the old program to the new program (e.g. due to an upgrade in versions), modifications of the old program to the new program, (e.g. due to corrections of bugs in the old program), and changing from a first old program to a second (and possibly different) new program;

SUMMARY OF THE INVENTION

For convenience of explanation, the invention is described with reference to a specific example of computer programs. The invention is by no means bound by this particular example.

As explained above, applying a known per se file difference utility to an old program and a new program normally results in a relatively large amount of data, even if the modifications that were introduced to the old program (in order to generate the new program) are very few. The present invention is based on the observation that the relatively large size of the difference result stems from the alterations of reference in reference entries as a result of other newly inserted entries (and/or entries that were deleted).

On the basis of this observation, the invention aims at generating a modified old program and a modified new program, wherein the difference in references in corresponding entries in said new and old programs as explained above, will be reflected as invariant entries in the modified old and new programs. The net effect is that the invariant reference entries (between the modified old program and the modified new program), will not appear in the difference result, thereby reducing its size as compared to a conventional difference result obtained by using hitherto known techniques.

Accordingly, the invention provides for a method for generating a compact difference result between an old program and a new program; each program including reference entries that contain reference that refer to other entries in the program; the method comprising the steps of:

(a) scanning the old program and for substantially each reference entry perform steps that include:
  (i) replacing the reference of said entry by a distinct label mark, whereby a modified old program is generated;

(b) scanning the new program and for substantially each reference entry perform steps that include:
  (i) replacing the reference of said entry by a distinct label mark, whereby a modified new program is generated;

(c) generating said difference result utilizing directly or indirectly at least said modified old program and modified new program.

The invention further provides for a method for performing an update in an old program so as to generate a new program; each program including reference entries that contain reference that refer to other entries in the program; the method comprising the steps of:

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old program and a modified new program;

(b) scanning the old program and for substantially each reference entry perform, steps that include:
  (i) replacing the reference of said entry by a distinct label mark, whereby the modified old program is generated;

(c) reconstituting the modified new program utilizing at least said compact difference result and said modified old program; said modified new program is differed from said new program at least in that substantially each reference entry in said new program is replaced in said modified new program by a distinct label mark;

(d) reconstituting said new program utilizing directly or indirectly at least said compact difference result and said modified new program.

Still further, the invention provides for a method for generating a compact difference result between an old program, and a new program; each program including reference entries that contain reference that refer to other entries in the program; the method comprising the steps of:

(a) generating a modified old program utilizing at least said old program;

(b) generating a modified new program utilizing at least said new program, said modified old program and modified new program have at least the following characteristics:
  (i) substantially each reference in an entry in said old program that is different than corresponding entry in said new program due to delete/insert modifications that form part of the transition between said old program and new program are reflected as invariant references in the corresponding entries in said modified old and modified new programs;

(c) generating said compact difference result utilizing at least said modified new program and modified old program.

Yet further, the invention provides for a method for performing an update in an old program so as to generate a new program; each program including reference entries that contain reference that refer to other entries in the program; the method comprising the steps of:

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old program and a modified new program;

(b) generating a modified old program utilizing at least said old program;

(c) reconstituting a modified new program utilizing directly or indirectly at least said modified old program and said compact difference result; said modified old program and modified new program have at least the following characteristics:
  (i) substantially each reference in an entry in said old program that is different than corresponding entry in said new program due to delete/inset modifications that form part of the transition between said old program and new programs are reflected as invariant references in the corresponding entries in said modified old and modified new programs;

(d) reconstituting said new program utilizing directly or indirectly at least said compact difference result and said modified new program.

According to another aspect, the invention provides for a system for generating a compact difference result between an old program and a new program; each program including reference entries that contain reference that refer to other entries in the program; the system comprising a processing device capable of:

(a) scanning the old program and for substantially each reference entry perform steps that include:
(i) replacing the reference of said entry by a distinct label mark, whereby a modified old program is generated;

(b) scanning the new program and for substantially each reference entry perform steps that include:
(i) replacing the reference of said entry by a distinct label mark, whereby a modified new program is generated;

(c) generating said difference result utilizing directly or indirectly at least said modified old program and modified new program.

Still further, the invention provides for a system for performing an update in an old program so as to generate a new program; each program including reference entries that contain reference that refer to other entries in the program; the system comprising a processing device capable of:

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old program and a modified new program;

(b) scanning the old program and for substantially each reference entry perform steps that include:
(i) replacing the reference of said entry by a distinct label mark, whereby the modified old program is generated;

(c) reconstituting the modified new program utilizing at least said compact difference result and said modified old program; said modified new program is differed from said new program at least in that substantially each reference entry in said new program is replaced in said modified new program by a distinct label mark;

(d) reconstituting said new program utilizing directly or indirectly at least said compact difference result and said modified new program.

The invention further provides for a system for generating a compact difference result between an old program and a new program; each program including reference entries that contain reference that refer to other entries in the program; the system comprising a processing device capable of:

(a) generating a modified old program utilizing at least said old program;

(b) generating a modified new program utilizing at least said new program, said modified old program and modified new program have at least the following characteristics:
(i) substantially each reference in an entry in said old program that is different than corresponding entry in said new program due to delete/insert modifications that form part of the transition between said old program and new program are reflected as invariant references in the corresponding entries in said modified old and modified new programs;

(c) generating said compact difference result utilizing at least said modified new program and modified old program.

Still further, the invention provides for a system for performing an update in an old program so as to generate a new program; each program including reference entries that contain reference that refer to other entries in the program; the system comprising a processing device capable of:

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old program and a modified new program;

(b) generating a modified old program utilizing at least said old program;

(c) reconstituting a modified new program utilizing directly or indirectly at least said modified old program and said compact difference result; said modified old program and modified new program have at least the following characteristics:
(i) substantially each reference in an entry in said old program that is different than corresponding entry in said new program due to delete/inset modifications that form part of the transition between said old program and new program are reflected as invariant references in the corresponding entries in said modified old and modified new programs;

(d) reconstituting said new program utilizing directly or indirectly at least said compact difference result and said modified new program.

Yet further, the invention provides for a method for generating a compact difference result between an old data table and a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the method comprising the steps of:

(a) scanning the old data table and for substantially each reference entry perform steps that include:
(i) replacing the reference of said entry by a distinct label mark, whereby a modified old data table is generated;

(b) scanning the new data table and for substantially each reference entry perform steps that include:
(i) replacing the reference of said entry by a distinct label mark, whereby a modified new data table is generated;

(c) generating said difference result utilizing directly or indirectly at least said modified old data table and modified new data table.

Moreover, the invention provides for a method for performing an update in an old data table so as to generate a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the method comprising the steps of:

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old data table and a modified new data table;

(b) scanning the old data table and for substantially each reference entry perform steps that include:
(i) replacing the reference of said entry by a distinct label mark, whereby the modified old data table is generated;

(c) reconstituting the modified new data table utilizing at least said compact difference result and said modified old data table; said modified new data table is differed from said new data table at least in that substantially each reference entry in said new data table is replaced in said modified new data table by a distinct label mark;

(d) reconstituting said new data table utilizing directly or indirectly at least said compact difference result and said modified new data table.

The invention her provides for a method for generating a compact difference result between an old data table and a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the method comprising the steps of:

(a) generating a modified old data table utilizing at least said old data table;

(b) generating a modified new data table utilizing at least said new data table, said modified old data table and modified new data table have at least the following characteristics:

(i) substantially each reference in an entry in said old data table that is different than corresponding entry in said new data table due to delete/insert modifications that form part of the transition between said old data table and new data table are reflected as invariant references in the corresponding entries in said modified old and modified new data tables;

(c) generating said compact difference result utilizing at least said modified new data table and modified old data table.

The invention provides for a method for performing an update in an old data table so as to generate a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the method comprising the steps of:

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old data table and a modified new data table;

(b) generating a modified old data table utilizing at least said old data table;

(c) reconstituting a modified new data table utilizing directly or indirectly at least said modified old data table and said compact difference result; said modified old data table and modified new data table have at least the following characteristics:

(i) substantially each reference in an entry in said old data table that is different than corresponding entry in said new data table due to delete/inset modifications that form part of the transition between said old data table and new data table are reflected as invariant references in the corresponding entries in said modified old and modified new data tables;

(d) reconstituting said new data table utilizing directly or indirectly at least said compact difference result and said modified new data table.

