US 20030115552A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2003/0115552 A1

Jahnke et al. (43) Pub. Date: Jun. 19, 2003

(54) METHOD AND SYSTEM FOR AUTOMATIC CREATION OF MULTILINGUAL IMMUTABLE IMAGE FILES

(76) Inventors: **Jorg Jahnke**, Hamburg (DE); **Dietmar Cordes**, Horneburg (DE); **Nils Fuhrmann**, Amelinghausen (DE)

Correspondence Address:
SONNENSCHEIN NATH & ROSENTHAL
P.O. BOX 061080
WACKER DRIVE STATION
CHICAGO, IL 60606-1080 (US)

(21) Appl. No.: **10/303,819**

(22) Filed: **Nov. 26, 2002**

(30) **Foreign Application Priority Data**

Nov. 27, 2001 (EP) 01 128 179.7
May 10, 2002 (EP) 02 010 132.5
May 10, 2002 (EP) 02 010 133.3

Publication Classification

(51) **Int. Cl.**[7] **G06F 15/00**
(52) **U.S. Cl.** **715/536**; 715/522

(57) **ABSTRACT**

An improved translation system is provided that allows users to view screen objects of a user interface in different languages even though the screen object's visual elements are stored in an immutable image file (e.g., .gif or bitmap). To accomplish this, a method and system automatically generate a number of immutable image files, each containing text in different languages. These files are then made available to users so that they may select a preferred language in which to view the screen objects of the user interface.

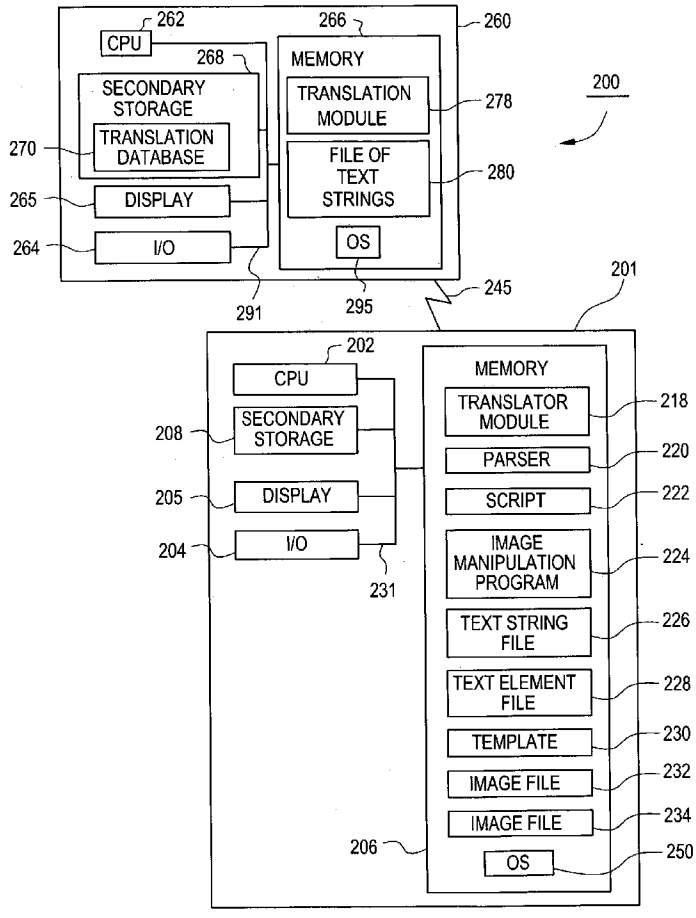


FIG. 1A

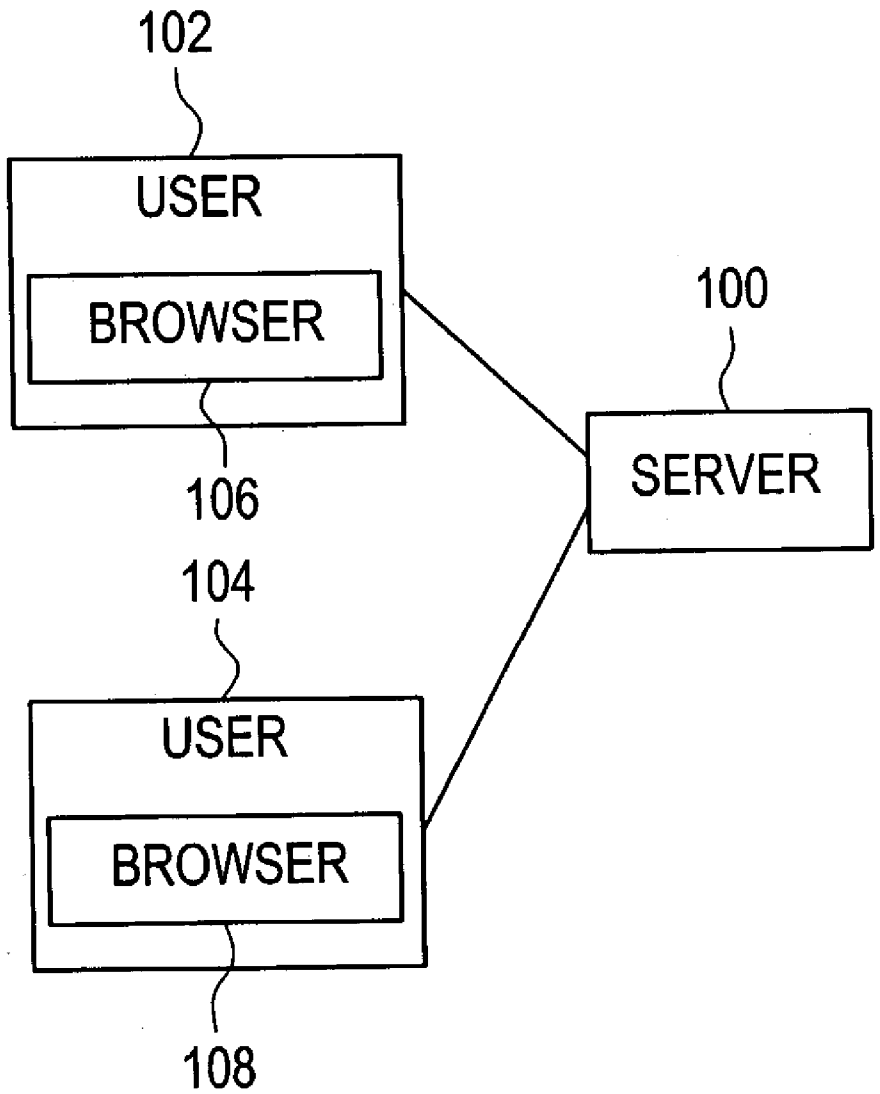


FIG. 1B

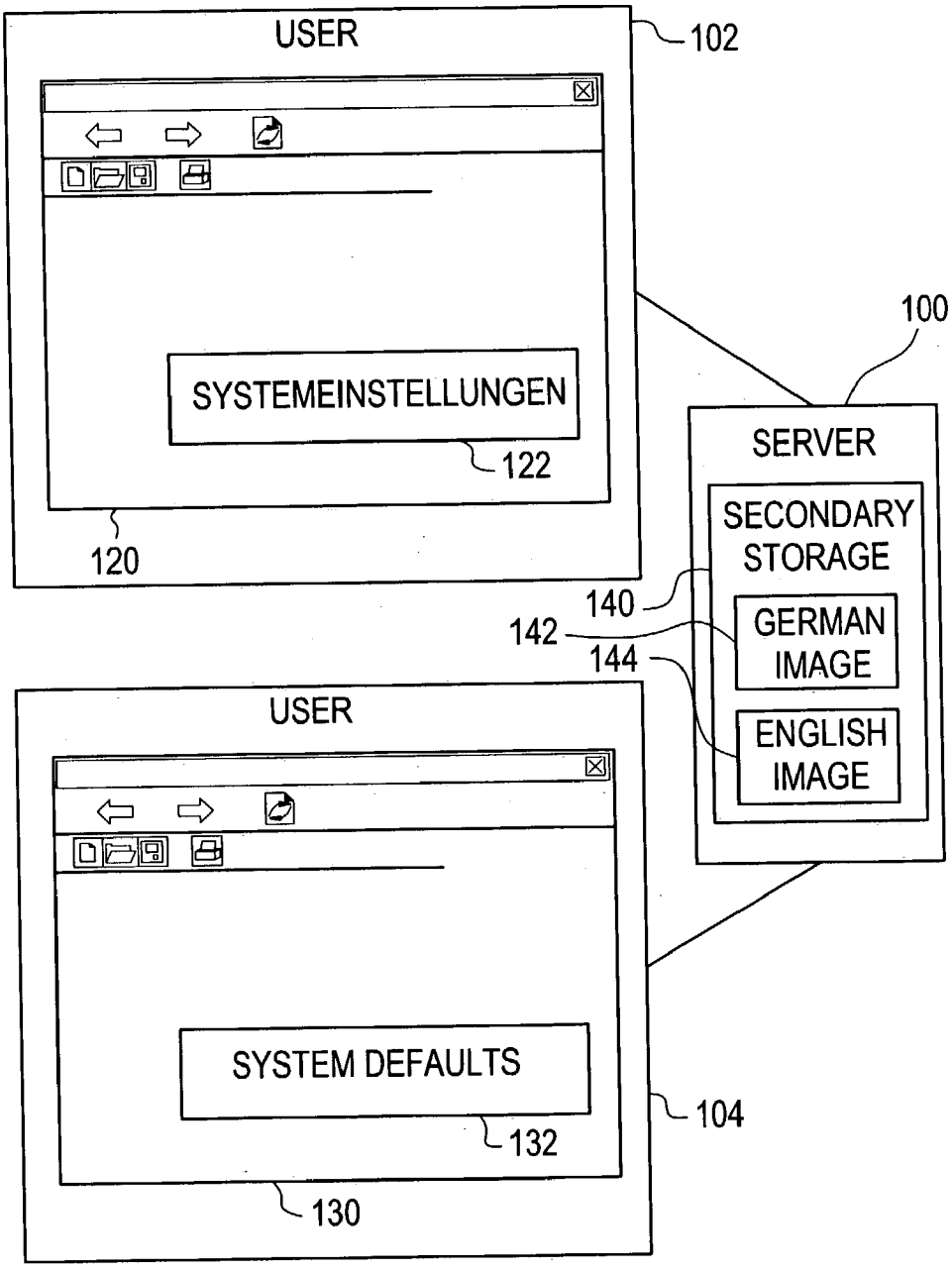

USER
102
SYSTEMEINSTELLUNGEN
122
120
100
SERVER
SECONDARY STORAGE
140
142
144
GERMAN IMAGE
ENGLISH IMAGE
USER
SYSTEM DEFAULTS
132
130
104

FIG. 2

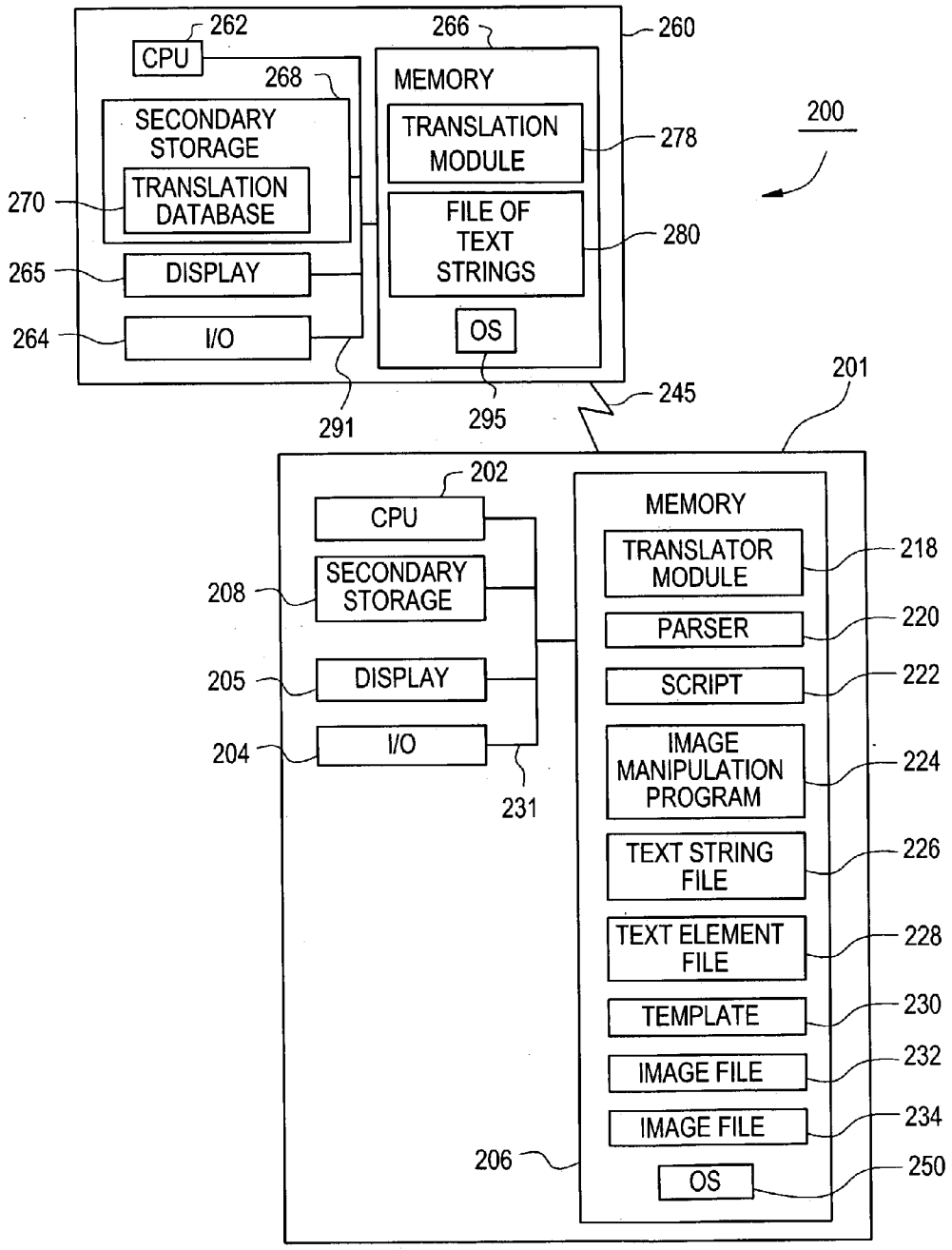

262
CPU
266
260
MEMORY
268
SECONDARY STORAGE
TRANSLATION MODULE
278
200
TRANSLATION DATABASE
270
FILE OF TEXT STRINGS
280
265
DISPLAY
264
I/O
OS
291
295
245
201
202
CPU
MEMORY
TRANSLATOR MODULE
218
208
SECONDARY STORAGE
PARSER
220
205
DISPLAY
SCRIPT
222
204
I/O
IMAGE MANIPULATION PROGRAM
224
231
TEXT STRING FILE
226
TEXT ELEMENT FILE
228
TEMPLATE
230
IMAGE FILE
232
IMAGE FILE
234
206
OS
250

FIG. 3

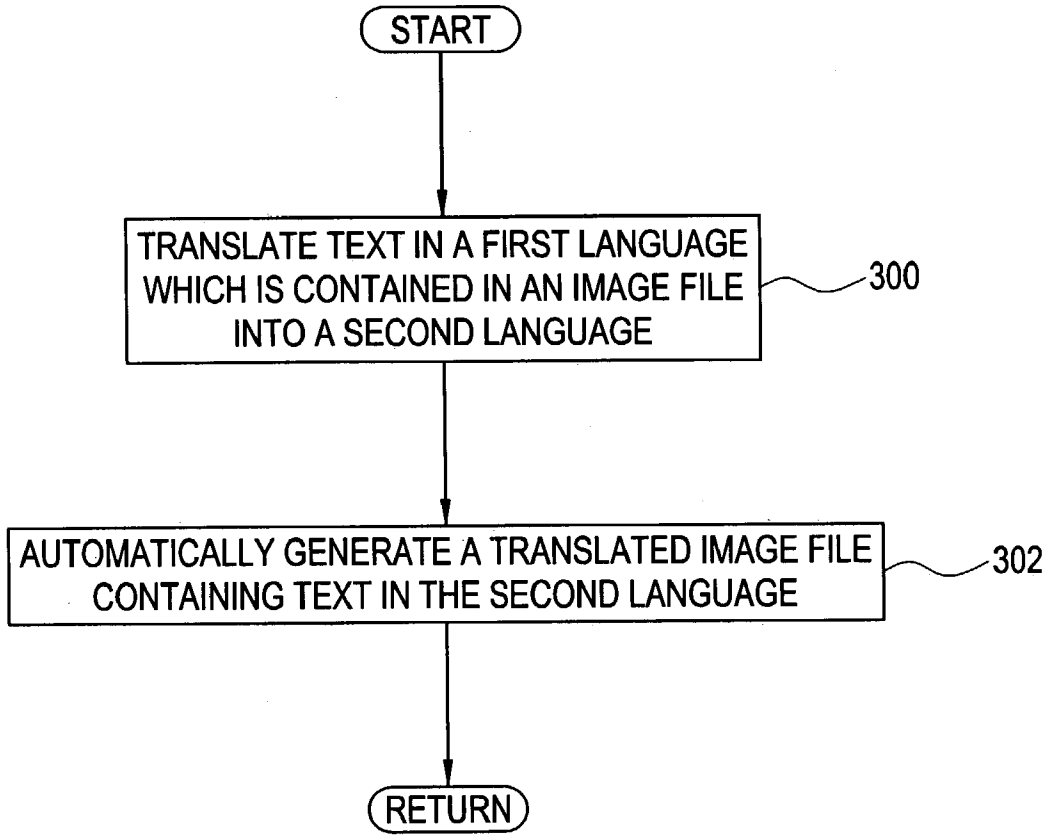

START
TRANSLATE TEXT IN A FIRST LANGUAGE WHICH IS CONTAINED IN AN IMAGE FILE INTO A SECOND LANGUAGE
300
AUTOMATICALLY GENERATE A TRANSLATED IMAGE FILE CONTAINING TEXT IN THE SECOND LANGUAGE
302
RETURN