The invention further provides for a system for generating a compact difference result between an old data table and a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the system comprising a processing device capable of:

(a) scanning the old data table and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby a modified old data table is generated;

(b) scanning the new data table and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby a modified new data table is generated;

(c) generating said difference result utilizing directly or indirectly at least said modified old data able and modified new data table.

Still further, the invention provides for a system for performing an update in an old data table so as to generate a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the system comprising a processing device capable of:

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old data table and a modified new data table;

(b) scanning the old data table and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby the modified old data table is generated;

(c) reconstituting the modified new data table utilizing at least said compact difference result and said modified old data table; said modified new data table is differed from said new data table at least in that substantially each reference entry in said new data table is replaced in said modified new data table by a distinct label mark;

(d) reconstituting said new data table utilizing directly or indirectly at least said compact difference result and said modified new data table.

Moreover, the invention provides for a system for generating a compact difference result between an old data table and a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the system comprising a processing device capable of:

(a) generating a modified old data table utilizing at least said old data table;

(b) generating a modified new data table utilizing at least said new data table, said modified old data table and modified new data table have at least the following characteristics:

(i) substantially each reference in an entry in said old data table that is different than corresponding entry in said new data table due to delete/insert modifications that form part of the transition between said old data table and new data table are reflected as invariant references in the corresponding entries in said modified old and modified new data tables;

(c) generating said compact difference result utilizing at least said modified new data table and modified old data table.

The invention provides for a system for performing an update in an old data table so as to generate a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the system comprising a processing device capable of:

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old data table and a modified new data table;

(b) generating a modified old data table utilizing at least said old data table;

(c) reconstituting a modified new data table utilizing directly or indirectly at least said modified old data table and said compact difference result; said modified old data table and modified new data table have at least the following characteristics:

(i) substantially each reference in an entry in said old data table that is different than corresponding entry in said new data table due to deleted/inset modifications that form part of the transition between said

old data table and new data table are reflected as invariant references in the corresponding entries in said modified old and modified new data tables;

(d) reconstituting said new data table utilizing directly or indirectly at least said compact difference result and said modified new data table.

BRIEF DESCRIPTION OF THE DRAWINGS

In order to understand the invention and to see how it may be carried out in practice, a preferred embodiment will now be described, by way of non-limiting example only, with reference to the accompanying drawings, in which:

FIG. **1** is a schematic illustration of a sequence of operation according to one embodiment of the invention;

FIG. **2** shows exemplary old and new programs and the various interim results that are obtained by applying the sequence of operation of FIG. **1**; and

FIG. **3** is an exemplary data table, presented in the form of graph which is subject to difference extraction in accordance with the invention.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

In FIG. **1**, module (**10**) represents the sequence of operations performed e.g. at the provider site, for generating the difference result according to one embodiment of the invention. The sequence of operation will now be described with reference to exemplary old and new programs (FIG. **2**). It should be noted that the specified sequence of operations may be carried out at any known per se platform including, but not limited to, a conventional P.C., a computer network, etc., all as known per se.

Thus, P_1 stands for the old program that includes entries (**41**), (**42**), (**43**), (**44**), (**45**) and (**46**) that contain references to entries **5**,**8**,**1**,**1**,**13**, and **11** respectively, (as indicated by arrows (**41'**) to (**46'**).

P_2 stands for the new program which was generated (or could have been generated) by the sequence of modifications as depicted in the imaginary memory table (**80**). Said sequence of modifications (either real or imaginary), constitutes a transition sequence between P_1 and P_2.

As shown in entry no. **6**, the reference to address '1' was replaced ('R' stands for replaced) by reference to address '11', e.g. due to a patch introduced by the programmer. Following the 6th entry, 3 new entries were inserted ('I' stands for inserted). The newly inserted entries reside at addresses **7** to **9**. The next three entries which originally resided in addresses **7** and **9** at P_1 are now shifted (due to the 3 inserted entries) to addresses **10** to **12** in P_2.

Next, entries **10** and **11** in P_1 were deleted (D), and are therefore not assigned with any address at P_2. Entries **12** and **13** in P_1 remain intact and reside in addresses **13** and **14** in P_2. Entry **14** in P_1 was deleted (D) and it is marked as such (with no address) in the memory table (**80**). Lastly, entry (**15**) remains intact and therefore resides in entry **15** in P_2.

Having reviewed the sequence of modifications that constitute the transition from P_1 to P_2, there follows a brief review on how the specified modifications affected the reference entries of P_1 and P_2.

Thus, and as expected, the reference **5** in entry **2** remains intact and it will therefore not appear in the difference result between, P_1 and P_2.

The reference **8** in entry **4** of P_1 is flow modified by reference **11** in entry **4** of P_2. The modification in the reference (from **8** to **11**) in entry **4** is caused by the insertion of entries **7** to **9** in P_2, which obviously shifted entry **8** (in P_1) to entry **11** (in P_2). Before proceeding any further, it should be noted that applying conventional file difference application to P_1 and P_2 will obviously reflect that entry **4** has been changed since the reference **8** has been changed to **11**. Those versed in the art will readily appreciate that according to the invention, it is desired to neutralize this change, since it has occurred solely due to the fact that other entries have been affected (i.e. entries **7** to **9**). It is accordingly an object of the invention to give rise to a situation where modifications of this kind will be modified to invariant references with the obvious consequence that they are not reflected in the difference result, thereby keeping the latter relatively compact.

Reverting now to the example of FIG. **2**, the next reference **1** resides in entry **6** of P_1. As recalled, this entry was intentionally modified to **11**, and as expected in entry **6** of P_2, contains reference **11**. Unlike the previous reference modification which stems from shifts in the program and which therefore should be neutralized from appearing in the difference results, the current modification is applied to the entry under question (i.e. the reference in entry **6** has been changed from **1** to **11**) and should be reflected in the difference result. Turning now to entry **7** of P_2, it forms part of the inserted entries, and therefore reference **9** thereof is obviously not reflected in P_1. Of course, since entry **7** has been inserted, it is expected to appear in the difference results.

Entry **10**, with its associated reference **13**, has been deleted from P_1, and as expected, it does not appear in P_2 and should, of course, be indicated in the difference results as an entry for deletion.

Entry **9** with its associated reference **1** in P_1, corresponds to entry **12** in P_2. Since the reference in entry **12** remains **1**, it will not appear in the difference result.

Turing to the last reference entry **14** in P_1, it does not appear in P_2 (since it was deleted), and therefore it is expected to appear in the difference result as an entry for deletion.

Having described in general the differences between P_1 and P_2, as well as their effect on the difference result, attention is now directed also to FIG. **1** for describing one non-limiting realization of a system and method of the invention for accomplishing the desired difference result.

By this particular embodiment, the desired invariant references are accomplished by generating modified old and new programs wherein address references in entries are replaced by label marks as follows:

a) Create $P"_1$ table from P_1 (steps (**201**) and (**202**) in FIG. **1**) and $P"_2$ table from P_2 (steps (**203**) and (**204**) in, FIG. **1**) by adding label marks and replacing references in entries with some fixed values. As shown in FIG. **2**, $P"_1$ is generated from P_1 by adding label marks to entries **1**, **5**, **8**, **11** and **13** (designated (**101**) to (**105**), respectively). As also shown in FIG. **2**, the references at entries **2**, **4**, **6**, **9** **10** and **14** were set to a fixed value and by this particular example 0. Although not shown in FIG. **2**, $P"_2$ is generated in a similar manner;

b) Create a translation table L_1 (step (**205**) in FIG. **1**) between entry references in P_1 and label distinct values. Thus, and as shown in FIG. **1** (**110**), a distinct label number is assigned to each label mark of $P"_1$. By this particular example, the distinct labels are assigned in ascending order and, as shown, labels marks (**101**) to (**105**) are assigned with the respective values 1 to 5;

c) $P"_1$ and $P"_2$ are compared giving rise to difference table D_1 (**206**) using file difference utilities of the kind specified above. As shown in FIG. **2**, D_1 (**120**) contains list of entries each having the structure of <X, n>, where X stands for C (copy), I (insert), D(delete), or T (Toggle) and n stands for the number of instructions. D_1 includes, in fact, a list of instructions for generating $P"_2$ from $P"_1$. By this particular example, D_1 includes the following entries: <C,6> which signifies that the first 6 entries of $P"_1$ should be copied to $P"_2$. Note that whilst the 6^{th} entry is different in P_1 and P_2 (i.e. the reference has been changed from **1** to **11**), this entry is the same in $P"_1$ and $P"_2$, since both references were set to 0 and, accordingly, the 6^{th} entry forms part of the copied part. The next entry <I,3>, signifies that three new entries should be inserted, and this accounts for the new three entries **7** to **9** that were inserted to P_2 (and are also reflected in $P"_2$). The next entry <C,3> stands for the three entries that reside in addresses **7** to **9** in, $P"_1$ (and P_1), and are shifted (without affecting their contents), to addresses **10** to **12** in $P"_2$ (and P_2). The next entry <D, 2> stands for the two entries (**10** and **11**) that were deleted from P_1 (and obviously also from $P"_1$). The next entry <C,1> stands for the entry **12** in $P"_1$ that was shifted (without affecting its contents) to entry **13** in $P"_2$. The next entry <T,1> is a special entry indicating that a change has occurred in the label of entry **13** of $P"_1$, i.e,. it was removed in the corresponding entry **14** in $P"_2$ for the simple reason that the entry that refers to this label was deleted (entry **10** in $P"_1$ (and P_1). <T,1> signifies, thus, that the entry **13** should be copied (to entry **14**), whilst deleting the label mark. Next, <D,,1> stands for the deleted entry (**14**) in $P"_1$, and <C,1> stands for copying the last intact entry that is to be copied from $P"_1$ to $P"_2$ (entry **15**).

d) Analyzing D_1 to determine the Position and size of deleted or inserted program fragments and apply equivalent changes to L_1 translation table to create L_2 translation table (**207**), which translates entry references in P_2 to their distinct label value in the following manner: For a non-inserted referred entry perform:

L_2[entry address]=L_1[matching entry address in P_1] (hereinafter the first condition)

For an inserted referred entry, perform:

L_2[entry address]=U( ) different than any value in L_1; U( ) signifies a label generation function for generating distinct labels (hereinafter the second condition). The function is of repeatable nature, namely, when activated in the same scenario it will always generate the same result. The latter characteristics will be clarified when the subsequent program reconstruction phase is described below.

The resulting L_2 table (**130**) is shown in FIG. **2**. Thus, the first six instructions fall in the first condition (i.e. non inserted entry) and, accordingly, the distinct labels (**1** and **2**) are simply copied from L_1.

The next three entries are inserted and, accordingly, the second condition applies and U( ) is activated to generate a distinct label value. Since, as shown in L_1, five labels are "occupied", the next free one is 6 and, accordingly, the reference entry **9** is assigned with the value **6**. Those versed in the art will readily appreciate that whilst in this example U( ) generates the distinct label values by simply incrementing the last occupied value by 1, this is only an example.

Turning now to the next three entries, <C,3>, the reference entry **11** falls in the first criterion and, accordingly, the label value (**3**) is taken from L_1.

The next entry in D_1 <D,2> is ignored, since it concerns two deleted entries.

The next entry, <C,1>, has no reference entry and therefore need not be processed for generating L_2.

The next entry, <T,1> corresponds to the deleted label (from entry **13** in L_1) and in this respect, is resembles the previous delete modification that is ignored.

The last two entries, <D,1> and <C,1>, do not involve reference entries and therefore need not be processed for the purpose of generating L_2.

e) Create P'_1 (**208**) and P'_2 (**209**) for P_1 and P_2 respectively, by replacing entry references with their translated values using L_1 and L_2 tables and by coping label marks from $P"_1$ and $P"_2$.

Step (e) will be described with reference to P'_1 and applies mutatis mutandis also to P'_2 (see **150** in FIG. **2**). Thus, the label marks (**101**) to (**105**) of $P"_1$ are copied to the respective locations in P'_1 (**140** in, FIG. **2**). Next, the reference entries of P_1 are replaced by their corresponding label marks as retrieved from L_1. More specifically, the reference **5** in entry **2** of P_1 is replaced by the corresponding label from L_1. As shown, label **2** resides in, entry **5** of L_1 and, accordingly, the reference in entry **2** in P'_1 is set to 2.

In a similar manner, the reference **8** in entry **4** of P_1 is replaced by label number **3** according to the label number (**3**) that resides in entry **8** of L_1. In a similar manner, the references **1**,**1**,**13** and **11** of entries **6**, **14** of P_1 are replaced by respective label numbers **1**,**1**,**5** and **4**.

f) Having generated P'_1 and P'_2, the final difference result D_2 is generated. To this end, D_1 is analyzed to determine the position of program fragments copied from P_1 to P_2 (i.e. in the example of FIG. **2**, the entries that fall in <C, x> or <T,x> commands; the <I, x> <D,x> commands are ignored>). For copied entries as derived from D_1, P'_1 and P'_2 are compared so as to generate D_2 (**210**) in the following manner:

f1) take each pair of matching entries in P'_1 and P'_2 (neither deleted nor inserted) that contain a (replaced) reference, and compare their replaced reference values. In the case of discrepancy, add a special modification command to reflect the difference.

f2) attach all the inserted program fragments and replaced values. These fragments are taken from P'_2, thus they contain label marks, and address references remain under L_2 translation.

Step (f) will now be exemplified with reference to the specific example of FIG. **2**. As recalled, only entries that fall in a <C,x> command are of interest for the f1 step analysis. Thus, the first non-inserted entry of D_1, i.e. <C,6> is analyzed. According to step f1, only the six^{th} entry contains a replaced reference (reference 1 in P'_1 as compared to reference **3** in P'_2). Accordingly, The first command of D_1 <C,6> is replaced in D_2 by <C,5> and a correction command <R,1> standing for "replace label in entry 6" is added to D_2. Entries **2** and **4** in P'_1 (that also fall in the first six entries and are encompassed by the <C,6> command), contain the respective references **2** and **3**, exactly as the corresponding entries in P'_2 and, accordingly, no correction command is required.

The rest of the entries that correspond to <C,3> <C,1> commands do not include replaced references in P'_1 and P'_2 and, accordingly, no replacement command is required.

Step f2 simply stipulates that the inserted data and replaced data (replaced reference) will be appended to D_2 in order for the reconstructing party to be able to reconstitute P_2 from D_2 and P_1. Thus the three entries (that should be

inserted to entries 7 to 9 in P_2) and which correspond to <I,3> in D_2, are added to D_2 in section (161). Likewise the replaced reference 3 (instead of 1) in the sixth entry which corresponds to <R,1> in D_2 is added to D_2, in section (161).

Depending upon the particular application, D_2 may be stored on a storage medium, or transmitted through a communication network (212 in FIG. 2), all as required and appropriate.

There follows a description for a typical sequence of operations (220) for reconstructing P_2 from P_1 and the so received D_2, according to the present embodiment. Reconstructing P_2 may also be realized on any desired platform, all as required and appropriate. A sequence of operation, according to this embodiment includes:

a) Generate L_1 (222) from P_1 (221); (see step b above in 200)

b) Generate P'_1 (223) from P_1 and L_1; (see step e above in 200)

c) Generate P'_2 (224) from P_1' and D_2 (210) by applying the modification commands of D_2 on P'_1;

d) Analyze the so received D_2 difference result to determine the position and size of copied program fragments of P_1 and use L_1 to create AL_2 (225 in FIG. 1), which translates the label enumeration values appearing in P'_2, back to their original address reference in the following manner:

For a non-inserted referred address:

$$AL_2[L_1[\text{matching address in } P'_1]]=\text{address (first condition)}$$

For an inserted referred address:

$$AL_2[U(\,)]=\text{address (second condition)}$$

Step (d) will now be exemplified with reference to the specific example of FIG. 2. Thus, in order to generate AL_2 (170), at first L_2 (130) is reconstructed, using to this end L_1, P'_2 and D_2 all of which are available at the reconstruction side (220). Having reconstructed L_2, AL_2 can be easily derived by reversing L_2. More specifically, entry 1 in L_2 holds the value 1 and accordingly, entry 1 of AL_2 holds the value 1. Entry 5 of L_2 holds the value 5 and accordingly, entry 2 in AL_2 holds the value 5. By following the same reverse logic, entry 9 of L_2 holds the value 6 and accordingly entry 6 of AL_2 holds the value 9 and lastly by following the same logic entry 3 of AL_2 holds the value 11.

e) Lastly, P_2 (226) is reconstructed from P'_2 (224) and AL_2 (225) by translating address references in P'_2 from label enumeration values back to the original address references using AL_2.