FIG. 4

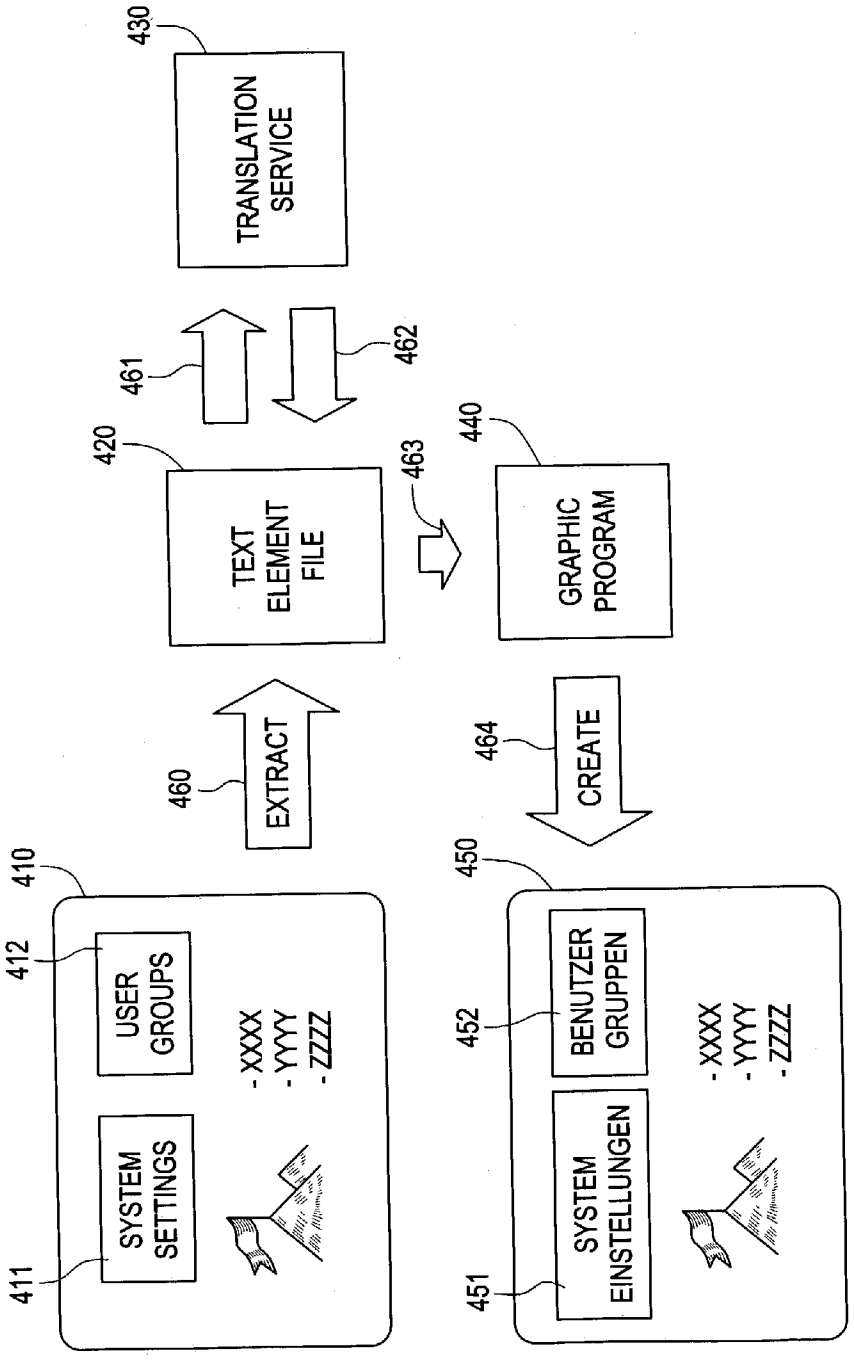

430
TRANSLATION SERVICE
461
462
420
TEXT ELEMENT FILE
463
440
GRAPHIC PROGRAM
460
EXTRACT
464
CREATE
410
412
USER GROUPS
- XXXX
- YYYY
- ZZZZ
411
SYSTEM SETTINGS
450
452
BENUTZER GRUPPEN
- XXXX
- YYYY
- ZZZZ
451
SYSTEM EINSTELLUNGEN

FIG. 5

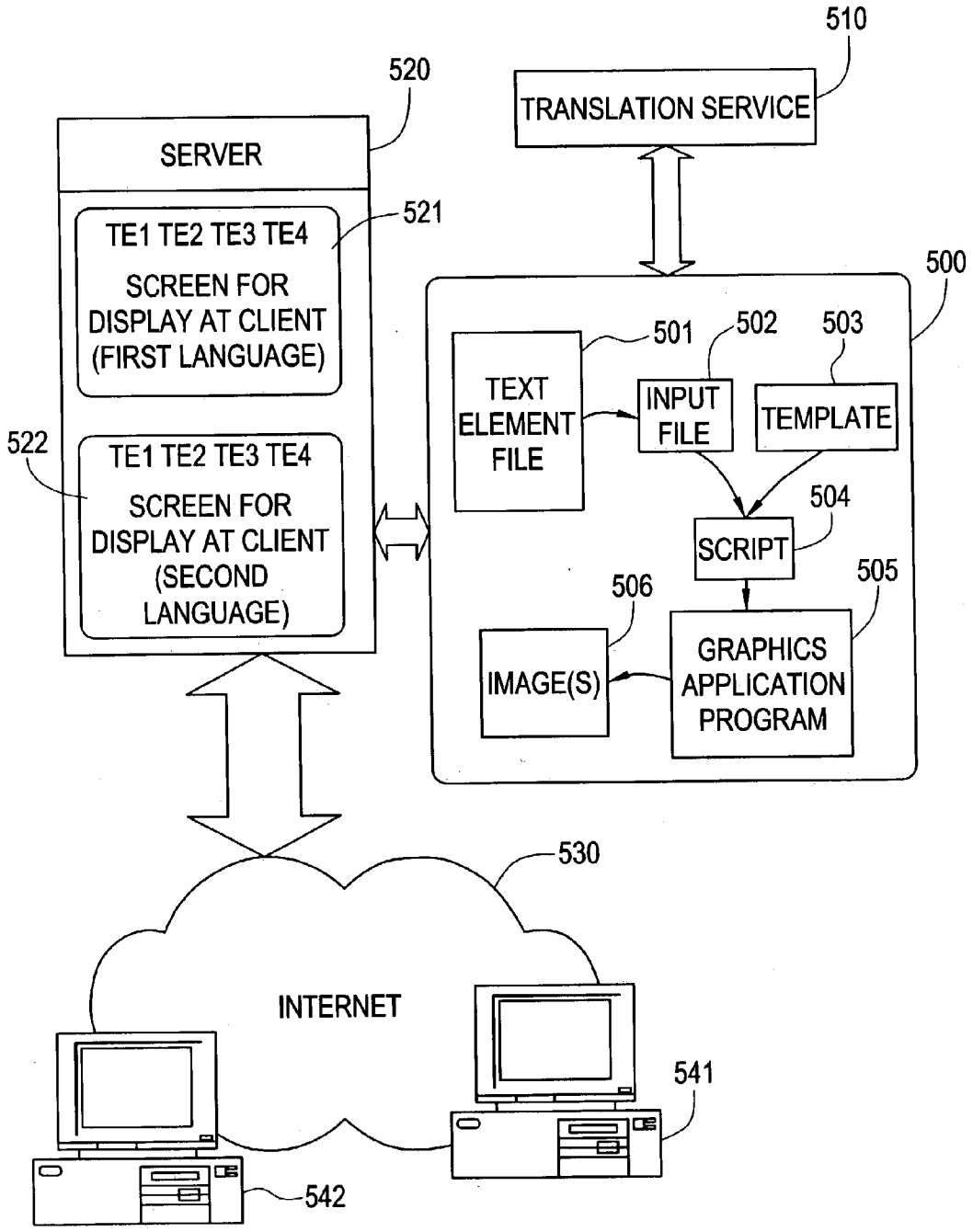

510
TRANSLATION SERVICE
520
SERVER
521
TE1 TE2 TE3 TE4
SCREEN FOR DISPLAY AT CLIENT (FIRST LANGUAGE)
522
TE1 TE2 TE3 TE4
SCREEN FOR DISPLAY AT CLIENT (SECOND LANGUAGE)
500
501
502
503
TEXT ELEMENT FILE
INPUT FILE
TEMPLATE
504
SCRIPT
505
506
IMAGE(S)
GRAPHICS APPLICATION PROGRAM
530
INTERNET
541
542

FIG. 6

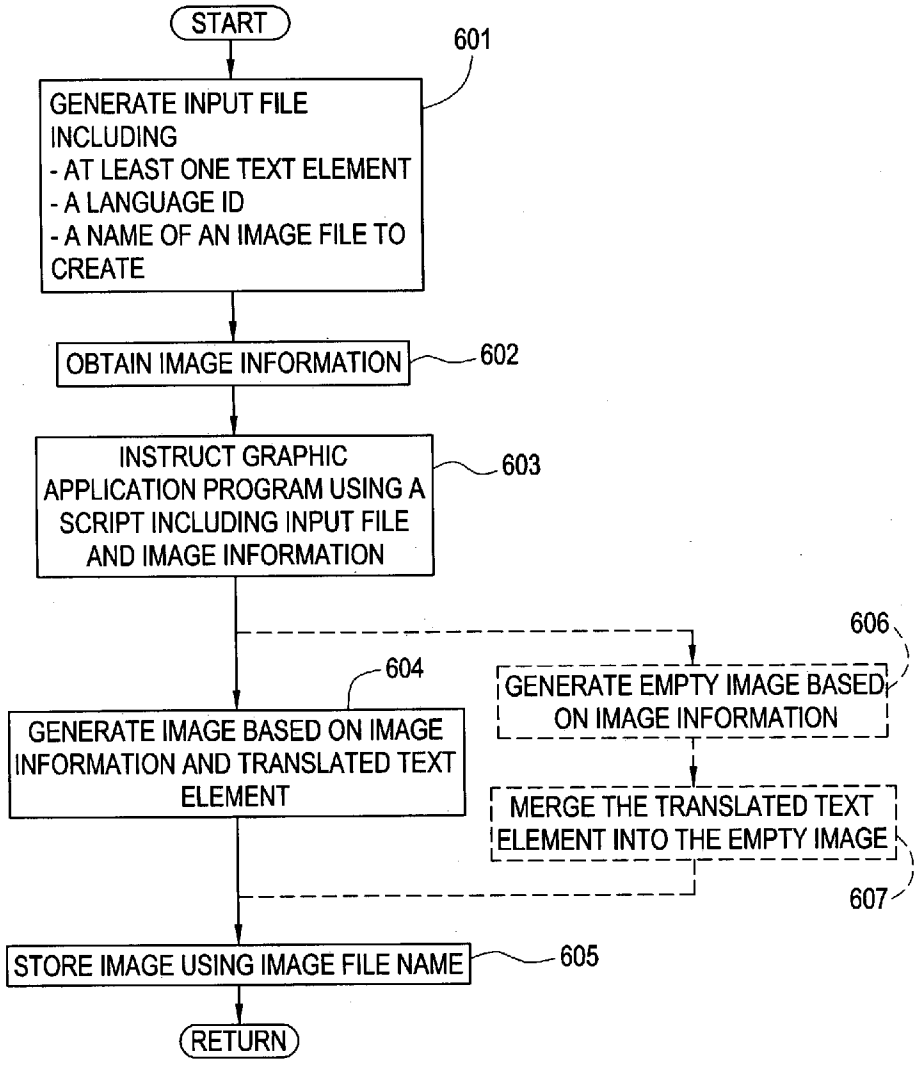

START
601
GENERATE INPUT FILE INCLUDING
- AT LEAST ONE TEXT ELEMENT
- A LANGUAGE ID
- A NAME OF AN IMAGE FILE TO CREATE
OBTAIN IMAGE INFORMATION
602
INSTRUCT GRAPHIC APPLICATION PROGRAM USING A SCRIPT INCLUDING INPUT FILE AND IMAGE INFORMATION
603
606
604
GENERATE EMPTY IMAGE BASED ON IMAGE INFORMATION
GENERATE IMAGE BASED ON IMAGE INFORMATION AND TRANSLATED TEXT ELEMENT
MERGE THE TRANSLATED TEXT ELEMENT INTO THE EMPTY IMAGE
607
STORE IMAGE USING IMAGE FILE NAME
605
RETURN

FIG. 7

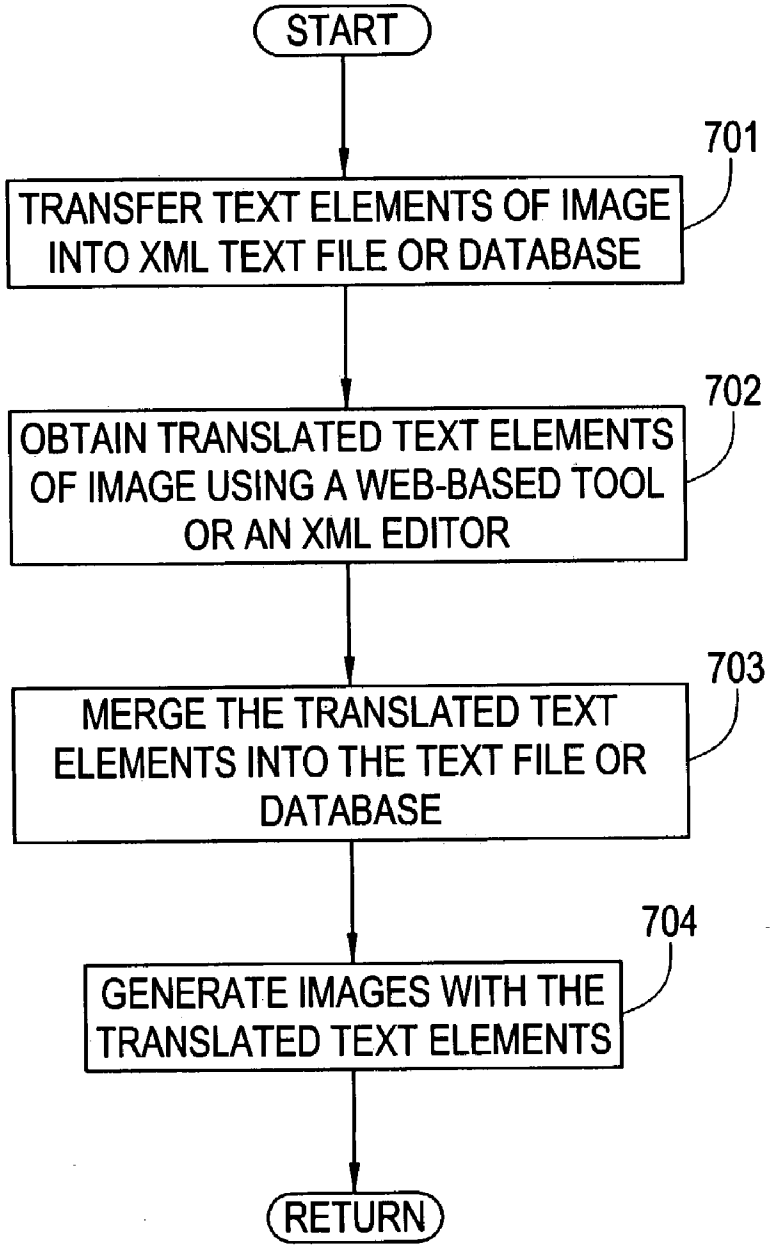

START
701
TRANSFER TEXT ELEMENTS OF IMAGE INTO XML TEXT FILE OR DATABASE
702
OBTAIN TRANSLATED TEXT ELEMENTS OF IMAGE USING A WEB-BASED TOOL OR AN XML EDITOR
703
MERGE THE TRANSLATED TEXT ELEMENTS INTO THE TEXT FILE OR DATABASE
704
GENERATE IMAGES WITH THE TRANSLATED TEXT ELEMENTS
RETURN