Step (e) will now be exemplified with reference to the specific example of FIG. 2. Thus, in order to reconstruct P_2 (180), the label references in P'_2 are translated according to AL_2. More specifically, label reference 2 in entry 2 of P'_2 is replaced by actual entry reference 5 in P_2 according to the value 5 in entry #2 of AL_2. Likewise, label references 3 in entries 4 and 6 of P'_2, are replaced by actual address reference 11 in P_2, according to the value 11 in entry #3 of AL_2. In a similar manner, label references 1 in entry 12 of P'_2 is replaced by actual address reference 1 in P_2 according to the value 1 in entry #1 of AL_2. Lastly, label reference 6 in entry 7 of P'_2, is replaced by actual address reference 9 in P_2, according to the value 9 in entry #6 of AL_2. As shown by this particular example, P_2 was generated indirectly from the difference result data D_2, by the intermediary data structure, AL_2.

Entries 4 and 5 of AL_2 hold the value 0, signifying that reference labels 4 and 5 need not be updated in P_2, since they form, part of entries in P_1 that were deleted.

Those versed in the art will readily appreciate that whilst the invention has been described with reference to a specific application of updating software through a communication network, the invention is by no means bound by this specific application. Thus, by way of another example, the invention is applicable for efficient version control. Consider, for example, a series of program versions P_1 P_2 and P_3. Applying the technique of the invention gives rise to compact D_{12} and D_{23}, which stand for the difference between P_1 P_2 and P_2 P_3, respectively. The resulting compact D_{12} and D_{23} as compared to conventional, larger in size, difference results, bring about the desired efficient version control tool. Other applications are, of course, feasible, all as required and appropriate.

Having described the invention with reference to a specific application of extraction differences between old and new versions of computer programs, there follows a description with reference to a more generalized representation of data table depicted in the form of a graph.

Before turning to the specific example, few observations which apply to the data table representation are set forth below:

The graph represents a collection of data-records interconnected by references which enable to access them.

Data records are stored in a linear storage such as array according to a pre-defined order.

The amount of storage allocated for storing the references is not negligible.

The data that represents the references is not held as a contiguous block, but rather is dispersed among other items of the data table (e.g. among the data records).

Bearing this in mind, attention is directed to FIG. 3A. The graph (300) represents an (old version of an) abstract data structure in the form of a graph (which is applicable in many applications as will be explained in greater details below). The nodes A–E represent data-records and the links represent references to other data-records. In this example, the pre-defined order for the data records is an alphabetical sort of records' names. The storage of the graph would appear as:

1. A, 2, 4
2. B, 3
3. C, 4, 5
4. D
5. E, 2

Thus, for example, data-record A (301), having address 1, is linked by means of links (302 and 303) that stand for references 2 and 4 respectively, to data-records B ad D (304 and 305). As shown above, B and D reside in addresses 2 and 4 respectively.

Should it now be required to modify the above graph by adding a data-record named ABA (306 in FIG. 3B) that is linked by means of link (307) to data-record D (305), this would bring about the following new graph storage, where ABA is inserted in accordance with the said alphabetical order:

1. A, 3, 5
2. ABA, 5
3. B, 4,
4. C, 5, 6
5. D
6. E, 3

It is accordingly appreciated that insertion of only one data-record gave rise to fairly large differences between the storage of the old and the storage of the new graphs.

It is readily appreciated that the above abstract data structure is similar to a computer program, except for the fact that in the specified data structure, alphabetical order is imposed as compared to a computer program, where the order of execution is imposed.

Accordingly, applying the technique described in detail with reference to FIGS. **1** and **2** above would give rise to an extraction of compact differences.

As specified above, the data table is by no means bound to the representation of a computer program. Thus, by way of non-limiting example, the specified graph (**300**) represents a map where nodes stand for cities and links for roads linking the various cities. The technique of the invention would allow for compact representation of topographical modification in the map (e.g. in response to the construction of a new road between the City having the symbol ABA and the city having the symbol D).

By way of another non-limiting example, the specified graph represents electronic circuitry where the nodes stand for the components and the links stand for the wiring connections between the components.

Numbers, alphabetic characters and roman symbols that appear in the following claims are designated for convenience of explanations only, and do not necessarily imply the particular order of the steps.

The invention has been described with a certain degree of particularity, but those versed in the art will readily appreciate that various alterations and modifications may be carried out without departing from the spirit and scope of the following Claims:

What is claimed is:

1. A method for generating a compact difference result between an old executable program and a new executable program; each program including reference entries that contain reference that refer to other entries in the program; the method comprising the steps of:

(a) scanning the old program and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby a modified old program is generated;

(b) scanning the new program and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby a modified new program is generated;

(c) generating said difference result utilizing directly or indirectly at least said modified old program and modified new program.

2. The method of claim **1**, further comprising the step of:

(d) transmitting said compact difference result over a communication network.

3. The method of claim **2**, wherein said network includes the Internet.

4. The method of claim **1**, further comprising the step of:

(d) storing said compact difference result on a storage medium.

5. A method for performing an update in an old executable program so as to generate a new executable program; each program including reference entries that contain reference that refer to other entries in the program; the method comprising the steps of:

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old program and a modified new program;

(b) scanning the old program and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby the modified old program is generated;

(c) reconstituting the modified new program utilizing at least said compact difference result and said modified old program; said modified new program is differed from said new program at least in that substantially each reference entry in said new program is replaced in said modified new program by a distinct label mark;

(d) reconstituting said new program utilizing directly or indirectly at least said compact difference result and said modified new program.

6. The method of claim **5**, wherein said data is received in step (a) from a remote site through a network.

7. The method of claim **6**, wherein said network includes the Internet.

8. A method for generating a compact difference result between an old executable program and a new executable program; each program including reference entries that contain reference that refer to other entries in the program; the method comprising the steps of:

(a) generating a modified old program utilizing at least said old program;

(b) generating a modified new program utilizing at least said new program, said modified old program and modified new program have at least the following characteristics:

(i) substantially each reference in an entry in said old program that is different than corresponding entry in said new program due to delete/insert modifications that form part of the transition between said old program and new program are reflected as invariant references in the corresponding entries in said modified old and modified new programs;

(c) generating said compact difference result utilizing at least said modified new program and modified old program.

9. The method of claim **8**, further comprising the step of:

(d) transmitting said compact difference result over a communication network.

10. The method of claim **9**, wherein said network includes the Internet.

11. The method of claim **8**, further comprising the step of:

(d) storing said compact difference result on a storage medium.

12. A method for performing an update in an old executable program so as to generate a new executable program; each program including reference entries that contain reference that refer to other entries in the program; the method comprising the steps of:

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old program and a modified new program;

(b) generating a modified old program utilizing at least said old program;

(c) reconstituting a modified new program utilizing directly or indirectly at least said modified old program and said compact difference result; said modified old program and modified new program have at least the following characteristics:

(i) substantially each reference in an entry in said old program that is different than corresponding entry in

said new program due to delete/inset modifications that form part of the transition between said old program and new program are reflected as invariant references in the corresponding entries in said modified old and modified new programs;

(d) reconstituting said new program utilizing directly or indirectly at least said compact difference result and said modified new program.

13. The method of claim **5**, wherein said data is received in step (a) from a storage medium.

14. A system for generating a compact difference result between an old executable program and a new executable program; each program including reference entries that contain reference that refer to other entries in the program; the system comprising a processing device capable of:

(a) scanning the old program and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby a modified old program is generated;

(b) scanning the new program and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby a modified new program is generated;

(c) generating said difference result utilizing directly or indirectly at least said modified old program and modified new program.

15. The system of claim **14**, wherein said processor device is further capable of transmitting said compact difference result over a communication network.

16. The system of claim **15**, wherein said network includes the Internet.

17. The system of claim **14**, wherein said processor device is further capable of storing said compact difference result on a storage medium.

18. A system for performing an update in an old executable program so as to generate a new executable program; each program including reference entries that contain reference that refer to other entries in the program; the system comprising a processing device capable of:

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old program and a modified new program;

(b) scanning the old program and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby the modified old program is generated;

(c) reconstituting the modified new program utilizing at least said compact difference result and said modified old program; said modified new program is differed from said new program at least in that substantially each reference entry in said new program is replaced in said modified new program by a distinct label mark;

(d) reconstituting said new program utilizing directly or indirectly at least said compact difference result and said modified new program.

19. The system of claim **18**, wherein said data is received from a remote site through a network.

20. The system of claim **19**, wherein said network includes the Internet.

21. A system for generating a compact difference result between an old executable program and a new executable program; each program including reference entries that contain reference that refer to other entries in the program; the system comprising a processing device capable of:

(a) generating a modified old program utilizing at least said old program;

(b) generating a modified new program utilizing at least said new program, said modified old program and modified new program have at least the following characteristics:

(i) substantially each reference in an entry in said old program that is different than corresponding entry in said new program due to delete/insert modifications that form part of the transition between said old program and new program are reflected as invariant references in the corresponding entries in said modified old and modified new programs;

(c) generating said compact difference result utilizing at least said modified new program and modified old program.

22. The system of claim **21**, wherein said processor is further capable of transmitting said compact difference result over a communication network.

23. The system of claim **22**, wherein said network includes the Internet.

24. The system of claim **21**, wherein said processor is further capable of storing said compact difference result on a storage medium.

25. A system for performing an update in an old executable program so as to generate a new executable program; each program including reference entries that contain reference that refer to other entries in the program; the system comprising a processing device capable of:

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old program and a modified new program;

(b) generating a modified old program utilizing at least said old program;

(c) reconstituting a modified new program utilizing directly or indirectly at least said modified old program and said compact difference result; said modified old program and modified new programs have at least the following characteristics:

(i) substantially each reference in an entry in said old program that is different than corresponding entry in said new program due to delete/inset modifications that form part of the transition between said old program and new program are reflected as invariant references in the corresponding entries in said modified old and modified new programs;

(d) reconstituting said new program utilizing directly or indirectly at least said compact difference result and said modified new program.

26. The system of claim **18**, wherein said data is received in step (a) from a storage medium.

27. A processing device having associated therewith a storage medium which holds compact difference result data that was generated by the method of anyone of claims **1** to **4**.

28. A processing device having associated therewith a storage medium which holds compact difference result data that was generated by the method of anyone of claims **8** to **11**.

29. The method of claim **12**, wherein said data is received in step (a) from a remote site through a network.

30. The method of claim **29**, wherein said network includes the Internet.

31. The method of claim 12, wherein said data is received in step (a) from a storage medium.

32. The system of claim 25, wherein said data is received in step (a) from a remote site through a network.

33. The system of claim 32, wherein said network includes the Internet.

34. The system of claim 25, wherein said data is received in step (a) from a storage medium.

35. A method for generating a compact difference result between an old data table and a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the method comprising the steps of:

(a) scanning the old data table and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby a modified old data table is generated;

(b) scanning the new data table and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby a modified new data table is generated;

(c) generating said difference result utilizing directly or indirectly at least said modified old data table and modified new data table.

36. The method of claim 35, further comprising the step of:

(d) transmitting said compact difference result over a communication network.

37. The method of claim 36, wherein said network includes the Internet.

38. The method of claim 35, further comprising the step of:

(d) storing said compact difference result on a storage medium.

39. A method for performing an update in an old data table so as to generate a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the method comprising the steps of:

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old data table and a modified new data table;

(b) scanning the old data table and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby the modified old data table is generated;

(c) reconstituting the modified new data table utilizing at least said compact difference result and said modified old data table; said modified new data table is differed from said new data table at least in that substantially each reference entry in said new data table is replaced in said modified new data table by a distinct label mark;

(d) reconstituting said new data table utilizing directly or indirectly at least said compact difference result and said modified new data table.

40. The method of claim 39, wherein said data is received in step (a) from a remote site through a network.

41. The method of claim 40, wherein said network includes the Internet.

42. A method for generating a compact difference result between an old data table and a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the method comprising the steps of:

(a) generating a modified old data table utilizing at least said old data table; (b) generating a modified new data table utilizing at least said new data table, said modified old data table and modified new data table have at least the following characteristics:

(i) substantially each reference in an entry in said old data table that is different than corresponding entry in said new data table due to delete/insert modifications that form part of the transition between said old data table and new data table are reflected as invariant references in the corresponding entries in said modified old and modified new data tables;

(c) generating said compact difference result utilizing at least said modified new data table and modified old data table.

43. The method of claim 42, further comprising the step of:

(d) transmitting said compact difference result over a communication network.

44. The method of claim 43, wherein said network includes the Internet.

45. The method of claim 42, further comprising the step of:

(d) storing said compact difference result on a storage medium.

46. A method for performing an update in an old data table so as to generate a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the method comprising the steps of:

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old data table and a modified new data table;

(b) generating a modified old data table utilizing at least said old data table;

(c) reconstituting a modified new data table utilizing directly or indirectly at least said modified old data table and said compact difference result; said modified old data table and modified new data table have at least the following characteristics:

(i) substantially each reference in an entry in said old data table that is different than corresponding entry in said new data table due to delete/inset modifications that form part of the transition between said old data table and new data table are reflected as invariant references in the corresponding entries in said modified old and modified new data tables;

(d) reconstituting said new data table utilizing directly or indirectly at least said compact difference result and said modified new data table.

47. The method of claim 39, wherein said data is received in step (a) from a storage medium.

48. A system for generating a compact difference result between an old data table and a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the system comprising a processing device capable of:

(a) scanning the old data table and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby a modified old data table is generated;

(b) scanning the new data table and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby a modified new data table is generated;

(c) generating said difference result utilizing directly or indirectly at least said modified old data table and modified new data table.

49. The system of claim **48**, wherein said processor device is further capable of transmitting said compact difference result over a communication network.

50. The system of claim **49**, wherein said network includes the Internet.

51. The system of claim **48**, wherein said processor device is further capable of storing said compact difference result on a storage medium.

52. A system for performing an update in an old data table so as to generate a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the system comprising a processing device capable of:

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old data table and a modified new data table;

(b) scanning the old data table and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby the modified old data table is generated;

(c) reconstituting the modified new data table utilizing at least said compact difference result and said modified old data table; said modified new data table is differed from said new data table at least in that substantially each reference entry in said new data table is replaced in said modified new data table by a distinct label mark;

(d) reconstituting said new data table utilizing directly or indirectly at least said compact difference result and said modified new data table.

53. The system of claim **52**, wherein said data is received from a remote site through a network.

54. The system of claim **53**, wherein said network includes the Internet.

55. A system for generating a compact difference result between an old data table and a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the system comprising a processing device capable of:

(a) generating a modified old data table utilizing at least said old data table;

(b) generating a modified new data table utilizing at least said new data table, said modified old data table and modified new data table have at least the following characteristics:

(i) substantially each reference in an entry in said old data table that is different than corresponding entry in said new data table due to delete/insert modifications that form part of the transition between said old data table and new data table are reflected as invariant references in the corresponding entries in said modified old and modified new data tables;

(c) generating said compact difference result utilizing at least said modified new data table and modified old data table.

56. The system of claim **55**, wherein said processor is further capable of transmitting said compact difference result over a communication network.

57. The system of claim **56**, wherein said network includes the Internet.

58. The system of claim **55**, wherein said processor is further capable of storing said compact difference result on a storage medium.

59. A system for performing an update in an old data table so as to generate a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the system comprising a processing device capable of:

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old data table and a modified new data table;

(b) generating a modified old data table utilizing at least said old data table;

(c) reconstituting a modified new data table utilizing directly or indirectly at least said modified old data table and said compact difference result; said modified old data table and modified new data table have at least the following characteristics:

(i) substantially each reference in an entry in said old data table that is different than corresponding entry in said new data table due to delete/inset modifications that form part of the transition between said old data table and new data table are reflected as invariant references in the corresponding entries in said modified old and modified new data tables;

(d) reconstituting said new data table utilizing directly or indirectly at least said compact difference result and said modified new data table.

60. The system of claim **59**, wherein said data is received in step (a) from a storage medium.

61. A processing device having associated therewith a storage medium which holds compact difference result data that was generated by the method of anyone of claims **35** to **38**.

62. A processing device having associated therewith a storage medium which holds compact difference result data that was generated by the method of anyone of claims **42** to **45**.