FIG. 8

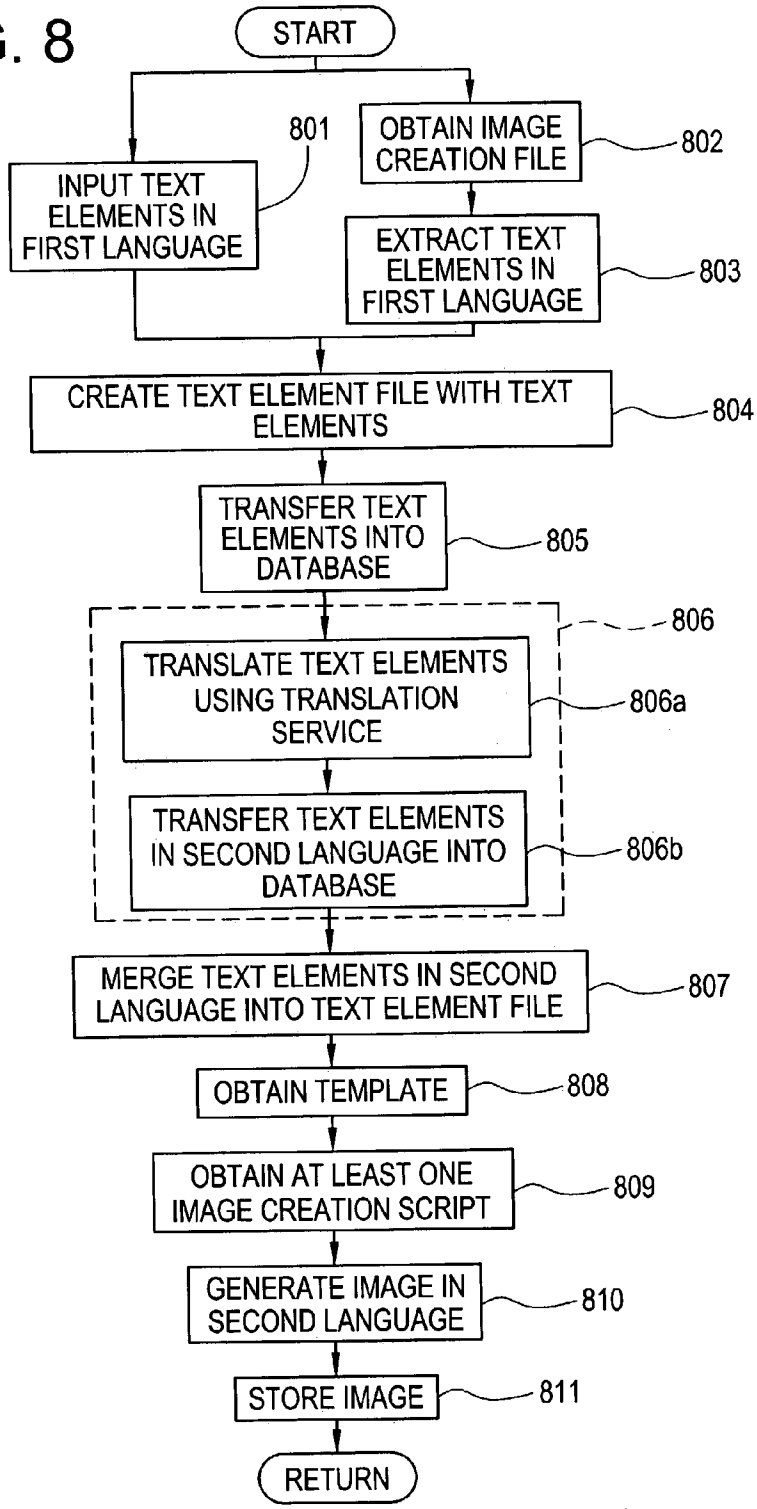

START
801
INPUT TEXT ELEMENTS IN FIRST LANGUAGE
OBTAIN IMAGE CREATION FILE
802
EXTRACT TEXT ELEMENTS IN FIRST LANGUAGE
803
CREATE TEXT ELEMENT FILE WITH TEXT ELEMENTS
804
TRANSFER TEXT ELEMENTS INTO DATABASE
805
806
TRANSLATE TEXT ELEMENTS USING TRANSLATION SERVICE
806a
TRANSFER TEXT ELEMENTS IN SECOND LANGUAGE INTO DATABASE
806b
MERGE TEXT ELEMENTS IN SECOND LANGUAGE INTO TEXT ELEMENT FILE
807
OBTAIN TEMPLATE
808
OBTAIN AT LEAST ONE IMAGE CREATION SCRIPT
809
GENERATE IMAGE IN SECOND LANGUAGE
810
STORE IMAGE
811
RETURN

FIG. 9A

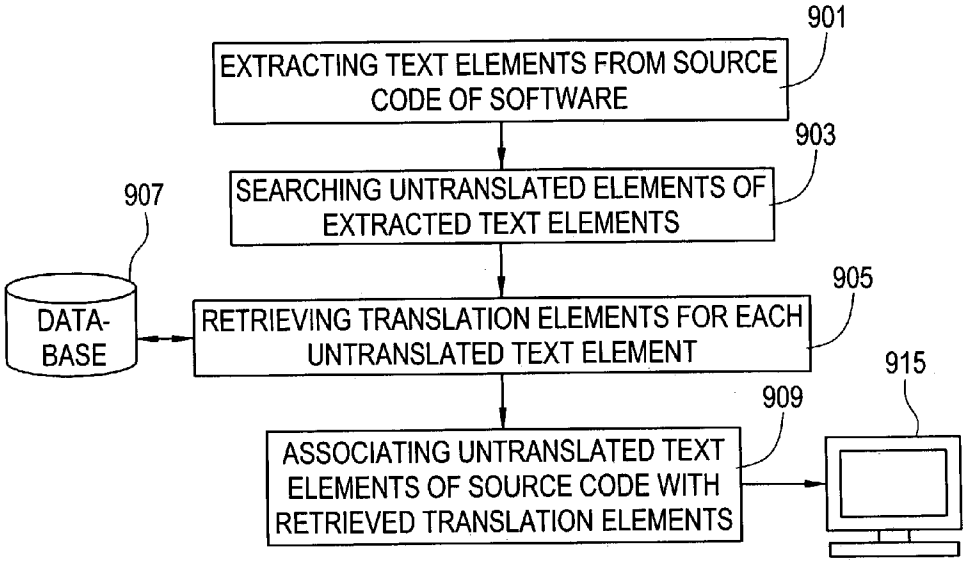

901
EXTRACTING TEXT ELEMENTS FROM SOURCE CODE OF SOFTWARE
903
SEARCHING UNTRANSLATED ELEMENTS OF EXTRACTED TEXT ELEMENTS
907
DATA-BASE
905
RETRIEVING TRANSLATION ELEMENTS FOR EACH UNTRANSLATED TEXT ELEMENT
915
909
ASSOCIATING UNTRANSLATED TEXT ELEMENTS OF SOURCE CODE WITH RETRIEVED TRANSLATION ELEMENTS

FIG. 9B

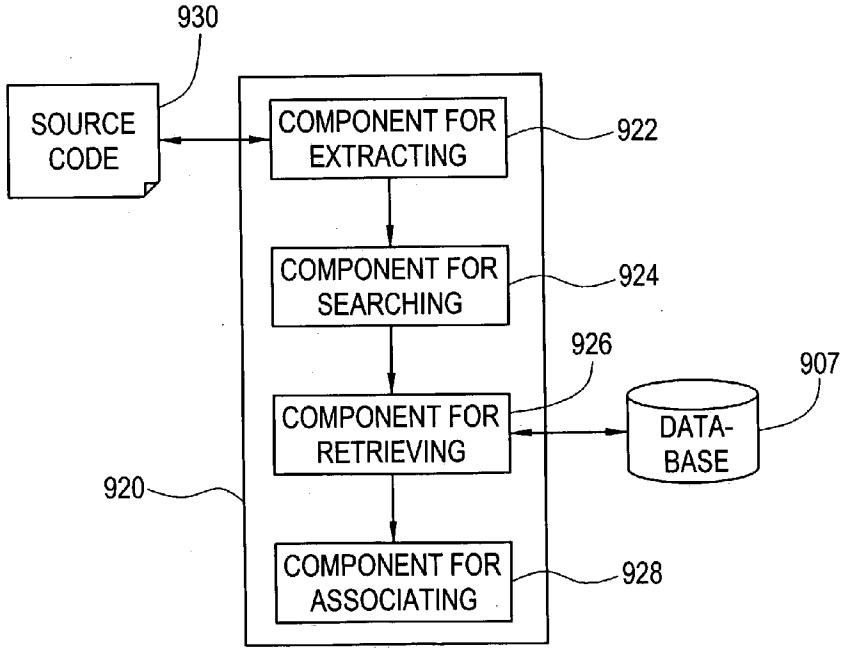

930
SOURCE CODE
COMPONENT FOR EXTRACTING
922
COMPONENT FOR SEARCHING
924
926
COMPONENT FOR RETRIEVING
907
DATA-BASE
920
COMPONENT FOR ASSOCIATING
928

FIG. 10

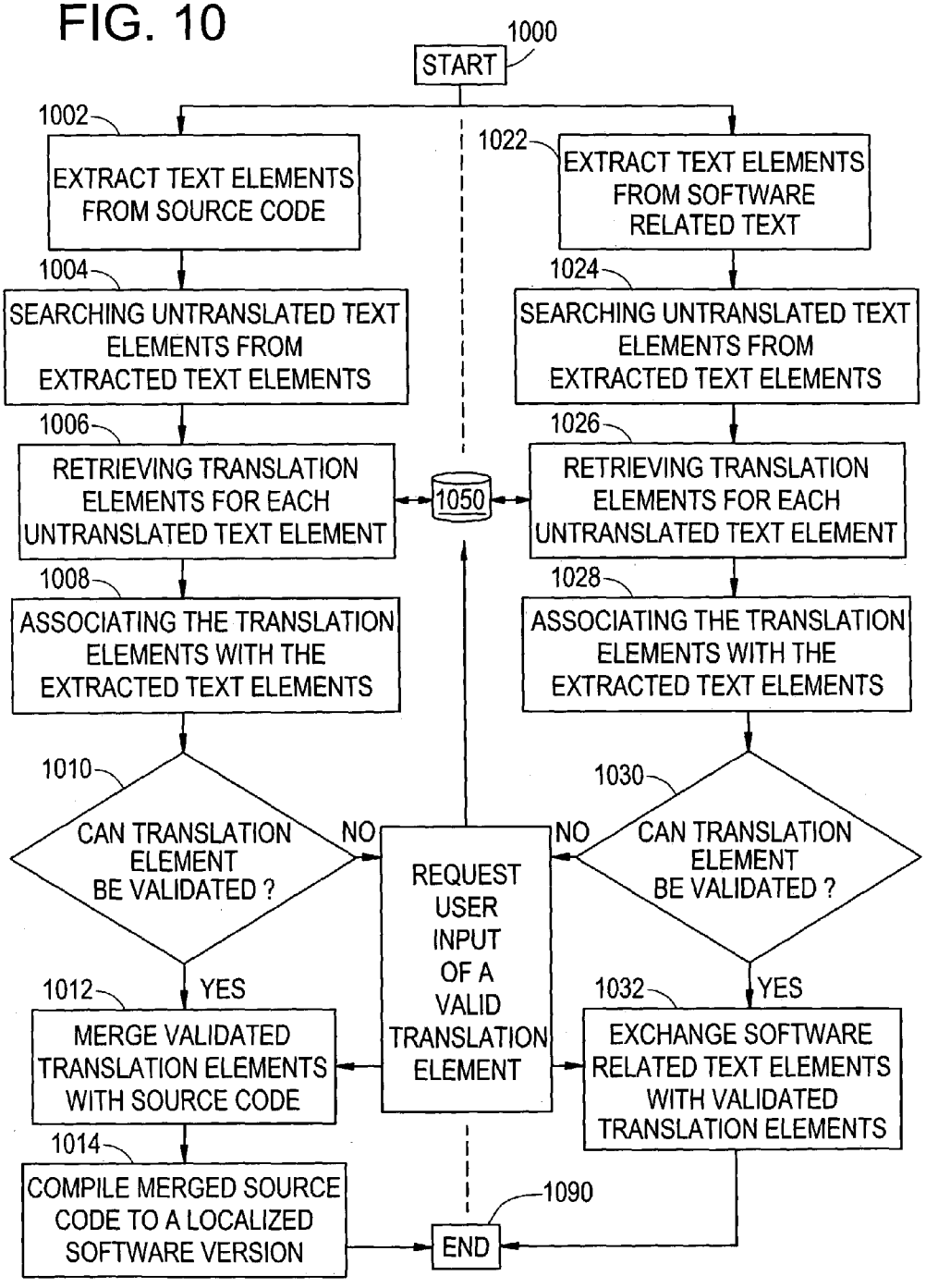

1000
START
1002
EXTRACT TEXT ELEMENTS FROM SOURCE CODE
1004
SEARCHING UNTRANSLATED TEXT ELEMENTS FROM EXTRACTED TEXT ELEMENTS
1006
RETRIEVING TRANSLATION ELEMENTS FOR EACH UNTRANSLATED TEXT ELEMENT
1008
ASSOCIATING THE TRANSLATION ELEMENTS WITH THE EXTRACTED TEXT ELEMENTS
1010
CAN TRANSLATION ELEMENT BE VALIDATED ?
NO
YES
1012
MERGE VALIDATED TRANSLATION ELEMENTS WITH SOURCE CODE
1014
COMPILE MERGED SOURCE CODE TO A LOCALIZED SOFTWARE VERSION
1022
EXTRACT TEXT ELEMENTS FROM SOFTWARE RELATED TEXT
1024
SEARCHING UNTRANSLATED TEXT ELEMENTS FROM EXTRACTED TEXT ELEMENTS
1026
RETRIEVING TRANSLATION ELEMENTS FOR EACH UNTRANSLATED TEXT ELEMENT
1028
ASSOCIATING THE TRANSLATION ELEMENTS WITH THE EXTRACTED TEXT ELEMENTS
1030
CAN TRANSLATION ELEMENT BE VALIDATED ?
NO
YES
1032
EXCHANGE SOFTWARE RELATED TEXT ELEMENTS WITH VALIDATED TRANSLATION ELEMENTS
1050
REQUEST USER INPUT OF A VALID TRANSLATION ELEMENT
1090
END

FIG. 11A

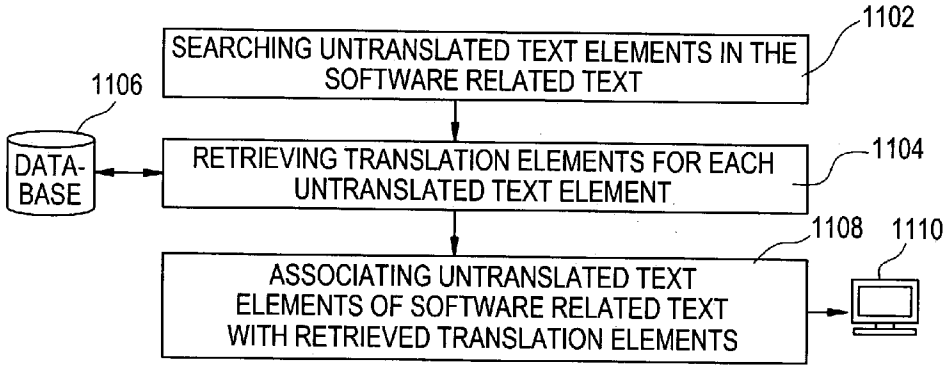


FIG. 11B

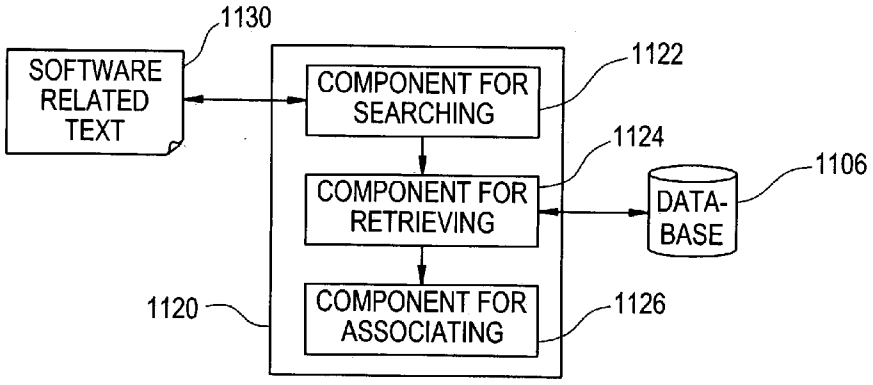


FIG. 12

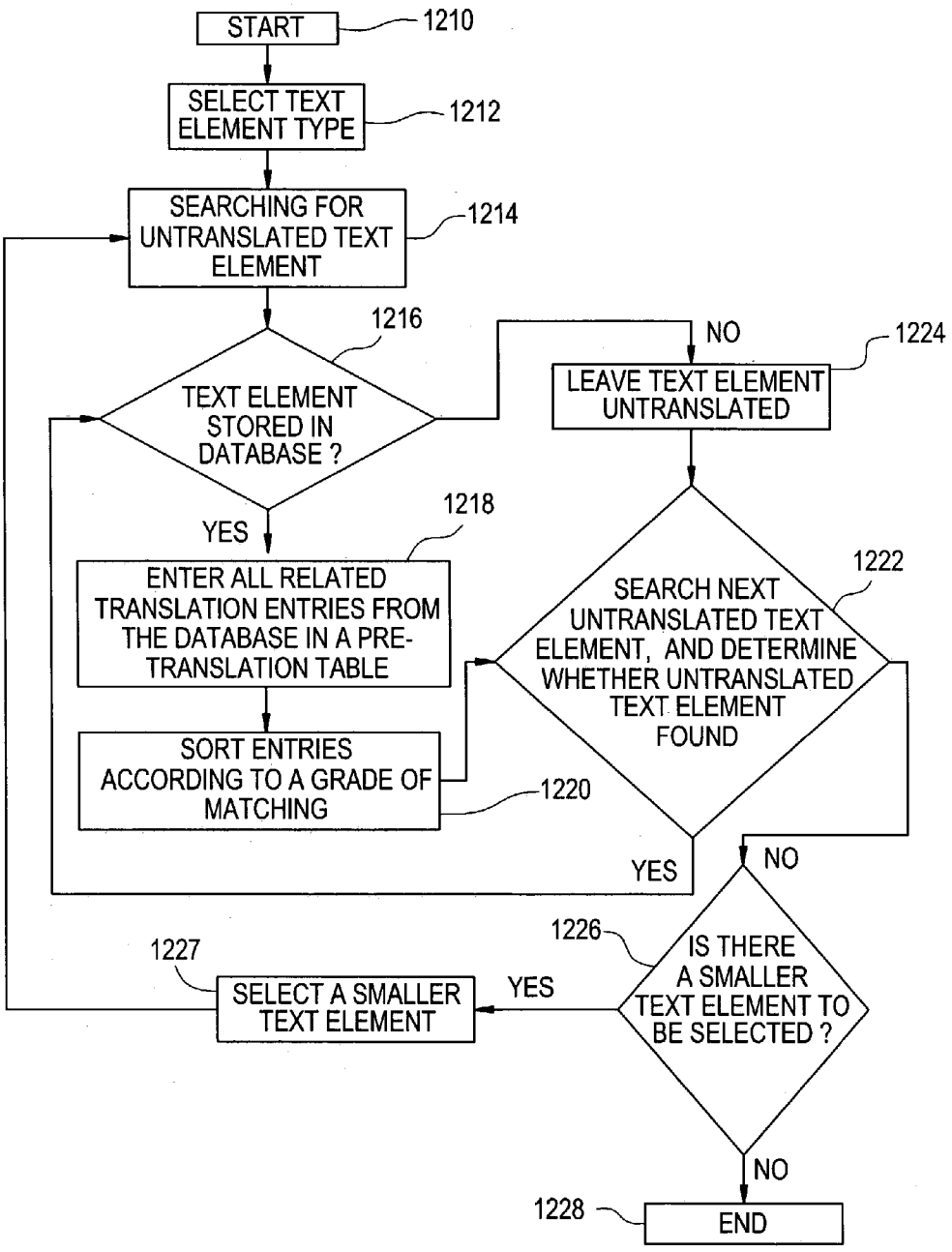

START
1210
SELECT TEXT ELEMENT TYPE
1212
SEARCHING FOR UNTRANSLATED TEXT ELEMENT
1214
1216
TEXT ELEMENT STORED IN DATABASE ?
NO
1224
LEAVE TEXT ELEMENT UNTRANSLATED
YES
1218
ENTER ALL RELATED TRANSLATION ENTRIES FROM THE DATABASE IN A PRE-TRANSLATION TABLE
SORT ENTRIES ACCORDING TO A GRADE OF MATCHING
1220
1222
SEARCH NEXT UNTRANSLATED TEXT ELEMENT, AND DETERMINE WHETHER UNTRANSLATED TEXT ELEMENT FOUND
YES
NO
1226
IS THERE A SMALLER TEXT ELEMENT TO BE SELECTED ?
YES
1227
SELECT A SMALLER TEXT ELEMENT
NO
1228
END

FIG. 13

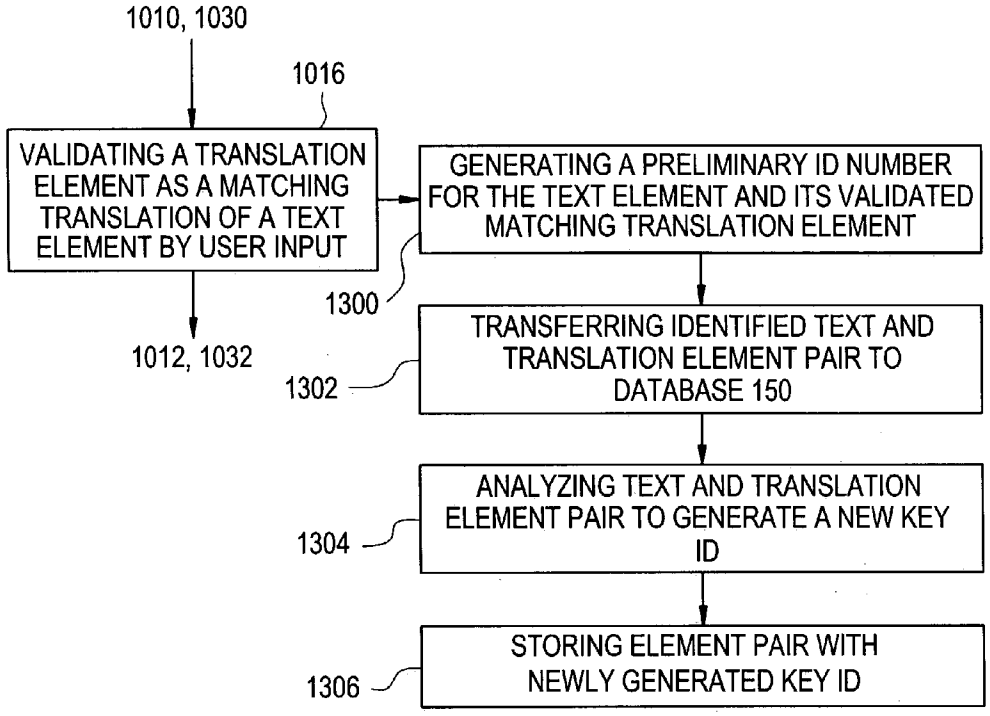


FIG. 14

KEY-ID	LANGUAGE	TEXT	SYNONYM
1	1	TABLE	DESK
1	49	TISCH	SCHREIBTISCH
2	1	TABLE	CHART
2	49	TABELLE	VERZEICHNIS
3	1	TABLE	BOARD
3	49	TAFEL	BRETT
4	1	TABLE	PLAIN (GEOL.)
4	49	PLATEAU	EBENE
5	1	TABLE	(OPT.)
5	49	BILDEBENE	(OPT.)

FIG. 15A

1540

KEY-ID	LANGUAGE	TEXT	STATUS / QUALITY
1	1	THE COMPUTER IS STANDING ON THE TABLE	UNTRANSLATED
1	1	THE COMPUTER IS STANDING ON THE TABLE	100%
1	1	THE COMPUTER IS STANDING ON THE TABLE [?]	97%
1	1	[A] COMPUTER IS STANDING ON THE TABLE.	95%
1	1	[A] COMPUTER IS STANDING ON [a] TABLE.	84%

SELECTING THE BOLD LINED TABLE ELEMENT BY USER INPUT.

1542

KEY-ID	LANGUAGE	TEXT	STATUS
1	1	THE COMPUTER IS STANDING ON THE TABLE	UNTRANSLATED
1	1	THE COMPUTER IS STANDING ON THE TABLE	100%
1	49	DER COMPUTER STEHT AUF DEM TISCH.	100%

SELECTING THE BOLD LINED TABLE ELEMENT BY USER INPUT.

1544

KEY-ID	LANGUAGE	TEXT	STATUS
1	49	DER COMPUTER STEHT AUF DEM TISCH.	FULLY TRANSLATED

FIG. 15B

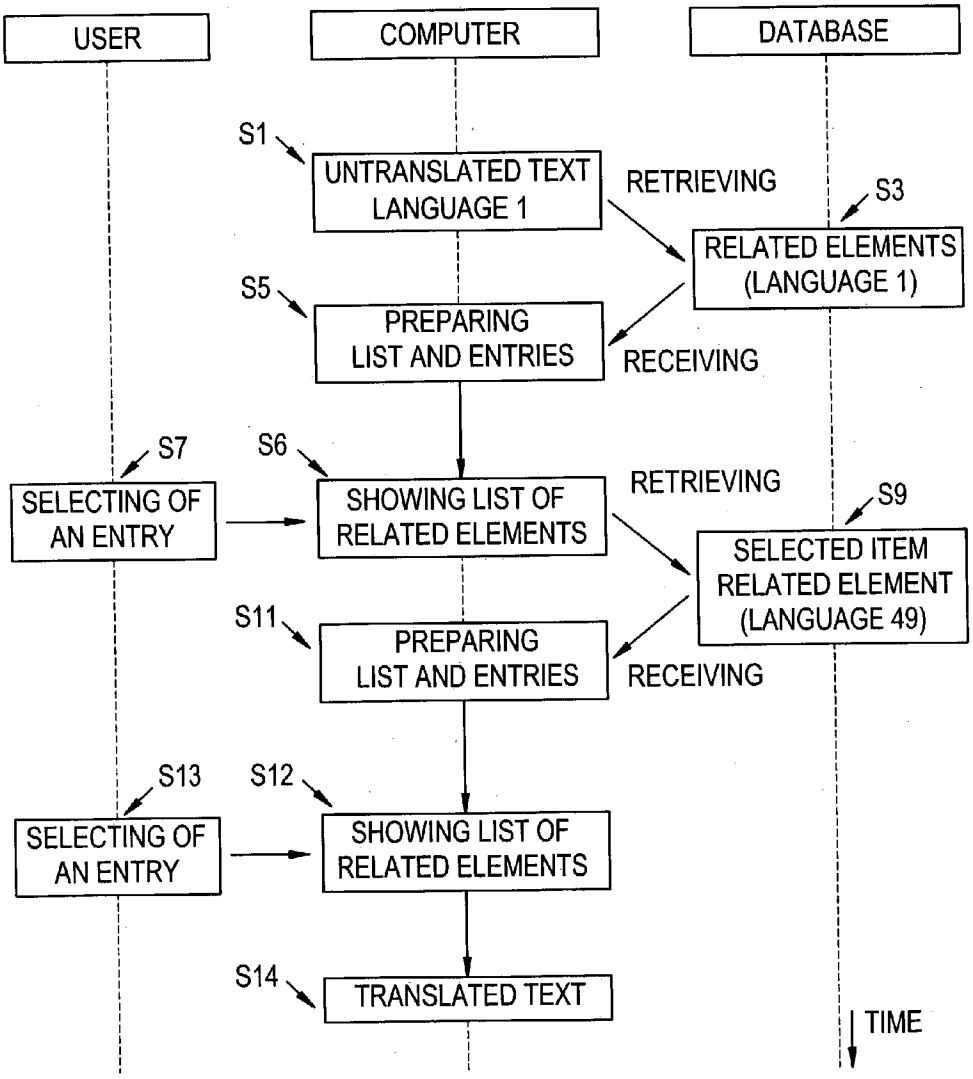

USER
COMPUTER
DATABASE
S1
UNTRANSLATED TEXT LANGUAGE 1
RETRIEVING
S3
RELATED ELEMENTS (LANGUAGE 1)
S5
PREPARING LIST AND ENTRIES
RECEIVING
S7
S6
SELECTING OF AN ENTRY
SHOWING LIST OF RELATED ELEMENTS
RETRIEVING
S9
SELECTED ITEM RELATED ELEMENT (LANGUAGE 49)
S11
PREPARING LIST AND ENTRIES
RECEIVING
S13
S12
SELECTING OF AN ENTRY
SHOWING LIST OF RELATED ELEMENTS
S14
TRANSLATED TEXT
TIME

FIG. 16A

1650

KEY-ID	LANGUAGE	TEXT	STATUS / QUALITY
1	1	THE COMPUTER IS STANDING ON THE TABLE	UNTRANSLATED
1	1	*A* *KEYBOARD* IS STANDING ON THE TABLE	66%
1	1	*THERE* *IS* *A* *KEYBOARD* STANDING ON THE TABLE ?	61%

SELECTING THE BOLD LINED TABLE ELEMENT BY USER INPUT.

1652

KEY-ID	LANGUAGE	TEXT	STATUS
1	1	THE COMPUTER IS STANDING ON THE TABLE	UNTRANSLATED
1	1	*A* *KEYBOARD* IS STANDING ON THE TABLE	66%
1	49	*EINE* *TASTATUR* STEHT AUF DEM TISCH	66%

EDITING THE SUGGESTED TRANSLATION BY USER INPUT.

1654

KEY-ID	LANGUAGE	TEXT	STATUS
1	-	*THE* *COMPUTER* STEHT AUF DEM TISCH	PARTIALLY TRANSLATED 66%

SELECTING THE DOTTED BOLD LINED TABLE ELEMENT FOR NEXT SMALLER TEXT ELEMENTS BY USER INPUT.

1656

KEY-ID	LANGUAGE	TEXT	STATUS / QUALITY
127	-	*THE* *COMPUTER* STEHT AUF DEM TISCH	PARTIALLY TRANSLATED 66%
127	1	THE COMPUTER	100%
127	49	DER COMPUTER	100%

ACCEPTING OR EDITING THE TRANSLATION SUGGESTION BY USER INPUT

1658

KEY-ID	LANGUAGE	TEXT	STATUS
1	49	DER COMPUTER STEHT AUF DEM TISCH	FULLY TRANSLATED

TRANSFERRING THE NEW TRANSLATIONS TO THE DATABASE

1660

KEY-ID	LANGUAGE	TEXT	STATUS
114115	49	DER COMPUTER STEHT AUF DEM TISCH	
114115	1	THE COMPUTER IS STANDING ON THE TABLE	

FIG. 16B

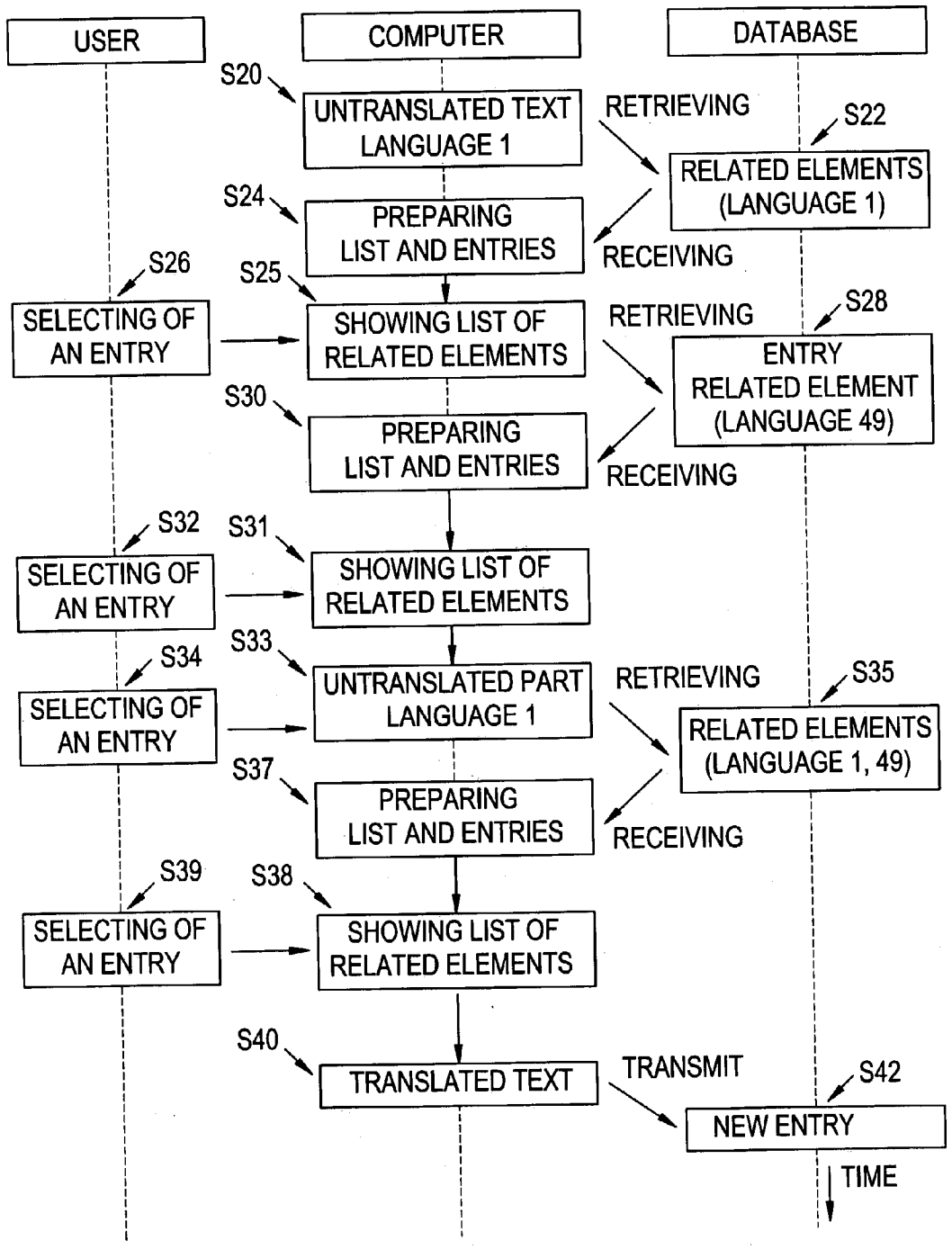

USER
COMPUTER
DATABASE
S20
UNTRANSLATED TEXT LANGUAGE 1
RETRIEVING
S22
RELATED ELEMENTS (LANGUAGE 1)
S24
PREPARING LIST AND ENTRIES
RECEIVING
S26
S25
SELECTING OF AN ENTRY
SHOWING LIST OF RELATED ELEMENTS
RETRIEVING
S28
ENTRY RELATED ELEMENT (LANGUAGE 49)
S30
PREPARING LIST AND ENTRIES
RECEIVING
S32
S31
SELECTING OF AN ENTRY
SHOWING LIST OF RELATED ELEMENTS
S33
S34
SELECTING OF AN ENTRY
UNTRANSLATED PART LANGUAGE 1
RETRIEVING
S35
RELATED ELEMENTS (LANGUAGE 1, 49)
S37
PREPARING LIST AND ENTRIES
RECEIVING
S39
S38
SELECTING OF AN ENTRY
SHOWING LIST OF RELATED ELEMENTS
S40
TRANSLATED TEXT
TRANSMIT
S42
NEW ENTRY
TIME

FIG. 17

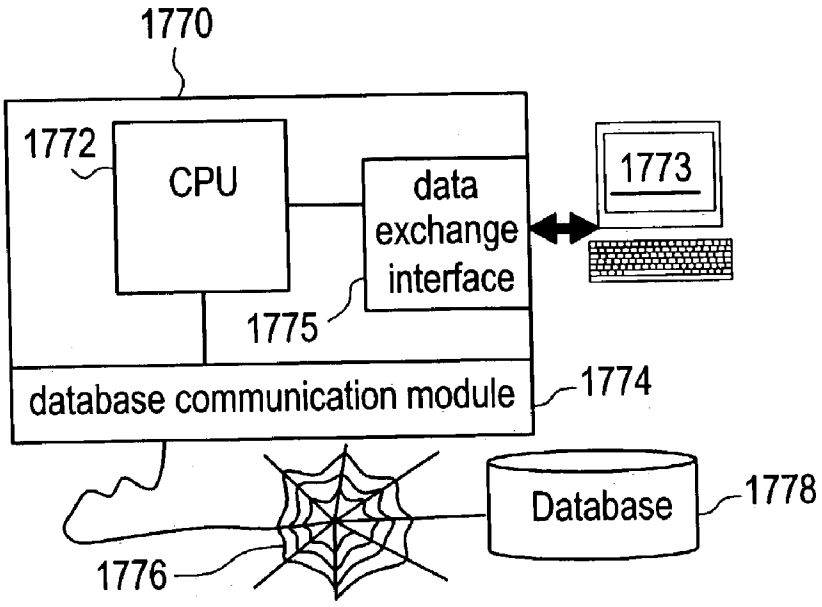

1770
1772
CPU
data exchange interface
1773
1775
database communication module
1774
Database
1778
1776