63. The method of claim **46**, wherein said data is received in step (a) from a remote site through a network.

64. The method of claim **63**, wherein said network includes the Internet.

65. The method of claim **46**, wherein said data is received in step (a) from a storage medium.

66. The system of claim **59**, wherein said data is received in step (a) from a remote site through a network.

67. The system of claim **66**, wherein said network includes the Internet.

68. The system of claim **59**, wherein said data is received in step (a) from a storage medium.

* * * * *

EXHIBIT 2

Smaller is Faster (and Safer Too)

Wednesday, July 15, 2009

We have just started using a new compression algorithm called Courgette to make Google Chrome updates small.

We have built Google Chrome to address multiple factors that affect browser security. One of the pillars of our approach is to keep the software up to date, so we push out updates to Google Chrome fairly regularly. On the stable channel these are mainly security bug fixes, but the updates are more adventurous and numerous on developer channel.

It is an anathema to us to push out a whole new 10MB update to give you a ten line security fix. We want smaller updates because it narrows the window of vulnerability. If the update is a tenth of the size, we can push ten times as many per unit of bandwidth. We have enough users that this means more users will be protected earlier. A secondary benefit is that a smaller update will work better for users who don't have great connectivity.

Rather then push put a whole new 10MB update, we send out a diff that takes the previous version of Google Chrome and generates the new version. We tried several binary diff algorithms and have been using bsdiff up until now. We are big fans of bsdiff - it is small and worked better than anything else we tried.

But bsdiff was still producing diffs that were bigger than we felt were necessary. So we wrote a new diff algorithm that knows more about the kind of data we are pushing - large files containing

compiled executables. Here are the sizes for the recent 190.1->190.4 update on the developer channel:

- **Full update**: 10,385,920 bytes
- **bsdiff update**: 704,512 bytes
- **Courgette update**: 78,848 bytes

The small size in combination with Google Chrome's silent update means we can update as often as necessary to keep users safe.

More information on how Courgette works can be found here.

Posted by Stephen Adams, Software Engineer

EXHIBIT 3

Software Updates: Courgette

How Courgette works

As I described in *Smaller is faster (and safer too)*, we wrote a new differential compression algorithm for making Google Chrome updates significantly smaller.

We want smaller updates because it *narrows the window of vulnerability*. If the update is a tenth of the size, we can push ten times as many per unit of bandwidth. We have enough users that this means more users will be protected earlier. A secondary benefit is that a smaller update will work better for users who don't have great connectivity.

Rather than push put a whole new 10MB update, we send out a diff that takes the previous version of Google Chrome and generates the new version. We tried several binary diff algorithms and have been using bsdiff up until now. We are big fans of bsdiff - it is small and worked better than anything else we tried.

But bsdiff was still producing diffs that were bigger than we felt were necessary. So we wrote a new diff algorithm that knows more about the kind of data we are pushing - large files containing compiled executables. Here are the sizes in bytes for the recent 190.1->190.4 update on the developer channel:

Full update	10,385,920
bsdiff update	704,512
Courgette update	78,848

The small size in combination with Google Chrome's silent update means we can update as often as necessary to keep users safe.

Compiled code

The problem with compiled applications is that even a small source code change causes a disproportional number of byte level changes. When you add a few lines of code, for example, a range check to prevent a buffer overrun, all the subsequent code gets moved to make room for the new instructions. The compiled code is full of internal references where some instruction or datum contains the address (or offset) of another instruction or datum. It only takes a few source changes before almost all of these internal pointers have a different value, and there are a lot of them - roughly half a million in a program the size of chrome.dll.

The source code does not have this problem because all the entities in the source are *symbolic*. Functions don't get committed to a specific address until very late in the compilation process, during assembly or linking. If we could step backwards a little and make the internal pointers symbolic again, could we get smaller updates?

Courgette uses a primitive disassembler to find the internal pointers. The disassembler splits the program into three parts: a list of the internal pointer's target addresses, all the other bytes, and an 'instruction' sequence that determines how the plain bytes and the pointers need to be interleaved and adjusted to get back the original input. We call this an 'assembly language' because we can run an 'assembler' to process the instructions and emit a sequence of bytes to recover the original file.

The non-pointer part is about 80% of the size of the original program, and because it does not have any pointers mixed in, it tends to be well behaved, having a diff size that is in line with the changes in the source code. Simply converting the program into the

assembly language form makes the diff produced by bsdiff about 30% smaller.

We bring the pointers under control by introducing 'labels' for the addresses. The addresses are stored in an array and the list of pointers is replaced by a list of array indexes. The array is a primitive 'symbol table', where the names of the symbols, or 'labels' are the integer indexes into the array. What we get from the symbol table is a degree of freedom in how we express the program. We can move the addresses around in the array provided we make the corresponding changes to the list of indexes.

How do we use this to generate a better diff? With bsdiff we would compute the new file, 'update' from the 'original' like this:

```
server:
   diff = bsdiff(original, update)
   transmit diff

client:
   receive diff
   update = bspatch(original, diff)
```

(The server would pre-compute diff so that it could be transmitted immediately)

Courgette transforms the program into the primitive assembly language and does the diffing at the assembly level:

```
server:
   asm_old = disassemble(original)
   asm_new = disassemble(update)
   asm_new_adjusted = adjust(asm_new, asm_old)
   asm_diff = bsdiff(asm_old, asm_new_adjusted)
   transmit asm_diff

client:
   receive asm_diff
   asm_old = disassemble(original)
   asm_new_adjusted = bspatch(asm_old, asm_diff)
   update = assemble(asm_new_adjusted)
```

The special sauce is the adjust step. Courgette moves the addresses within the asm_new symbol table to minimize the size of asm_diff. Addresses in the two symbol tables are matched on their statistical properties which ensures the index lists have many long common substrings. The matching does not use any heuristics based on the surrounding code or debugging information to align the addresses.

More than one executable, less than an executable

For the above to work, 'assemble' and 'disassemble' have to be strict inverses, and 'original' and 'update' have to be single well-formed executable files. It is much more useful if 'original' and 'update' can contain several executables as well as a lot of non-compiled files like JavaScript and PNG images. For Google Chrome, the 'original' and 'update' are an archive file containing all the files needed to install and run the browser.

We can think of a differential update as a prediction followed by a correction, a kind of guessing game. In its simplest form (just bsdiff / bspatch), the client has only a dumb guess, 'original', so the server sends a binary diff to correct 'original' to the desired answer, 'update'. Now what if the server could pass a hint that could be used to generate a better guess, but we are not sure the guess will be useful? We could insure against losing information by using the original and the guess together as the basis for the diff:

```
server:
   hint = make_hint(original, update)
   guess = make_guess(original, hint)
```

```
    diff = bsdiff(concat(original, guess), update)
    transmit hint, diff

client
    receive hint, diff
    guess = make_guess(original, hint)
    update = bspatch(concat(original, guess), diff)
```

This system has some interesting properties. If the guess is the empty string, then we have the same diff as with plain bsdiff. If the guess is perfect, the diff will be tiny, simply a directive to copy the guess.

Between the extremes, the guess could be a perfect subset of 'update'. Then bsdiff will construct a diff that mostly takes material from the perfect prediction and the original to construct the update. This is how Courgette deals with inputs like tar files containing both executable files and other files. The hint is the location of the embedded executables together with the asm_diff for each one.

Once we have this prediction / correction scheme in place we can use it to reduce the amount of work that the client needs to do. Executables often have large regions that do not contain internal pointers, like the resource section which usually contains string tables and various visual elements like icons and bitmaps. The disassembler generates an assembly language program which pretty much says 'here is a big chunk of constant data', where the data is identical to the original file. bsdiff then generates a diff for the constant data. We can get substantially the same effect by omitting the pointer-free regions from the disassembly and letting the final diff do the work.

Summary

Courgette transforms the input into an alternate form where binary diffing is more effective, does the differential compression in the transformed space, and inverts the transform to get the patched output in the original format. With careful choice of the alternate format we can get substantially smaller updates.

We are writing a more detailed paper on Courgette and will post an update when it is ready.

EXHIBIT 4

Navigation

Home

Getting Involved

For Developers

For Testers

Contact

Sitemap

Get the Code

The Chromium codebase consists of hundreds of thousands of files, which means that a checkout straight from the Subversion (SVN) repository can take a long time. To speed up the process, we have provided a tarball that you can use to bootstrap the download. Alternatively, you can skip the tarball and download straight from SVN (not recommended).

Note: There is no advantage to checking out straight from SVN. The tarball includes SVN directories so that after you unpack the tarball, you can get up to the latest revision by using `gclient sync`.