FIG. 18

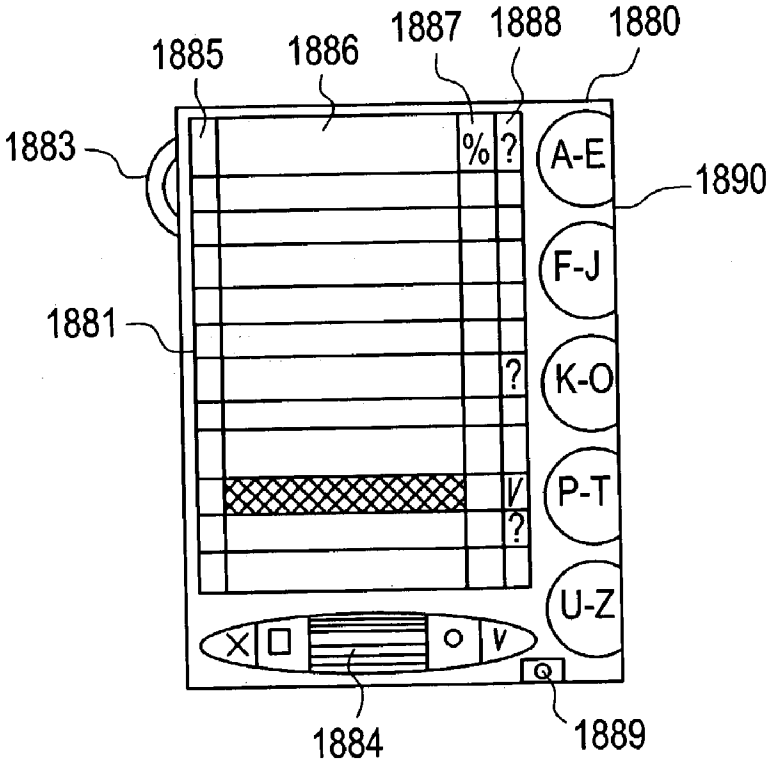
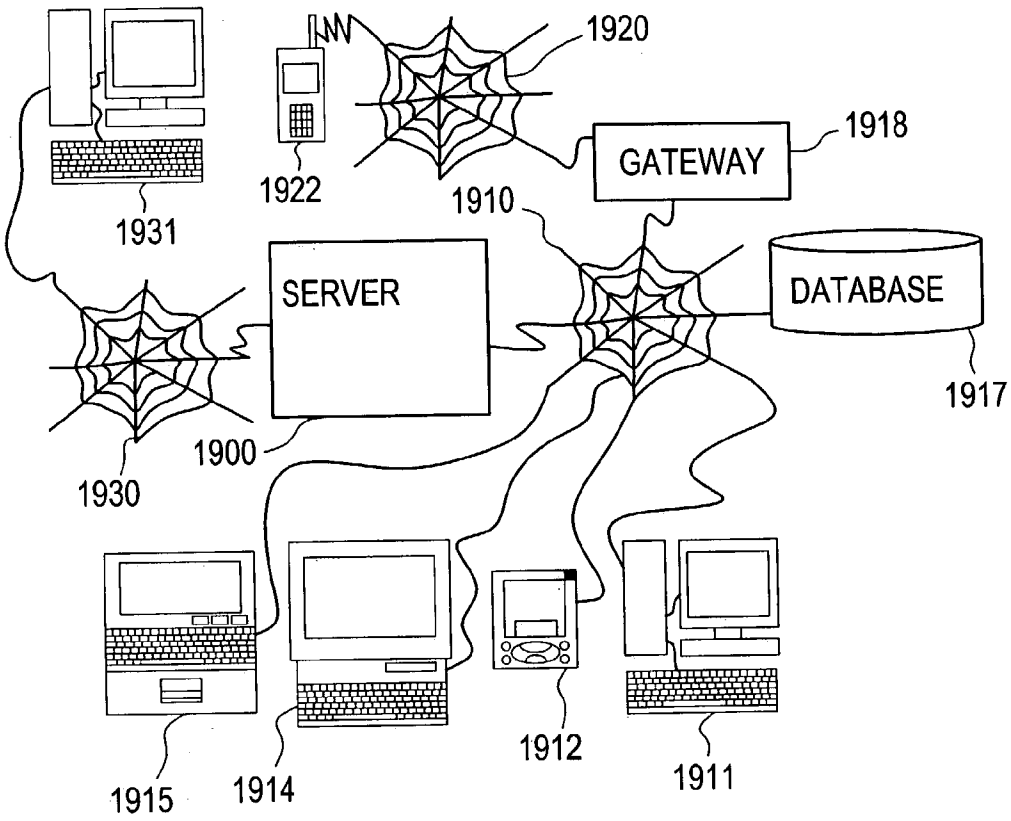

1885
1886
1887
1888
1880
1883
%
?
A-E
1890
F-J
1881
?
K-O
P-T
U-Z
1884
1889

FIG. 19


1920
1918
GATEWAY
1922
1910
1931
SERVER
DATABASE
1917
1900
1930
1915
1914
1912
1911

METHOD AND SYSTEM FOR AUTOMATIC CREATION OF MULTILINGUAL IMMUTABLE IMAGE FILES

CROSS REFERENCES TO RELATED APPLICATIONS

[0001] This application is related to, and claims priority to, European Patent Application No. 01 128 179.7, commonly owned, and entitled "Method and Apparatus for Localizing Software," filed on Nov. 27, 2001, and to European Patent Application No. 02 010 132.5, filed on May 10, 2002, commonly owned, and entitled "Automatic Image-button Creation Process," and to European Patent Application No. 02 010 133.3, filed on May 10, 2002, commonly owned, and entitled "Generation of Localized Software Applications," all of which are hereby incorporated by reference herein in their entirety. Additionally, this application is related to U.S. patent application Attorney Docket No. 30014200-1050, commonly owned, and entitled "Generation of Localized Software Applications," and which is being filed concurrently, and is hereby incorporated by reference herein in its entirety.

BACKGROUND

[0002] 1. Field of the Invention

[0003] The present invention generally relates generally to data processing systems, and, more particularly, to data processing systems for automatically generating multilingual immutable image files containing text.

[0004] 2. Background of the invention

[0005] Data processing devices have become valuable assistants in a rapidly expanding number of fields, where access to, and/or processing of data is necessary. Applications for data processing devices range from office applications such as text processing, spreadsheet processing, and graphics applications to personal applications such as e-mail services, personal communication services, entertainment applications, banking applications, purchasing or sales applications, and information services. Data processing devices for executing such applications include any type of computing device, such as multipurpose data processing devices including desktop or personal computers, laptop, palmtop computing devices, personal digital assistants (PDAs), and mobile communication devices such as mobile telephones or mobile communicators.

[0006] Many such applications are currently implemented using communication networks including individual data processing devices positioned at arbitrary locations. Thus, a data processing device used to provide a service, such as a server maintaining data which can be accessed through a communication network, may be located at a first location, while another data processing device for obtaining a service, such as a client device operated by a user for accessing and manipulating the data, may be located at a second location remote from the first location.

[0007] Servers which provide services are frequently configured to be accessed by more than one client at a time. For example, a server application may enable users at any location and from virtually any data processing device to access personal data maintained at a server device or a network of server devices. Accordingly, users of the client devices may be located in different geographic areas that support different languages, or may individually have different language preferences. The term "user" as used herein refers to a human user, software, hardware, or any other entity using the system.

[0008] For example, a user operating a client device may connect to the server device or network of server devices using personal identification information to access data maintained at the server device or network of server devices, such as an e-mail application or a text document. The server in turn may provide information to be displayed at the client device, enabling the user to access and/or control applications and/or data at the server. This may include the display of a desktop screen, including menu information, or any other kind of information allowing selection and control of applications and data at the server.

[0009] At least some portions of this information may be displayed at the client device using a screen object whose visual elements are stored in an immutable image file containing text information. A "screen object" is an entity that is displayed on a display screen. Examples of screen objects include screen buttons and icons. An "immutable image file" is a file that is not character based, but rather bit based. Thus, editing such files to change any text therein is very difficult. Examples of such files include files created in formats such as .gif, bitmap (.bmp), .tiff, .jpg, .png, .xpm, .tga, .mpeg, .ps, .pdf, .pcx, and other graphic files. This text information may provide information on a specific function which may be activated by selecting the screen object (e.g., by clicking on a button using a mouse and/or cursor device). The formatting of immutable image files such as files in .gif format does not generally support extraction of text strings (e.g., in ASCII format) which may be perceived by viewing a screen display of a rendering of the image file, as the immutable image files (e.g., .gif, bitmap (.bmp), .tiff, .jpg, .png, .xpm, .tga, .mpeg, .ps, .pdf, .pcx, and other graphic files) are generally created based on graphics information instead of text string information (i.e., in bitmap format instead of ASCII, or character-based format). Conventionally, if it is desired to produce immutable images in different languages, it may be necessary to view a rendering of the image to perceive the desired text string in a first language, perform a translation on the perceived individual string into another language, and then manually create a new immutable image file which embodies the translated string information.

[0010] If a number of users access the service from different geographic regions supporting different languages or have different individual language preferences, text information of the images is preferably to be provided in respective languages understood by each individual user. This, however, would require that the text information of the images be provided in different languages. For example, the text strings in the images may be translated into different languages. While this does not pose a problem for a small number of images, where manual translation and creation of the immutable images is possible by manually recreating the immutable image for each different language, a manual translation is neither feasible nor cost effective when a large number of immutable images must be provided in different languages.

[0011] For example, an application provided by a server or network of servers such as an office application may include

a very large number of different menus in different layers including hundreds of individual menu or immutable images containing text. If the service is available in a large number of different geographic areas supporting different languages, a very large number of translation operations and image creation operations is necessary. For example, if an application provides 100 different immutable images (e.g., .gif files or bitmap files) containing text, and if a translation into 20 different languages is necessary, a total of 2000 translation and image creation operations may be necessary.

[0012] Therefore, a need has long existed for a method and system that overcome the problems noted above and others previously experienced.

SUMMARY OF THE INVENTION

[0013] Methods and systems consistent with the present invention provide an improved translation system that allows users to view screen objects of a user interface in different languages even though the screen object's visual elements are stored in an immutable image file (e.g., a file having a graphic format such as .gif, bitmap (.bmp), .tiff, .jpg, .png, .xpm, .tga, .mpeg, .ps, .pdf, or .pcx). To do so, methods and systems consistent with the present invention automatically generate a number of immutable image files, each containing text in different languages. These files are then made available to users so that they may select a preferred language in which to view the screen objects of the user interface.

[0014] To facilitate the automatic generation of the image files in different languages, when the screen objects are initially developed, the textual elements of the screen objects are associated with the image files as text strings. Then, the improved translation system automatically translates the text strings into different languages and generates image files that contain the translated textual elements. As a result, these image files are made available so that when a user wishes to display a user interface, it will be displayed in the language of their choice. For example, a browser user may select a language of their choice and receive web pages from a web server in this language. Such functionality facilitates the international use of web sites.

[0015] In accordance with methods, systems and articles of manufacture consistent with the present invention, a method in a data processing system for localizing an immutable image file containing text in a first language is provided. The method translates the text from the first language into a second language that is different from the first language, and automatically generates a translated immutable image file containing the text in the second language.

[0016] In accordance with methods, systems and articles of manufacture consistent with the present invention, a computer-readable medium is provided. This computer-readable medium is encoded with instructions that cause a data processing system for localizing an immutable image file containing text in a first language to perform a method. The method translates the text from the first language into a second language that is different from the first language, and automatically generates a translated immutable image file containing the text in the second language.

[0017] In accordance with methods, systems and articles of manufacture consistent with the present invention, a data processing system is provided for localizing an immutable image file containing text in a first language. The data processing system comprises an immutable image file creation system that translates the text from the first language into a second language that is different from the first language, and automatically generates a translated immutable image file containing the text in the second language; and a processor for running the immutable image file creation system.

BRIEF DESCRIPTION OF THE DRAWINGS

[0018] **FIGS. 1*a*-1*b*** depict block diagrams of an exemplary data processing system suitable for using immutable image files in a number of languages.

[0019] **FIG. 2** depicts a block diagram of a data processing system for automatically generating localized immutable image files containing text, the system suitable for practicing methods and systems consistent with the present invention.

[0020] **FIG. 3** depicts a flowchart illustrating steps of a method for automatically generating localized immutable image files containing text in accordance with methods, systems, and articles of manufacture consistent with the present invention.

[0021] **FIG. 4** depicts a block diagram illustrating a logical flow of an exemplary system for automatically generating localized immutable image files containing text in accordance with methods, systems, and articles of manufacture consistent with the present invention.

[0022] **FIG. 5** depicts a block diagram illustrating a logical flow of an exemplary system for automatically generating localized immutable image files containing text in accordance with methods, systems, and articles of manufacture consistent with the present invention.

[0023] **FIG. 6** depicts a flowchart illustrating steps of an exemplary method for automatically generating immutable image files containing text in a number of languages in accordance with methods, systems, and articles of manufacture consistent with the present invention.

[0024] **FIG. 7** depicts a flowchart illustrating steps of an exemplary method for automatically creating immutable image files having text in a number of languages in accordance with methods, systems, and articles of manufacture consistent with the present invention.

[0025] **FIG. 8** depicts a flowchart illustrating steps of an exemplary method for automatically creating immutable image files in a number of languages in accordance with methods, systems, and articles of manufacture consistent with the present invention.

[0026] **FIGS. 9*a*-9*b*** depict flowcharts illustrating steps of an exemplary method for identifying text and translating the text.

[0027] **FIG. 10** depicts a flowchart illustrating steps of an exemplary method for identifying text and translating the text both in source code text strings and in software related text.

[0028] **FIGS. 11*a*-11*b*** depict flowcharts illustrating steps of an exemplary method for identifying text and translating the text both in source code text strings and in software related text.

[0029] **FIG. 12** depicts a flowchart illustrating steps of an exemplary method for identifying text element types and searching for matching strings in a database.

[0030] **FIG. 13** depicts a flowchart illustrating steps of an exemplary method for validating a translation element and storing the validated element in a database.

[0031] **FIG. 14** depicts an exemplary table in a pretranslation database.

[0032] **FIG. 15***a* depicts an exemplary display for a user processing a translation of text using exact matching.

[0033] **FIG. 15***b* depicts a flowchart illustrating exemplary method steps for a user, a computer, and a database for translating text using exact matching.

[0034] **FIG. 16***a* depicts an exemplary display for a user processing the translation of text using fuzzy matching.

[0035] **FIG. 16***b* depicts a flowchart which depicts exemplary method steps for a user, a computer, and a database for translating text using fuzzy matching.

[0036] **FIG. 17** depicts an exemplary system for translating text using a pretranslation database.

[0037] **FIG. 18** depicts an exemplary sentence based electronic translation dictionary.

[0038] **FIG. 19** depicts an exemplary networked system for use with an exemplary translation technique.

DETAILED DESCRIPTION

[0039] Methods and systems consistent with the present invention provide an improved translation system that allows users to view screen objects of a user interface in different languages even though the screen object's visual elements are stored in an immutable image file (e.g., a file having a graphic format such as .gif, bitmap (.bmp), .tiff, .jpg, .png, .xpm, .tga, .mpeg, .ps, .pdf, or .pcx). To do so, methods and systems consistent with the present invention automatically generate a number of immutable image files, each containing text in different languages. These files are then made available to users so that they may select a preferred language in which to view the screen objects of the user interface.

[0040] To facilitate the automatic generation of the image files in different languages, when the screen objects are initially developed, the textual elements of the screen objects are associated with the image files as text strings. Then, the improved translation system automatically translates the text strings into different languages and generates image files that contain the translated textual elements. As a result, these image files are made available so that when a user wishes to display a user interface, it will be displayed in the language of their choice. For example, a browser user may select a language of their choice and receive web pages from a web server in this language. Such functionality facilitates the international use of web sites.

[0041] Reference will now be made in detail to an implementation in accordance with methods, systems and articles of manufacture consistent with the present invention as illustrated in the accompanying drawings. The same reference numerals may be used throughout the drawings and the following description to refer to the same or like parts.

[0042] **FIG. 1***a* depicts a block diagram of an exemplary data processing system including a client **102**, a client **104** and a server **100**, similar to a data processing for implementing StarOffice™ running in Sun One Webtop, developed by Sun Microsystems, in which a number of client devices may access the server device **100** to use a service application such as a word processor, a spreadsheet application or a graphics application. Client **102** or client **104** may include a human user or may include a user agent.

[0043] Clients **102** and **104** each include a browser **106** and **108**, respectively. Browsers **106** and **108** may be used for displaying data provided by the server **100** (e.g., HTML data or XML data). The data may include references to immutable image files containing text (e.g., .gif files or bitmap files) which may be downloaded to the client browsers **106** and **108** for display as images such as control buttons or icons.

[0044] Clients **102** and **104** may communicate with server **100** through communication networks which include a large number of computers, such as local area networks or the Internet. Access to information located in either client **102**, client **104**, or server **100** may be obtained through wireless communication links or fixed wired communication links or any other communication means. Standard protocols for accessing and/or retrieving data files over a communication link, for example, over a communication network, may be employed, such as a HyperText Transfer Protocol ("HTTP").

[0045] **FIG. 1***b* depicts a block diagram of the exemplary data processing of **FIG. 1***b*, more particularly showing browser displays **120** and **130** on clients **102** and **104**, respectively, of immutable images **122** and **132** having corresponding text in different languages. Server **100** includes a secondary storage device **140** which includes a German image file **142** which is displayed as a button **122** on the browser display **120** of client **102**, and an English image file **144** which is displayed as a button **132** on the browser display **130** of client **104**. Each of client **102** and **104** may specify a language in order to be provided with image files in the appropriate language for display during execution of the same general application such as a word processing application which may be executed as a separate process for each individual user. Thus, each individual user may use the application and receive screen displays including text information in a preferred language of the individual user.

[0046] **FIG. 2** depicts a block diagram of a data processing system **200** suitable for practicing methods and systems consistent with the present invention, which provides immutable image files having text **232** and **234** in a number of languages. **FIG. 2** particularly illustrates how immutable images containing text strings in different languages may be created at reduced complexity and costs.

[0047] System **200** includes a server **201** for generating the immutable image files and a server **260** for translating strings in particular languages into different languages. Servers **201** and **260** communicate with each other via a connection **245** which may be a direct connection, a network connection such as a LAN connection or a WAN connection such as the Internet. Text strings in a first language are identified on the server **201** and are transferred to the server **260** by a translator module **218** on the server **201** as a file of text strings **280** for translation into at least one second

language that is different from the first language. After translation, the text strings are transferred back to the server **201** for processing to generate immutable image files containing the text strings in different languages. The immutable image files containing the text strings may then be used in applications so that users may view screen objects on a display in their choice of different languages.

[**0048**] Server **260** includes a CPU **262**, a secondary storage device **268** which includes a translation database **270**, a display **265**, an I/O device **264**, and a memory **266**, which communicate with each other via a bus **291**. Memory **266** embodies an operating system **295** and a translation module **278** which inputs text strings from the file of text strings **280** and translates the strings into different languages using the translation database **270**. One suitable translation process for use with methods and systems consistent with the present invention is described below. However, other processes may also be used. The translated text strings are then transferred back to server **201** via the connection **245**. Alternatively, the elements of servers **260** and **201** are all located on one device.

[**0049**] Server **201** includes a CPU **202**, a secondary storage device **208**, a display **205**, an I/O device **204**, and a memory **206**, which communicate with each other via a bus **231**. Memory **206** embodies an operating system **250**, a translator module **218**, a parser **220**, and a script **222** which is associated with an image manipulation program **224** to input text information from a text string file **226** and a template **230** to generate immutable image files **232** and **234**, each of which contains text in a language typically different from the other, when the script **222** is initiated. The script **222** may be initiated by a call entered by a user using the operating system **250**, or the script **222** may be initiated by a call included in a batch file which is executed by a user or system command. A batch file of a number of calls to the script **222** may be stored in the secondary storage device **208** for access by a user or a system in order to generate a number of immutable image files in a number of different languages, and for a number of different base images (e.g., cancel button, system setting buttons). The parser **220** may be used to parse text information from a text element file **228** to create the text string file **226**. A "text element file" is a file which contains text elements. An exemplary text element file written in XML is shown below. A "text element" is a property or attribute associated with a text string, including the text string itself. Examples of text elements include the text string, a language of the text string, information regarding an associated immutable image file which includes the text string (e.g., display characteristics), and location information indicating a storage location for the associated immutable file.

[**0050**] The parser **220** may also parse the text element file **228** to obtain text strings for the translator module **218** to transfer to the server **260** for translation. After translation, the translated strings may be transferred back to the translator module **218** to be merged back into the text element file **228** to enable multilingual applications by accessing the text elements in different languages.

[**0051**] The translation database **270** may be used to store strings in different languages, as well as other information such as template information associated with the template **230**. An example of a very simple database table containing template information is:

Template	File Name	Width	Height	Color
SysDef	SysDef.gif	300	60	16
Cancel	Cancel.gif	200	60	25

[**0052**] In the table as shown, information is provided for two immutable image files, one for a system user button and one for a cancel button. A base file name is provided for storing an immutable image file for each template. Additionally, width, height, and color information of the buttons is provided. Additional information such as shape, a base language text string (e.g., "System Defaults" or "Cancel" if English is the base language for translating the strings for these buttons), placement of the string within the button, and information regarding a previously generated immutable image file are stored in additional columns (not shown). The information regarding a previously generated immutable image file may be used to generate an empty (not having text) immutable image file, so that a translated string may be merged into the file. Thus, a call to the script **222** may be generated by requesting template information in the translation database **270**, and may be included in the call to the script **222** in lieu of using the template file **230**. A specific example of a script, a call to the script, a text element file, and a text string file are shown below. An exemplary image manipulation program used for the example is GIMP (GNU Image Manipulation Program). A commercially available program may be used for translating the text strings, such as a translation program available from Trados. Such a program may be used by configuring code to obtain the strings for translation, provide the strings to the translation program, receive the translated strings, and add the translated strings into the text element file.

[**0053**] Alternatively, the parser **220** may be configured to extract the text for translation from at least one image creation file including position information and text elements from a number of immutable image files containing text in the first language. For example, the text elements may be extracted from an image creation file such as a HTML (Hyper Text Markup Language) file or an XML (Extensible Markup Language) file or any other image creation file suitable for interpretation at a data processing device (e.g., user **102** or **104**) for display.

[**0054**] Thus, the parser **220** may automatically extract text elements from the image creation file and collect the text elements in the text element file **228** for further processing. An image creation file may generally be used to create a screen display (e.g., at a client accessing a service application at a server or group of servers).

[**0055**] After the translated text elements are obtained, the parser **220** may further merge the translated text elements in a second language into an image creation file for the second language. Thus, the text elements for the immutable image files in the second language may be provided in the image creation file for the second language, so that a client operated by a user who is fluent in the second language may be provided with a display screen including text elements in the second language and a display of immutable image files containing text in the second language.

[0056] This example is particularly suitable for application in network environments such as the Internet.

[0057] The original uses of the Internet were electronic mail (e-mail), file transfers (ftp or file transfer protocol), bulletin boards and newsgroups, and remote computer access (telnet). The World Wide Web (web), which enables simple and intuitive navigation of Internet sites through a graphical interface, expanded dramatically during the 1990s to become the most important component of the Internet. The web gives users access to a vast array of documents that are connected to each other by means of links, which are electronic connections that link related pieces of information in order to allow a user easy access to them. Hypertext allows the user to select a word from text and thereby access other documents that contain additional information pertaining to that word; hypermedia documents feature links to images, sounds, animations, and movies.

[0058] The web operates within the Internet's basic client-server format. Servers include computer programs that may store and transmit documents (i.e., web pages) to other computers on the network when requested, while clients may include programs that request documents from a server as the user requests them. Browser software enables users to view the retrieved documents. A web page with its corresponding text and hyperlinks may be written in HTML or XML and is assigned an online address called a Uniform Resource Locator (URL).

[0059] Information may be presented to a user through a graphical user interface called a web browser (e.g., browser **106** or **108**). When presented with data in the proper format, the web browser displays formatted text, pictures, sounds, videos, colors, and other data. To instruct a web browser to present the data in the desired manner, HTML was originally used. HTML is a language whereby a file is created that has the necessary data and also information relating to the format of the data.

[0060] Further, XML has emerged as a next generation of markup languages. XML is a language similar to HTML, except that it also includes information (called metadata) relating to the type of data as well as the formatting for the data and the data itself.

[0061] Thus, this example may be used to extract text elements from HTML or XML pages or other types of pages such as image creation files, in order to facilitate provision of immutable image files in different languages for internationalizing service applications accessible over the Internet or any other network.

[0062] The text element file **228** may include any kind of collection of text elements stored in a memory area of the data processing system **200**, or stored at any other location. For example, the text element file **228** may be provided at an external location, accessible from the data processing system **200** through a communication network. Thus, the text element file **228** may be a centralized resource accessible by a number of data processing systems such as the data processing system **200** from arbitrary locations. Accordingly, a decentralized generation of immutable image files in various languages may be possible, including a consistent use of translated text elements. This allows consistent presentation of textual information to users, in contrast to environments in which different translation operations yield different translations of text elements for use in the same or different service applications.

[0063] Collecting text elements in the text element file **228** may be facilitated by a check-in process, allowing registration of text elements and translations thereof in the text element file **228**, in order to maintain information on languages in which the individual immutable image files are available.

[0064] The text element file **228** may include an XML file and an XML editor may be used for generating the translated text elements. Further, the text element file **228** may include information of a desired output file name of the respective immutable image files and information regarding a template, such as template **230**, which may be needed for creation of an immutable image file.

[0065] Thus, the text element file **228** may also serve as a basis of information enabling proper handling of the created immutable image files and enabling selection of a suitable template for the text elements which are collected. For example, different types of text elements may be associated with different templates, enabling creation of immutable image files in certain categories (e.g., characterized by size, color or any other graphic information). For example, immutable image files related to data handling may be generated using a first template, whereas immutable image files enabling a setting of references (e.g., for display) may be generated using a second template.

[0066] The translation module **278** for obtaining translated text elements in at least one second language may include a program or a sequence of instructions (as described below) to be executed by the CPU **202**, by corresponding hardware or a combination of hardware and software for realizing the functionality of obtaining translated text elements in the second language. The translation of the text elements may be accomplished by using pretranslation databases.

[0067] The translator module **218** may be configured to extract relevant information from the text element file **228** or may be configured to import this information into a database such as the translation database **270** for improved handling of the text elements. After identifying individual text elements, the translator module **218** may invoke a translation service providing a translation of the text element into the second language.

[0068] The translation service may include an internal application, such as a thesaurus provided in a memory such as memory **206**, within or accessible from the data processing system **201**. Further, the translation service may be an external translation service provided by an external application program (e.g., offered by a third party). For example, the translator module **218** may be configured to invoke a web-based translation tool, accessed through a computer network such as a local area network or the Internet, or a combination thereof.

[0069] The translated image text elements may be collected in the translation database **270** and/or may then be merged into the text element file **228**.

[0070] A program may be provided having instructions configured to cause a data processing device to carry out the method of at least one of the above operations. Further, a computer readable medium may be provided, in which a program is embodied, where the program is to make a computer execute steps of the method discussed above.

[0071] Also, a computer-readable medium may be provided having a program embodied thereon, where the program is to make a computer or a system of data processing devices to execute functions or operations of the features and elements of the examples described previously. A computer-readable medium may be a magnetic or optical or other tangible medium on which a program is recorded, but may also be a signal, e.g., analog or digital, electronic, magnetic or optical, in which the program is embodied for transmission. Further, a computer program product may be provided comprising the computer-readable medium.

[0072] The text string file **226** may include information of a desired output file name of the respective immutable image files and information regarding a template, such as the template **230**, used for creation of an immutable image file. A template **230** may include any kind of image file information. For example, a template may include information related to at least one of image size; image shape; color; and position of a text element in an image. A number of templates for various images may be provided and stored in the memory **206** or at an external location. Alternatively, the template information may be included in the call to the script **222**.

[0073] Thus, the text string file **226** may also serve as a basis of information enabling proper handling of the created images and enabling selection of a suitable template for the text elements collected. For example, different types of text elements may be associated with different templates, allowing creation of images in various categories such as size, color or any other type of graphic information. For example, images related to data handling may be generated using a first template, whereas images enabling a setting of references such as for display may be generated using a second template.

[0074] The image size may specify an area of the image such as an area in a display screen to be displayed at a client unit. Further, the image size may specify a minimum size of the image, such as a size suitable to fit the text element. The image shape information may specify any graphic parameters for obtaining certain graphic effects such as shading or 3-D effects. Further, the shape may also specify geometric shapes such as rectangles, circles and polygons.

[0075] The image color may include information related to shading of the image to obtain further graphic effects. The image information may further include information regarding a position of a text element, for an image enabling a proper placement of the text element within the image and/or may relate to a position of the image in a display screen, as specified in an image creation file (e.g., an HTML file), enabling proper placement of the image such as in a menu including different images to be displayed at a client device.

[0076] The system of **FIG. 2** facilitates a generation of immutable images for screen displays similar to those discussed with regard to **FIG. 1**, wherein the images have text elements with different languages, in order to adapt a service application to different languages of different users. For example, a first user who is fluent in a first language may be provided with display screens displaying screen objects with text information in the first language, and a second user who prefers a second language may be provided with a display screen displaying screen objects with text elements in the second language.

[0077] The provision of display screens displaying screen objects with text elements in the first language and the second language can be simplified, particularly by collecting the text elements of the images in the text element file **228** and effecting creation of images for the text elements in the text element file **228** in the different languages.

[0078] The data processing system **200** may generally include any computing device or group of computing devices connected with one another. For example, the data processing system **200** may be a general purpose computer such as a desktop computer, a laptop computer, a workstation or combinations thereof. Further, the functionality of the data processing system **200** may be realized by distributed data processing devices connected in a local area network such as a company-wide computer network or may be connected by a wide area network such as the internet, or combinations thereof.

[0079] The image manipulation program **224** may be a graphics application provided as a stand alone program, e.g., offered by a third party. Further, the image manipulation program **224** may be a commercially available graphics application enabling generation of immutable image files based on given text elements and given template information. For example, the image manipulation program **224** may include the GNU (Gnu's Not UNIX) image manipulation program GIMP, which is described at www.gimp.org.

[0080] The system **200**, as shown, is configured to instruct the image manipulation program **224** to generate immutable image files based on given parameters and may be adapted to generate a script to instruct the image manipulation program **224** to generate the immutable image files based on the given parameters. If the image manipulation program **224** is script enabled, a script such as script **222** may conveniently instruct the image manipulation program **224** based on the parameters. The script may be a sequence of instructions provided to the image manipulation program **224**, instructing the image manipulation program **224** to generate an immutable image file having certain properties and including a specific text element. Further, the script **222** may include a text element for insertion into the immutable image file. The script may be written and intermediately stored in a file (e.g., as a batch file stored in secondary storage **208**), to be provided to the image manipulation program **224**, and may include the parameters discussed above (i.e., parameters needed to generate the immutable image file).

[0081] The parameters may include a template and the text element of the immutable image. For example, the parameters may include immutable image file information as discussed above, including image size, shape, color and/or position of a text element within the image.

[0082] Additionally, the parameters may include a number of text elements of the image and at least one template, enabling a concurrent generation of various immutable image files with different text elements and/or templates.

[0083] Each text element may be associated with a particular template (e.g., specified in the input file), or the text elements may be categorized in groups, each group associated with a particular template.

[0084] Further, the parser **220** may be configured to generate the text string file **226** including at least one text

element (e.g., from the text element file **228**) and the image manipulation program **224** may be instructed, using the script **222**, to access the text string file **226** in order to retrieve at least one text element of the text string file **226**. The text string file **226** may be intermediately stored in the memory **206**, or at any other location. Thus, the text string file **226** may be based on at least a portion of the text element file **228**.

[**0085**] The text string file **226** may further include at least one of a language identifier; and a desired name of an immutable image file to be created. The language ID facilitates an easier classification of immutable image files created based on the text elements, and a desired name of an immutable image file to be created facilitates easy access to the generated graphic elements including the image files, such as by updated HTML code used to instruct the display on a display screen for display at a client unit.

[**0086**] By including the name of the input file and the immutable image file information in the script **222**, the image manipulation program **224** may access the text string file **226**, retrieve the text elements and generate the immutable image file based on the immutable image file information. Alternatively the immutable image file information may be directly included in the text string file **226**.

[**0087**] Alternatively, the script **222** and image manipulation program **224** may be configured to generate an empty immutable image file based on the template **230** and may be configured to merge the translated text element into the empty immutable image file. Thus, a collection of immutable image files with no text may be created, and text elements may be included on demand. This enables generation of a generic image creation file (e.g., an HTML file) for generating a screen display at a client unit. The text elements in the respective languages can then be included in the generic image creation file in order to generate image creation files in different languages.

[**0088**] The image manipulation program **224** may generate immutable image files for storage at the data processing system **200** or at any other location. The immutable image files may also be directly included into image creation files for generating screen displays at client units or image creation files with references to the immutable image files stored may be created.

[**0089**] Two exemplary immutable image files which may be created using system **200** are an immutable image file in a first language, English, displaying the English expression "system defaults", and a second image button in the second language, German, which includes the corresponding German expression "Systemeinstellungen." The buttons **122** and **132** of **FIG. 1***b* illustrate screen displays of the immutable image files including these text elements.

[**0090**] **FIG. 3** depicts a high-level flowchart illustrating steps of a method for automatically localizing and immutable image file containing text in a first language, the method suitable for use with methods, systems and articles of manufacture consistent with the present invention, such as the system of **FIG. 2**. First, a translation system such as the system shown on server **260** translates the text from the first language into a second language that is different from the first language (Step **300**). Then, script **222** and image manipulation program **224** automatically generate a translated immutable image file containing the text in the second language (Step **302**).

[**0091**] **FIG. 4** depicts an exemplary high level logical flow using elements of a system for generating immutable image files containing specific exemplary text in different languages according to another example. **FIG. 4** particularly illustrates extraction of text elements and merging of translated image text elements into image creation files such as HTML or XML files. As described previously, screen displays may be generated for client devices accessing services at a server. As the client devices may be located in different geographic areas supporting different languages, the service provided at the server may be enabled to generate image creation files or screen displays for the client devices in different languages.

[**0092**] **FIG. 4** illustrates two screen displays based on image creation files, in different languages. A first screen display **410** includes English language buttons **411** and **412**. The displayed buttons in the present example are used as menu items allowing control of a corresponding service application at a server. The button **411** in the present example is assumed to relate to system settings and therefore shows the English expression "system settings." The second image button **412** relates to user group functionality, and therefore displays the expression "user groups."

[**0093**] An image creation file corresponding to the screen display **410** may be used for users preferring the English language and may thus be provided from the server to corresponding client devices which have selected English as the preferred language.