If you only want to look at the source code on your own machine, you'll need at least 1.6 GB of hard drive space available. (Somewhat less for Linux, since it already has some of the dependencies installed.) If you want to build it, you will need just under 10 GB of space, including all the object files and executables.

Contents

Bootstrap using the tarball

1. Make sure that you have a program that can untar `.tgz` (`.tar.gz`) files.
   - Mac OS X and Linux both have `tar` built in. (On a Mac, use a shell in the Terminal application.)
   - Examples for Windows include the open-source 7-Zip archiver, the free BsdTar utility (part of LibArchive), and WinZip. **Note:** Cygwin's tar tool will not work; it will mess up the file permissions.
2. Download the source tarball. Some download applications change the file suffix without extracting the file. If yours did, rename it back to `chromium.rXXXXX.tgz`.
3. Choose a directory to hold your source code. **Important:** Make sure the directory path has **no spaces**.
   - Windows example: `c:\chromiumtrunk`
   - Mac OS X (Terminal) or Linux example: `~/chromium`
4. Untar the source tarball into the directory you've chosen. Examples: If you're using 7-Zip, extract the .tgz file, then extract the resulting .tar file. If you're using LibArchive, issue the following command:
   `"C:\Program Files\GnuWin32\bin\bsdtar.exe" -xzf chromium.tgz`
5. Install the depot_tools.
6. Updating your checkout once by running `gclient sync --force` in the source code directory.
   1. If you don't want to sync, you need to generate the project files with `gclient runhooks --force`. This will call GYP to generate your platform-specific files. You won't be able to build otherwise. If you don't sync, you'll miss some os-specific dependencies so you're better to sync anyway. :)

This should give you a complete source tree. But if you have any problems, first check that you've installed any prerequisites listed on the build instructions for your platform.

Check out directly from SVN

You'll use the gclient depot tool to download the Chromium code from its SVN repository. The first time you execute `gclient`, there will be a delay (a minute or so) while it updates the depot tools. Downloading the Chromium code takes about an hour.

The `gclient config` step only needs to be run once to set up your working directory. It creates a `.gclient` file in your working directory that identifies the corresponding structure to pull from the repository. The `gclient sync` step creates several subdirectories. To update your tree in the future, you only need to run `gclient sync` from anywhere within your working directory.

NOTE: These instructions will pull a read-only tree. If you are a committer, and plan to make changes to source code, use the instructions given to you when you received commit access.

Windows

> **Note:** It's not necessary to have Subversion or Python installed already: the first run of gclient will install them for you. On the other hand, if you *do* already have Subversion installed through Cygwin, you'll need to set up your PATH to have the depot_tools svn ahead of the cygwin svn, and **use only the depot_tools svn, not Cygwin's, to check out Chromium**.

1. Create a directory to hold your source code. This example assumes `c:\chromiumtrunk`, but other names are fine.
   Important: Make sure the full directory path has **no spaces**.
2. In a shell window, execute the following commands:
   ```
   cd c:\chromiumtrunk
   gclient config http://src.chromium.org/svn/trunk/src
   ```
   To download the initial code, update your checkout as described below.

Mac OS X

1. Create a directory to hold the code. This example assumes the directory is `~/chromium`, but other names are fine.
2. From a shell in the Terminal, execute the following commands:
   ```
   $ cd ~/chromium
   $ gclient config http://src.chromium.org/svn/trunk/src
   ```
3. To download the initial code, update your checkout as described below.

Linux

1. Pick a directory for your build. We will call this directory `$CHROMIUM_ROOT` below.
2. Check out Chromium:
   ```
   $ cd $CHROMIUM_ROOT
   $ gclient config http://src.chromium.org/svn/trunk/src
   ```
3. To download the initial code, update your checkout as described below.

Staying Green most of the time

When running gclient config, you can specify a second URL to be referenced when doing updates. Instead of pulling the most recent revision, the version number at this URL will be queried, allowing you to track the "most recent green" revision so you can spend less time debugging other people's issues or running builds only to find out that the waterfall was red. Chromium has two of these URLs:

Continuous build

- `http://build.chromium.org/buildbot/continuous/LATEST/REVISION`
  This corresponds to the most recent revision that passed both unit tests and layout tests. Since layout tests can take a while to run, this revision may be an hour or more "stale".

LKGR

- `http://chromium-status.appspot.com/lkgr`
  This URL holds the version of the latest revision to pass only unit tests (in debug mode). This can happen faster, so for most developers this is probably what you want since it will help you ensure that your changes work against a "fresher" version of Chromium.

Setup

To use one of these URLs, pass it when you run `gclient config`:

```
    $ cd ~/chromium
    $ gclient config
http://src.chromium.org/svn/trunk/src http://chromium-
status.appspot.com/lkgr
```

Now whenever you call `gclient sync`, it will only sync as far as the configured URL specifies. To over-ride this, pass the `--head` parameter to `gclient`, e.g.: `gclient sync --head`

You can also add this directly to your .gclient file if you already have one:

```
solutions = [
  { "name" : "src",
    "url"  : "http://src.chromium.org/svn/trunk/src",
    "safesync_url" : "http://chromium-status.appspot.com/lkgr"
  },
]
```

Reducing the size of your checkout

You can edit your .gclient file to avoid pulling down certain pieces of the checkout that you may not want. For example, inserting something like

```
"custom_deps" : {
  "src/third_party/WebKit/LayoutTests": None,
}
```

into one of the solutions (i.e. just underneath the "url": ... line) should save a lot of space. The list of repos that gclient pulls is stored in src/DEPS.

Update to the latest revision

Whether you started with a source tarball or an svn checkout, at some point you'll want to update your checkout to the latest revision.

The first time you execute gclient, there will be a delay (a minute or so) while it updates the depot tools. How long the Chromium code update takes depends on how much has changed since you last updated (or since the bootstrap tarball was created).

1. Install the depot_tools, if you haven't already.
2. Visit the Chromium Buildbot waterfall to see the state of the tree. If the top of the waterfall says:
   OPEN - The tree is in a good state and you should be able to compile the code. Go to the next step.
   CLOSED - There might be compile or test failures. You can download the code, but you'll get those same failures when you try to compile or run tests. Best to check back later.
3. In a shell window, execute the following commands:
   `cd` *`[your Chromium source directory]`*
   `gclient sync`

To update to a specific revision, use

```
gclient sync --revision src@####
```

and the DEPS file will make sure you get the other directories in their matching forms.