[**0094**] The second screen display **450** shown in **FIG. 4** shows German language buttons and therefore is suitable for users in a region that supports the German language, or users who have selected German as their preferred language. Corresponding to the buttons **411** and **412** of the screen display **410**, the screen display **450** includes buttons **451** and **452**. In correspondence to button **411**, the button **451** includes the German expression "Systemeinstellungen," indicating that the corresponding button also relates to system functions. Further, the button **452** corresponding to the button **412** contains the German expression "Benutzergruppen" and indicates that this button also relates to a user group functionality. The screen displays may include a larger number of immutable images and/or sub-menus or sub-screens, so that there may well be a very large number of different immutable image files to be considered.

[**0095**] The exemplary text elements may be extracted from the screen display **410** corresponding to the English language and may be translated into text elements for the screen display **450** for the German language. This extraction and creation process may be carried out offline. Thus, immutable image files for screen displays in different languages may be created and stored beforehand and made accessible to a service application such as a text processing application.

[**0096**] Further, the extraction and creation process for providing immutable image files containing text in different languages may be provided on demand (e.g., if a user with a particular preferred language logs into the server). In this case the screen display (i.e., the image creation file) in the corresponding language including text elements and images in the preferred language may be created dynamically (i.e., on demand).

[**0097**] In **FIG. 4**, an arrow **460** illustrates a corresponding extraction process enabling extraction of the text elements

from the image buttons **411** and **412** and provision to the text element file **420**. The extraction process may be carried out as described previously. The text element **420** therefore will contain, in the present simplified example, the expression "system settings" and "user groups." Then, in steps **461** and **462** translated text elements involving a translation service **430** are obtained.

[**0098**] The text element file **420** may include corresponding language identifiers (i.e., showing that the original text elements are in the English language and that needed text elements should be in the German language).

[**0099**] After the translation step, the text element file **420** will further include the German expressions "Systemeinstellungen" and "Benutzergruppen." Thereafter, as illustrated by arrow **463**, a graphic program **440** is instructed to generate immutable image files in the German language, corresponding to the buttons **411** and **412** in the English language. The creation of the immutable image files was described previously.

[**0100**] As a result of the immutable image files creation process, immutable image files corresponding to the buttons **451** and **452** are generated. The immutable image files may be checked into a database of immutable image files or may be stored intermediately in any other way. Thereafter, as outlined by an arrow **464**, the image screen **450** for display at a client device or generation of a corresponding image creation file is illustrated.

[**0101**] Accordingly, an English language user accessing a particular service controlled by the screen displays **410** or **450** at the client units may be provided with the English language screen display **410**, whereas a German language user may be provided with the German language screen display **450**.

[**0102**] **FIG. 5** depicts an exemplary logical flow of elements of a system for obtaining and using immutable image files in a number of different languages according to another example. **FIG. 5** particularly shows how the immutable image files in different languages may be used to supply users with display screens in different languages. **FIG. 5** shows a data processing device **500** for obtaining image buttons in a number of different languages.

[**0103**] The data processing device **500** may include a single processing device or a number of processing devices in communication with one another. The data processing device **500** includes a text element file **501** (e.g., as described previously). The text element file **501** may be stored in an internal memory or may be provided at an external memory location.

[**0104**] Further, the data processing device **500** includes an input file **502**. The input file **502**, as described previously, may include at least one text element, and may further include a language ID and/or name of an immutable image file to create. The input file may also be stored at an internal location or at any other external memory location.

[**0105**] The data processing device **500** also includes a template **503** including immutable image file information such as image size and/or image shape and/or color and/or a position of a text element within the image.

[**0106**] The data processing device **500** may store a number of templates at an internal memory location or an external memory accessible from the data processing device **500**. A suitable template (e.g., the template **503**) may be selected based on information contained in the text element file **501** or on information on image parameters obtained from another source. Further, the template **503** may be dynamically generated.

[**0107**] Further, the data processing device **500** includes a script **504**, as described previously. The script **504** may be generated based on the input file **502**, and may include information of the input file **502** or a reference or storage location enabling access to the input file. Further, the script **504** may include image information from the template **503**.

[**0108**] The script **504** may be intermediately stored at the data processing device **500** or may be dynamically generated and transferred to the graphics application program **505**. The graphics application program **505** may be a graphics application (e.g., GIMP) as described previously, provided at the data processing device, as shown in **FIG. 5**, or may be provided at an external location, arranged to be accessed from the data processing device **500** through a communication link. Based on the script **504**, the graphics application program **505** is instructed to generate at least one immutable image file, as described previously. The immutable image files with text elements in different languages, as illustrated by reference numeral **506**, may be stored in the data processing device **500**, or may be stored at an external location.

[**0109**] Further, the data processing device **500** invokes a translation service **510** in order to obtain translated text elements. The translation may be performed by a translation system as described below in more detail. Additionally, a server **520** which provides services to a number of users **541** and **542**. The services enable users to access and/or manipulate data at the server **520** from remote locations through client devices. The server **520** may include a large computing device providing services such as office applications, communications applications or any other applications.

[**0110**] Exemplary clients **541** and **542** are arranged to access the server **520** through the Internet, as illustrated at **530**, in order to control execution of a service application, as described previously. As the clients **542** and **541** may be located in different geographic areas that support different languages, or as users operating the clients may have different language preferences, the application at the server **520** generates image creation files or screen displays for the clients **541** and **542** in different languages.

[**0111**] **FIG. 5** illustrates two exemplary screen displays **521** and **522** in a first and a second language. The server **520** provides image creation files to the clients **541** and **542** in the preferred language. In operation, in order to obtain text elements in a number of languages, the data processing device **500** may be configured to extract text elements (e.g., from an image creation file corresponding to screen display **521** at the server **520**), including text elements in a first language TE**1**, TE**2**, TE**3**, and TE**4**. The text elements are stored in the text element file **501**.

[**0112**] Thereafter, the data processing device **500** obtains translated text elements in the second language using the translation service **510**. The translated text elements are then merged into the text element file **501**. Thereafter, based on the text element file **501** the input file **502** is generated and, based thereon and on the template **503** the script **504** is

generated. The script **504** then instructs the graphics application program **505** to generate the immutable image files with text elements TE1, TE2, TE3, and TE4 in the second language.

[0113] The server **520** then uses the immutable image files in the various languages to generate image creation files (e.g., in HTML or XML), such as image creation files corresponding to the screen display **521** and **522**.

[0114] **FIG. 6** depicts a flowchart depicting steps of an exemplary method for generating screen objects (e.g., image buttons) in a number of languages. The steps of **FIG. 6** may be carried out using the system shown in **FIG. 2**; however, **FIG. 6** is not limited thereto.

[0115] First, a parser generates an input file including at least one image text element, and/or a language ID and/or a name of an immutable image files to be created (Step **601**). The input file may resemble the text string file **226** and may correspond to at least part of the text element file **228**.

[0116] Next, a parser obtains image information, (i.e., information on the image such as color, shape, dimensions, and position of a text string within the image) from a template (Step **602**). The image information may relate to one or a number of templates (as required for the text elements). For example, image information for different groups of text elements and/or different languages may be obtained. Obtaining the image information may also include dynamically generating image information based on information from the input file (e.g., a size of a text element in a particular language).

[0117] After the image information is obtained, a script is initiated to instruct a graphic application program (e.g., the image manipulation program **224** which may be implemented using the GNU (Gnu's Not UNIX) image manipulation program GIMP) to generate at least one immutable image file, the script including information on the input file (e.g., a file name or address, and the at least one immutable image file) (Step **603**). For example, the script may be used to instruct a graphic application program on a remote server to create an immutable image file based on given parameters. The given parameters may be obtained from the input file and may be provided with the image information. Based on the script the graphic application program may access or retrieve the input file from a memory including at least one of text elements, language IDs and immutable image file names. Further, the input file may also include the image information.

[0118] Next, under the direction of the script, the graphic application program generates at least one immutable image file based on the image information and the translated text element (Step **604**). For example, a number of immutable image files for a particular text element in different languages may be created simultaneously (e.g., if the input file includes one text element in a number of languages). Further, a number of immutable image files may be created simultaneously for a number of text elements in one language. Moreover, a combination of both cases is possible, wherein a number of immutable image files for a number of text elements in a number of languages may be created.

[0119] Further, a number of templates may be employed to generate immutable image files for different groups of text elements. The text elements may be classified according to language ID or according to content, in order to be able to make characteristics or appearance of an immutable image file language dependent and/or content dependent. Different templates may be used for each classified group. For example, text elements for data manipulation may be classified into a first group for images having a first appearance, and text elements for setting preferences may be collected in a second group for images having a second appearance.

[0120] The graphic application program then stores at least one immutable image file using the at least one immutable image file name (Step **605**). The file names may enable a convenient identification of the immutable image files in later processing steps (e.g., during generation of image creation files, for example for clients fluent in different languages).

[0121] As an alternative, subsequent to Step **603**, the graphic application program may generate an empty immutable image file (i.e., an immutable image file free of a text element) based on the image button information (Step **606**). Thus, immutable image files for receiving text elements may be created. Next, the translated text elements may be merged into the respective empty immutable image files (Step **607**). Thus, a library of immutable image files may be created and the text elements may be merged into the retrieved empty immutable image files.

[0122] In an example shown below, the graphic application program described previously includes the GNU (Gnu's Not UNIX) image manipulation program GIMP.

[0123] In the example, creation of immutable image files **232** and **234** is accomplished by a GIMP call, a GIMP input file (e.g., text string file **226**) and a GIMP script (e.g., script **222**). Code examples of a GIMP call, a GIMP input file (file name: .utf8.txt) and a GIMP script are given below. The GIMP call is initiated (e.g., on a command line of an operating system such as Windows or from a batch file by the operating system) and in turn instructs the GIMP to input text string information from the input file and template information from the call itself.

[0124] **FIG. 7** depicts a flowchart depicting steps of another exemplary method for generating immutable image files in different languages. The steps of **FIG. 7** may be carried out using the system shown in **FIG. 2**; however, **FIG. 7** is not limited thereto.

[0125] The example of **FIG. 7** particularly outlines steps for efficient handling of text elements and for obtaining translated text elements. First, a parser transfers text elements into an XML text file or a database (Step **701**). Transferring the text elements may be effected by extracting text elements from image creation files or from screen displays including immutable image files (e.g., HTML or XML files for display at a client device). Transferring the text elements to an XML file or a database facilitates efficient handling of the text elements. The database may be provided internal to a data processing device such as the data processing system **200** shown in **FIG. 2**, or at any other location. The XML file including the text elements may be visualized for an operator using an XML editor.

[0126] Next, a web-based tool or an XML editor obtains translated image text elements (Step **702**). The text elements may be automatically extracted from the XML text file or database and transferred to the translation service, which

returns translated text elements as described previously. Commands for the translation service may include a desired language. The translation service may be a commercially available translation service or a translation service using an application at the data processing system **200** shown in **FIG. 2** and as described below.

[**0127**] After translation, the web-based tool or the XML editor merges the translated text elements received from the translation service back into the XML file or the data base (Step **703**). Accordingly, a collection of different text elements, each of the text elements in different languages, may be provided in the XML file or database. Thereafter, the graphic application program (e.g., GIMP) as discussed above generates immutable image files based on the translated image text elements as described previously (Step **704**).

[**0128**] **FIG. 8** depicts a flowchart depicting exemplary options of steps for a method for generating immutable image files in a number of languages. The steps of **FIG. 8** may be carried out using the system shown in **FIG. 2**; however, **FIG. 8** is not limited thereto.

[**0129**] First, in one option, text elements in a first language are input (Step **801**). A user may generate the text elements manually. Alternatively, a parser may obtain an image creation file (e.g., an HTML or XML file) (Step **802**). Next, the parser extracts text elements in a first language from the image creation file (Step **803**). Then, the parser generates a text element file (e.g., text element file **228** as discussed above) with the text elements (Step **804**). After generating the text element file **228**, the parser transfers the text elements from the text element file into a database (Step **805**).

[**0130**] Next, a translator module (e.g., translator module **218**) invokes a translation service (e.g., the system associated with server **260**) to translate the text elements into a second language (Step **806**). As a first part of Step **806**, the translator module **218** invokes a translation service for translating the text elements in the first language into text elements in the second language (Step **806***a*). As a second part of Step **806**, the translation module **278** then inserts the text elements in the second language into the database (Step **806***b*). An example of a technique for translation is given in more detail below.

[**0131**] Next, the translator module **218** obtains the translated text elements and merges the text elements in the second language into the text element file **228** as discussed above (Step **807**). Thus, the text element file **228** now includes text elements in the first language and corresponding text elements in the second language.

[**0132**] A parser then obtains display characteristics of the image (e.g., color, shape, dimensions, and position of text within image) from at least one template **230** as discussed above (Step **808**). Next, an operating system (e.g., operating system **250**) obtains at least one image creation script (e.g., script **222**) (Step **809**). A user may create at least one immutable image file script **222** for immutable image files. Alternatively, the script may be generated automatically. According to one example, a script may be generated for each individual text element in each particular language. Further, a script for a number of text elements in one language may be generated or a script for one text element in a number of languages may be generated for use by the operating system in conjunction with an image manipulation program (e.g., image manipulation program **224**).

[**0133**] Next, the script **222** instructs the image manipulation program **224** to generate at least one immutable image file **232** with a text element in the second language as discussed above (Step **810**). The image manipulation program then stores the immutable image file in a location specified by desired location information stored in the text string file **226** as discussed above (Step **811**). The immutable image file **232** may then be used to serve users with different language preferences.

[**0134**] In a further example shown below, the image manipulation program **224** described previously includes the GNU (Gnu's Not UNIX) image manipulation program GIMP.

[**0135**] Creation of immutable image files **232** and **234** is accomplished by a GIMP call, a GIMP input file (e.g., text string file **226**) and a GIMP script (e.g., script **222**). Code examples of a GIMP call, a GIMP input file (file name: .utf8.txt) and a GIMP script are given below. The GIMP call is initiated (e.g., on a command line of an operating system such as Windows or from a batch file by the operating system) and in turn instructs the GIMP to input text string information from the input file and template information from the call itself.

[**0136**] The exemplary GIMP input file shown below (which corresponds to the text string file **226**) includes exemplary desired location information for storing generated immutable image files. The exemplary desired location information is designed so that each file generated for a particular template is assigned a file name with the same general base name, with a distinguishing prefix which includes a language code (e.g., "de" for German, "en_US" for U.S. English). The files are stored in a common directory. This naming and storage convention enables convenient update of image creation programs such as HTML or XML code by simply modifying the prefix of a referenced image file to agree with the language preference of a particular user to enable a browser of the user to access the immutable image files which have been generated for the user's preferred language.

[**0137**] An example of an XML text element file is also shown below. The XML file is an example of a file initially created by a developer when developing a system using immutable image files for display, in order to describe elements of the immutable image file such as text strings which may need to be translated for multilingual users. The XML file as shown has been processed by extracting the original German text strings, translating the strings into other languages, and then merging the translated strings back into the text element file as discussed above. The file may be used to generate a GIMP input file by extracting strings in different languages for insertion into the GIMP input file as discussed above.

[0135] An example of an XML text element file is also shown below. The XML file is an example of a file initially created by a developer when developing a system using immutable image files for display, in order to describe elements of the immutable image file such as text strings which may need to be translated for multilingual users. The XML file as shown has been processed by extracting the original German text strings, translating the strings into other languages, and then merging the translated strings back into the text element file as discussed above. The file may be used to generate a GIMP input file by extracting strings in different languages for insertion into the GIMP input file as discussed above.