EXHIBIT 5

View of /trunk/src/courgette/win32_x86_patcher.h

Parent Directory | Revision Log

Revision 15692 - (download) (annotate)
Fri May 8 23:00:29 2009 UTC (6 months, 1 week ago) by *sra@chromium.org*
File size: 2677 byte(s)

```
Move Courgette
   from src\third_party\courgette
   to   src\courgette and src\courgette\third_party

Fixed #includes

Added properties to ignore generated files:

C:\c5\src>svn pg svn:ignore courgette
courgette.xcodeproj
courgette.sln
courgette_fuzz.vcproj
courgette_lib.vcproj
courgette_minimal_tool.vcproj
courgette_tool.vcproj
courgette.vcproj
courgette_unittests.vcproj
SConstruct
courgette_fuzz.scons
courgette_lib.scons
courgette_main.scons
courgette_minimal_tool.scons
courgette.scons
courgette_tool.scons
courgette_unittests.scons


Review URL: http://codereview.chromium.org/115062

// Copyright (c) 2009 The Chromium Authors. All rights reserved.
// Use of this source code is governed by a BSD-style license that can be
// found in the LICENSE file.

// This is the transformation for Windows X86 executables.

#ifndef COURGETTE_WIN32_X86_PATCHER_H_
#define COURGETTE_WIN32_X86_PATCHER_H_

#include "courgette/ensemble.h"

namespace courgette {

// CourgetteWin32X86Patcher is a TransformationPatcher for Windows 32-bit
// executables.
//
class CourgetteWin32X86Patcher : public TransformationPatcher {
 public:
  explicit CourgetteWin32X86Patcher(const Region& region)
      : ensemble_region_(region) {
  }
```

```
  Status Init(SourceStream* parameter_stream) {
    if (!parameter_stream->ReadVarint32(&base_offset_))
      return C_BAD_TRANSFORM;
    if (!parameter_stream->ReadVarint32(&base_length_))
      return C_BAD_TRANSFORM;

    if (base_offset_ > ensemble_region_.length())
      return C_BAD_TRANSFORM;
    if (base_length_ > ensemble_region_.length() - base_offset_)
      return C_BAD_TRANSFORM;

    return C_OK;
  }

  Status PredictTransformParameters(SinkStreamSet* predicted_parameters) {
    // No code needed to write an 'empty' predicted parameter set.
    return C_OK;
  }

  Status Transform(SourceStreamSet* corrected_parameters,
                   SinkStreamSet* transformed_element) {
    Status status;
    if (!corrected_parameters->Empty())
      return C_GENERAL_ERROR;   // Don't expect any corrected parameters.

    AssemblyProgram* program = NULL;
    status = ParseWin32X86PE(ensemble_region_.start() + base_offset_,
                             base_length_,
                             &program);
    if (status != C_OK)
      return status;

    EncodedProgram* encoded = NULL;
    status = Encode(program, &encoded);
    DeleteAssemblyProgram(program);
    if (status != C_OK)
      return status;

    status = WriteEncodedProgram(encoded, transformed_element);
    DeleteEncodedProgram(encoded);
    if (status != C_OK)
      return status;

    return status;
  }

  Status Reform(SourceStreamSet* transformed_element,
                SinkStream* reformed_element) {
    Status status;
    EncodedProgram* encoded_program = NULL;
    status = ReadEncodedProgram(transformed_element, &encoded_program);
    if (status != C_OK)
      return status;

    status = Assemble(encoded_program, reformed_element);
    DeleteEncodedProgram(encoded_program);
    if (status != C_OK)
      return status;

    return C_OK;
  }

 private:
  Region ensemble_region_;

  uint32 base_offset_;
  uint32 base_length_;
```

```
  DISALLOW_COPY_AND_ASSIGN(CourgetteWin32X86Patcher);
};

}  // namespace
#endif  // COURGETTE_WIN32_X86_PATCHER_H_
```

EXHIBIT 6

EXHIBIT 7

A Note on Google Apps for Android

Posted by Dan Morrill on 25 September 2009 at 2:31 PM

Lately we've been busy bees in Mountain View, as you can see from the recent release of Android 1.6 to the open-source tree, not to mention some devices we're working on with partners that we think you'll really like. Of course, the community isn't sitting around either, and we've been seeing some really cool and impressive things, such as the custom Android builds that are popular with many enthusiasts. Recently there's been some discussion about an exchange we had with the developer of one of those builds, and I've noticed some confusion around what is and isn't part of Android's open source code. I want to take a few moments to clear up some of those misconceptions, and explain how Google's apps for Android fit in.

Everyone knows that mobile is a big deal, but for a long time it was hard to be a mobile app developer. Competing interests and the slow pace of platform innovation made it hard to create innovative apps. For our part, Google offers a lot of services — such as Google Search, Google Maps, and so on — and we found delivering those services to users' phones to be a very frustrating experience. But we also found that we weren't alone, so we formed the Open Handset Alliance, a group of like-minded partners, and created Android to be the platform that we all wished we had. To encourage broad adoption, we arranged for Android to be open-source. Google also created and operates Android Market as a service for developers to distribute their apps to Android users. In other words, we created Android because the industry needed an injection of openness. Today, we're thrilled to see all the enthusiasm that developers, users, and others in the mobile industry have shown toward Android.

With a high-quality open platform in hand, we then returned to our goal of making our services available on users' phones. That's why we developed Android apps for many of our services like YouTube, Gmail, Google Voice, and so on. These apps are Google's way of benefiting from Android in the same way that any other developer can, but the apps are not part of the Android platform itself. We make some of these apps available to users of any Android-powered device via Android Market, and others are pre-installed on some phones through business deals. Either way, these apps aren't open source, and that's why they aren't included in the Android source code repository. Unauthorized distribution of this software harms us just like it would any other business, even if it's done with the best of intentions.

I hope that clears up some of the confusion around Google's apps for Android. We always love seeing novel uses of Android, including custom Android builds from developers who see a need. I look forward to seeing what comes next!

TRACKBACKS

Google's Response to the Cyanogen C&D Letter
The Android/Cyanogen Dispute Takes Android in New Directions
Google Cracks Down on Android Developer, Offers Olive Branch
Google slaps Android developer with cease-and-desist letter
El affair Google, Android y Cyanogen
Google being evil with developers
Android Developers Blog: A Note on Google Apps for Android
CyanogenMod 継続に署名
Just a Reminder: MS Has Been Pretty Nice To Us
Just a Reminder: MS Has Been Pretty Nice To Us
Android - The official line on modding
Android Open Source Model Has a Short Circuit
Android Open Source Model Has a Short Circuit
Descendo do pedestal

Google takes back what they never gave to Android
Lessons from Android: Unintended Consequences (or How to Kneecap ...
Google ostatecznie blokuje rozwój CyanogenMod
Google's "statement" about Cyanogen
CyanogenMod: Na nátlak Googlu skončí současné ROM
Cyanogen: Na nátlak Googlu skončil, webové stránky nejedou
Google Responds To Cyanogen Uproar
Save CyanogenMod 앱 안드로이드 마켓 1위!!

Create a Link

Newer Post Home Older Post

EXHIBIT 8

Red Bend Software Corporate Overview

Red Bend Software
Bend Rules. Break Boundaries.

Red Bend Software, the leader in Mobile Software Management (MSM), provides software solutions for managing firmware, applications and devices over the air. The company's award-winning MSM products enable device manufacturers, mobile operators and software developers to increase revenues, reduce support costs and achieve faster time to market by remotely managing their software assets on mobile devices.

Red Bend's software has been deployed in more than half a billion mobile devices and adopted by eight of the top 10 handset manufacturers, including Kyocera, LG Electronics, Motorola, Sharp, Sony Ericsson and ZTE, as well as dozens of other leading companies in the mobile, M2M and WiMAX markets. Unlike device management vendors with proprietary end-to-end systems and manufacturers' internally developed solutions that are platform specific, Red Bend is the only company offering independent client software that is interoperable with any standards-based server and that works with any platform on any type of mobile device.

Founded in 1999, Red Bend is a privately held, venture capital-financed company with about 100 employees. Red Bend's corporate headquarters are near Boston, Massachusetts, with offices in China, Israel, Japan, Korea and the U.K.

Target Markets

Red Bend's mobile software management solutions deliver significant advantages to device manufacturers, network operators, platform providers and application developers. The company's client software is designed for mobile phones, mobile broadband PC cards and modems, WiMAX mobile devices, machine-to-machine (M2M) modules and other wireless equipment to:

- Enhance revenues through consumers' broader adoption of new services and features
- Improve time-to-market for new mobile devices, applications and features
- Lower warranty and service costs by eliminating costly device recalls and reducing service calls
- Increase customer satisfaction and retention by resolving problems more quickly and providing consumers with a continuous stream of enhancements
- Reduce risks of deploying new technology

Mobile Software Management Solutions

vRapid Mobile® FOTA for Firmware Updates

Red Bend's vRapid Mobile FOTA (firmware over the air) enables device manufacturers to create firmware updates and to deploy and install the updated firmware over the air to deployed devices. vRapid Mobile FOTA has been adopted by the world's leading manufacturers of mobile phones and wireless devices, and is proven in millions of accurate and secure firmware updates. Red Bend is the FOTA market share leader, with vCurrent Mobile embedded in 64% of FOTA-enabled mobile phones, according to Ovum.

- Identifies the essential changes from an existing version of firmware to a new, updated version
- Reduces the new updated firmware to a compact package, up to 97% smaller than can be achieved through other techniques, to allow for efficient over-the-air delivery
- Installs the new firmware within the limited memory available on a wireless device
- Manages different versions of updated firmware and their deployment on a variety of devices

vRapid Mobile® SCOTA for Software Component Management

Red Bend's vRapid Mobile SCOTA (software components over the air) offers unmatched capabilities to customize the growing software assets of mobile devices. vRapid Mobile SCOTA is the first comprehensive solution to provide complete flexibility and control over updating, adding, changing and removing individual embedded software components over the air.

- Operators can instantly deliver new revenue-generating services to their entire subscriber base, without being constrained by the software capabilities of existing handsets or waiting for new handsets to become available
- Device manufacturers gain substantial flexibility and efficiency in their software maintenance processes, and can provide consumers with the ability to personalize the core functions of their phones to match their lifestyles