Gimp call with template parameters:

gimp -i -s -c -b "(script-fu-so-button-generator-get-file\"c:\\development\\gimp\\translate.utf8.txt\" \"-*-Arial Unicode MS-*-r-*-*-24-*-*-*-p-*-*-*\" 14 '(51 51 102) FALSE 2 0 0 1 TRUE '(192 208 224) TRUE '(51 51 102) '(255 255 255) 0 15 135 1 FALSE 135 16 FALSE FALSE '(0 0 0) 1 1 FALSE 3 FALSE 0 0 32 TRUE)"

GIMP INPUT FILE: (file name: .uft8.txt)

1	d:\web\images\new_user.gif	New Users
1	d:\web\images\en_US.new_user.gif	New Users
3	d:\web\images\pt.new_user.gif	Novo utilizador
33	d:\web\images\fr.new_user.gif	Nouveaux utilisateurs
34	d:\web\images\es.new_user.gif	Nuevo usuario
46	d:\web\images\sv.new_user.gif	Nya användare
48	d:\web\images\pl.new_user.gif	Nowy u¿ytkownik
49	d:\web\images\de.new_user.gif	Neue Benutz

GIMP-SCRIPT

```
; Define the function.

(define
  (script-fu-so-button-generator-get-file-wd inFile inQuitGimp)
```

```
        (script-fu-so-button-generator-get-file inFile "-*-Arial Unicode MS-*-r-*-*-24-*-*-*-p-*-*-*" 20 '(0 0 255) FALSE 0 0 1 1
FALSE '(173 168 145) TRUE '(255 0 0) '(0 255 0) 3 30 114 2 FALSE 120 35 TRUE FALSE '(220 220 220) 3 3 TRUE 3 FALSE 0 0
256 inQuitGimp
        )
)

(define
;   (script-fu-so-button-generator-get-file inFile inQuitGimp)
   (script-fu-so-button-generator-get-file      inFile
                                                inTextFont inTextFontSize inTextColor inTextAntialiasing
inTextOffsetX inTextOffsetY inTextAlignX inTextAlignY
                                        inBackgroundTrans inBackgroundColor
                                        inGradientVisible inGradientColor1 inGradientColor2 inGradientOffsetX
inGradientOffsetY inGradientSizeX inGradientSizeY
                                        inImageAutoAdjustSize inImageSizeX inImageSizeY
                                        inBorderVisible inBorderTrans inBorderColor inBorderSizeX inBorderSizeY
                                        inBumpmapUse inBumpmapBeval inBumpmapPressed
                                                inSavePalette inSaveDitherType inSaveMaxColors
                                        inGimpQuit
        )

        (set! theFile (fopen inFile))

        (set! theData ())
        (set! theIndentList ())
        (set! theChar "X")
        (set! theButtonFile "")
        (set! theButtonText "")
        (while (not (equal? () theChar))
        (set! allspaces TRUE)
        (set! theIndent 0)
        (set! theLine "")

        ; while not \n and not EOL
        (while
                (begin
                        (set! theChar (fread 1 theFile))
        (and    (not (equal? "\n" theChar))
                (not (equal? () theChar))
        )
        )
                ; do
        (cond
                        ; Tab?
                        (
                                (equal? theChar "\t")
                                (set! theChar "")
                                        (gimp-message "Tab gefunden")
                                        (set! theButtonFile theLine)
                                        (set! theLine "")

                        (if (= allspaces TRUE)
                                        (begin
                                        (set! theIndent (+ theIndent ))
                                        )
                                ()
                                )
                )
                        , Space?
        (
                                (equal? theChar " ")
                        (if (= allspaces TRUE)
                                (set! theIndent (+ theIndent 1))
                                ()
                                )
        )
```

```
        (TRUE (set! allspaces FALSE))
        )
        (set! theLine (string-append theLine theChar))
                ;       (gimp-message (string-append "Line. " theLine))
        )
        (gimp-message (string-append "Line  " theLine))
        (set! theButtonText theLine)
        (gimp-message (string-append "File: " theButtonFile))
        (gimp-message (string-append "Text: " theButtonText))

        (if (= allspaces TRUE)
        (set! theLine "")
        ()
    )
        (if (and (equal? () theChar) (equal? "" theLine))
        ()
        (begin
                        (set! theData (cons theLine theData))
                (set! theIndentList (cons theIndent theIndentList))
        )
    )

        (script-fu-so-button-generator    theButtonFile theButtonText inTextFont inTextFontSize inTextColor inTextAntialiasing
inTextOffsetX inTextOffsetY inTextAlignX inTextAlignY
                                        inBackgroundTrans inBackgroundColor
                                        inGradientVisible inGradientColor1 inGradientColor2 inGradientOffsetX
inGradientOffsetY inGradientSizeX inGradientSizeY
                                        inImageAutoAdjustSize inImageSizeX inImageSizeY
                                        inBorderVisible inBorderTrans inBorderColor inBorderSizeX inBorderSizeY
                                        inBumpmapUse inBumpmapBeval inBumpmapPressed
                                        inSavePalette inSaveDitherType inSaveMaxColors
                                        TRUE FALSE
        )
        )

        ; Quit
        (if (= inGimpQuit TRUE)
                (begin
                        ; Quit GIMP
                        (gimp-message "Quit!")
                        (gimp-quit 0)
                )
                ()
        )
)

; ************************************************************************

(define
   (script-fu-so-button-generator
                inFileName
                inText
                inTextFont
                inTextFontSize
                inTextColor
                inTextAntialiasing
                inTextOffsetX
                inTextOffsetY
                inTextAlignX
                inTextAlignY
                inBackgroundTrans
                inBackgroundColor
                inGradientVisible
                inGradientColor1
                inGradientColor2
```

```
                inGradientOffsetX
                inGradientOffsetY
                inGradientSizeX
                inGradientSizeY
                inImageAutoAdjustSize
                inImageSizeX
                inImageSizeY
                inBorderVisible
                inBorderTrans
                inBorderColor
                inBorderSizeX
                inBorderSizeY
                inBumpmapUse
                inBumpmapBeval
                inBumpmapPressed
                inSavePalette
                inSaveDitherType
                inSaveMaxColors
                inBatchMode
                inGimpQuit
    )

; ********** Init **********

    (set! DisplayMessages TRUE)

    ; store old Fore/Background colors
    (set! old-bg (car (gimp-palette-get-background)))
    (set! old-fg (car (gimp-palette-get-foreground)))

    ; print Info
    (if (or (= inBatchMode FALSE) (=DisplayMessages TRUE))
      (begin
        (gimp-message " --- ")
        (if (= inBatchMode TRUE) (gimp-message "Batchmode"))
        (gimp-message (string-append "Filename  : " inFileName))
        (gimp-message (string-append "ButtonText: " inText))
        (gimp-message (string-append "Font      : " inTextFont))
        (gimp-message " --- ")
      )
      ()
    )

    ; Create Image
    (set! theImageWidth inImageSizeX)
    (set! theImageHeight inImageSizeY)
    (set! theImage (car (gimp-image-new theImageWidth theImageHeight RGB)))

; ********** do **********
    , add TextLayer
    (if (= DisplayMessages TRUE) (gimp-message "Add TextLayer"))
    (script-fu-so-add-layer-text theImage inText inTextFont inTextFontSize inTextColor inTextAntialiasing inTextAlignX
inTextAlignY inTextOffsetX inTextOffsetY TRUE)
    (set! theTextLayer(car (gimp-image-get-active-layer theImage)))

    , add BackgroundLayer
    (if (= DisplayMessages TRUE) (gimp-message "Add BackgroundLayer"))
    (script-fu-so-add-layer-background theImage inBackgroundColor inBackgroundTrans TRUE)
    (set! theBackgroundLayer(car (gimp-image-get-active-layer theImage)))

    , add GradientLayer
    (if (= inGradientVisible TRUE)
      (begin
                    (if (= DisplayMessages TRUE) (gimp-message "Add GradientLayer"))
                    (script-fu-so-add-layer-gradient theImage inGradientColor1 inGradientColor2 inGradientSizeX
inGradientSizeY inGradientOffsetX inGradientOffsetY inGradientVisible)
```

```
                    (set! theGradientLayer(car (gimp-image-get-active-layer theImage)))
    )
    ()
  )

  ; add BorderLayer
  (if (= inBorderVisible TRUE)
    (begin
      (if (= DisplayMessages TRUE) (gimp-message "Add BorderLayer"))
      (script-fu-so-add-layer-border theImage inBorderColor inBorderSizeX inBorderSizeY inBorderTrans)
      (set! theBorderLayer(car (gimp-image-get-active-layer theImage)))
    )
    ()
  )

  ; set TextLayer to Top
  (gimp-image-raise-layer-to-top theImage theTextLayer)

  ; merge all visible Layers to FinalLayer
  (if (= DisplayMessages TRUE) (gimp-message "Merge Layers"))
  (set! theFinalLayer (car (gimp-image-merge-visible-layers theImage CLIP-TO-IMAGE)))

  ; use Bumpmap on FinalLayer
  (if (= inBumpmapUse TRUE)
    (begin
      (if (= DisplayMessages TRUE) (gimp-message "Apply Bumpmap"))
      (gimp-selection-all theImage)
      (script-fu-so-apply-bumpmap theImage theFinalLayer inBumpmapBeval inBumpmapPressed)
      (gimp-selection-none theImage)
    )
    ()
  )

  ; save Image
  (if (= DisplayMessages TRUE) (gimp-message (string-append "Save Image: " inFileName)))
  (script-fu-so-image-save-gif theImage inFileName inSavePalette inSaveDitherType inSaveMaxColors)


; ********** Cleanup **********
  ; restore old colors
  (gimp-palette-set-background old-bg)
  (gimp-palette-set-foreground old- fg)

  ; delete Dirty-flag
  (if (= DisplayMessages TRUE) (gimp-message "Clean Image"))
  (gimp-image-clean-all theImage)

  ; print Info
  (if (= inBatchMode TRUE)
    ()
    (begin
      ; Display picture
      (gimp-display-new theImage)
    )
  )

  , print Info
  (if (= inGimpQuit TRUE)
    (begin
      , Quit GIMP
      (if (= DisplayMessages TRUE) (gimp-message "Quit!"))
      (gimp-quit 0)
    )
    ()
  )
)
```

```
; ***************************************************************************

; *************** Bumpmap ***************
(define
   (script-fu-so-apply-bumpmap inImage inLayer inBeval inPressed)

   (set! theImageWidth (car (gimp-image-width inImage)))
   (set! theImageHeight (car (gimp-image-height inImage)))

   (set! theBumpmap (car (gimp-layer-new inImage theImageWidth theImageHeight RGBA_IMAGE "Bumpmap" 100 NORMAL)))

   ; Create bumpmap layer

   ;(gimp-image-add-layer inImage theBumpmap -1)
   ;(gimp-image-lower-layer-to-bottom inImage theBumpmapLayer)
   (gimp-palette-set-foreground '(0 0 0))
   (gimp-palette-set-background '(255 255 255))
   (gimp-edit-fill theBumpmap BG-IMAGE-FILL)

   (gimp-rect-select inImage 0 0 inBeval theImageHeight REPLACE FALSE 0)
   (blend-bumpmap inImage theBumpmap 0 0 (- inBeval 1) 0)

   (gimp-rect-select inImage 0 0 theImageWidth inBeval REPLACE FALSE 0)
   (blend-bumpmap inImage theBumpmap 0 0 0 (- inBeval 1))

   (gimp-rect-select inImage (- theImageWidth inBeval) 0 inBeval theImageHeight REPLACE FALSE 0)
   (blend-bumpmap inImage theBumpmap (- theImageWidth 1) 0 (- theImageWidth inBeval) 0)

   (gimp-rect-select inImage 0 (- theImageHeight inBeval) theImageWidth inBeval REPLACE FALSE 0)
   (blend-bumpmap inImage theBumpmap 0 (- theImageHeight 1) 0 (- theImageHeight inBeval))

   (gimp-selection-none inImage)

   (plug-in-bump-map 1 inImage inLayer theBumpmap 135 45 inBeval 0 0 0 TRUE inPressed 0)

   ;(gimp-image-set-active-layer inImage theBumpmap)

)

(define (blend-bumpmap img drawable x1 y1 x2 y2)
 (gimp-blend drawable
             FG-BG-RGB
             DARKEN-ONLY
             LINEAR
             100
             0
             REPEAT-NONE
             FALSE
             0
             0
             x1
             y1
             x2
             y2))

; *************** Background ***************
(define
   (script-fu-so-add-layer-background inImage inColor inTrans inVisible)

   (set! theWidth (car (gimp-image-width inImage)))
   (set! theHeight (car (gimp-image-height inImage)))

   ; create Backgroundlayer
```

```
  (set! theLayer (car (gimp-layer-new inImage theWidth theHeight RGBA_IMAGE "Background" 100 NORMAL)))
  (gimp-palette-set-background inColor)
  (gimp-image-add-layer inImage theLayer -1)

  ; set Background to Transparent?
  (gimp-selection-all inImage)
  (if (= inTrans TRUE)
    ; Background is transparent
    (begin
      (gimp-edit-clear theLayer)
    )
    ; Background is solid
    (gimp-edit-fill theLayer BG-IMAGE-FILL)
  )
  (gimp-selection-none inImage)

  ; turn off Backgroundlayer
  (if (= inVisible TRUE)
    ()
    (begin
      (gimp-layer-set-visible theLayer 0)
    )
  )
  (gimp-image-set-active-layer inImage theLayer)
)


; ********** Gradient **********
(define
  (script-fu-so-add-layer-gradient inImage inColor1 inColor2 inWidth inHeight inOffsetX inOffsetY inVisible)

  ; create Gradientlayer
  (set! theLayer (car (gimp-layer-new inImage inWidth inHeight RGBA_IMAGE "Gradient" 100 NORMAL)))

  ; create Colorblendlayer
  (gimp-palette-set-foreground inColor1)
  (gimp-palette-set-background inColor2)
  (gimp-blend theLayer
      FG-BG-RGB
      NORMAL
      LINEAR
      100
      0
      REPEAT-NONE
      FALSE
      0
      0
      0
      0
      (- inWidth 1)
      0
  )
  ; add Colorblendlayer
  (gimp-image-add-layer inImage theLayer -1)
  (gimp-layer-set-offsets theLayer inOffsetX inOffsetY)

  ; turn off Colorblendlayer?
  (if (= inVisible TRUE)
    ()
    (begin
      (gimp-layer-set-visible theLayer 0)
    )
  )
  (gimp-image-set-active-layer inImage theLayer)
)
```

```
; ************** Text ***************
(define
  (script-fu-so-add-layer-text inImage inText inFont inFontSize inColor inAntialiasing inAlignX inAlignY inOffsetX inOffsetY
inVisible)

  , create Textlayer
  (gimp-palette-set-foreground inColor)
  (set! theLayer (car (gimp-text-fontname inImage -1 inOffsetX inOffsetY inText -1 inAntialiasing inFontSize PIXELS inFont)))

  (set! theImageWidth (car (gimp-image-width inImage)))
  (set! theImageHeight (car (gimp-image-height inImage)))

  (set! theLayerWidth (car (gimp-drawable-width theLayer)))
  (set! theLayerHeight (car (gimp-drawable-height theLayer)))

  ; Text align
  (set! TOP 0)
  (set! LEFT 0)
  (set! CENTER 1)
  (set! BOTTOM 2)
  (set! RIGHT 2)

  (if (and (>= inAlignX 0) (<= inAlignX 2))
    (begin
      (if (= inAlignX LEFT)   (set! theAlignOffsetX 0))
      (if (= inAlignX CENTER)
        (begin
          (set! theAlignOffsetX (/ (- theImageWidth theLayerWidth) 2))
          ; correct 6 empty pixel at end of layer
          (if (= inAntialiasing FALSE) (set! theAlignOffsetX (+ theAlignOffsetX 3)))
        )
        ()
                                )
      (if (= inAlignX RIGHT)  (set! theAlignOffsetX (- theImageWidth theLayerWidth)))
    )
    (begin
      (gimp-message "Warning: TextAlignX out of bound! Align is set to LEFT.")
      (set! theAlignOffsetX 0)
    )
  )
  (if (and (>= inAlignY 0) (<= inAlignY 2))
    (begin
      (if (= inAlignY TOP)    (set! theAlignOffsetY 0))
      (if (= inAlignY CENTER) (set! theAlignOffsetY (/ (- theImageHeight theLayerHeight) 2)))
      (if (= inAlignY BOTTOM) (set! theAlignOffsetY (- theImageHeight theLayerHeight)))
    )
    (begin
      (gimp-message "Warning: TextAlignY out of bound! Align is set to TOP ")
      (set! theAlignOffsetY 0)
    )
  )

  (gimp-layer-set-offsets theLayer (+ inOffsetX theAlignOffsetX) (+ inOffsetY theAlignOffsetY) )

  ; turn off Backgroundlayer n
  (if (= inVisible TRUE)
    ()
    (begin
      (gimp-layer-set-visible theLayer 0)
    )
  )
  (gimp-image-set-active-layer inImage theLayer)
)

; ************** Border ***************
(define
```

```
  (script-fu-so-add-layer-border inImage inColor inSizeX inSizeY inTrans)

  (set! theImageWidth (car (gimp-image-width inImage)))
  (set! theImageHeight (car (gimp-image-height inImage)))

  (set! theImageWidth (+ (* 2 inSizeX) theImageWidth))
  (set! theImageHeight (+ (* 2 inSizeY) theImageHeight))

  (gimp-image-resize inImage theImageWidth theImageHeight inSizeX inSizeY)

   ; add Border
  (set! theLayer (car (gimp-layer-new inImage theImageWidth theImageHeight RGBA_IMAGE "Border" 100 NORMAL) ) )
  (gimp-palette-set-background inColor)
  (gimp-image-add-layer inImage theLayer -1)
  (if (= inTrans TRUE)
     (begin
        (gimp-edit-fill theLayer TRANS-IMAGE-FILL)
     )
     (begin
        (gimp-edit-fill theLayer BG-IMAGE-FILL)
        (gimp-rect-select inImage inSizeX inSizeY (- theImageWidth (* 2 inSizeX)) (- theImageHeight (* 2 inSizeY)) REPLACE
FALSE 0)
        (gimp-edit-clear theLayer)
        (gimp-edit-fill theLayer TRANS-IMAGE-FILL)
        ,(gimp-layer-set-visible theLayer 0)
     )
  )
  (gimp-selection-none inImage)

  (gimp-image-set-active-layer inImage theLayer)
)

; ********** Save **********
(define
  (script-fu-so-image-save-gif inImage inFile inPalette inDitherType inMaxColors)

  ; Convert picture and save as  GIF
;  (gimp-image-merge-visible-layers inImage 1)
  ;(gimp-image-flatten theImage)


  (gimp-convert-indexed inImage inDitherType inPalette inMaxColors FALSE TRUE "")
  (file-gif-save 1 inImage 0 inFile "" FALSE FALSE 0 0)
)

; ********** Register the function with the GIMP **********

(script-fu-register
 "script-fu-so-button-generator"
 _"<Toolbox>/Xtns/Script-Fu/SO/so-Button"
 ""
 "Dietmar Cordes (DC)"
 "21.11 2000"
 "Copyright StarOffice GmbH"
 ""
 SF-FILENAME  _"Filename"                "FileName gif"              , Output Filename
 SF-STRING    "Text"                     "ButtonText"                , Text
 SF-FONT      _"Font"                    "-*-Arial Unicode MS-*-r-*-*-24-*-*-*-p-*-*-*"    ; Font
 SF-ADJUSTMENT _"Fontsize (pixels)"      '(16 2 1000 1 10 0 1)             ; Font size in PIXEL
 SF-COLOR     _"Textcolor"               '(0 0 255)                  ; Textcolor '(RGB)
 SF-TOGGLE    _"Antialising ON/OFF"      TRUE                        ; Antialiasing for Text (ON)/OFF
 SF-VALUE     "Textposition (offset) X"  "0"                         ; Textposition Offset in PIXEL X
 SF-VALUE     "Textposition (offset) Y"  "0"                         ; Textposition Offset in PIXEL Y
 SF-ADJUSTMENT _"Align X: LEFT(0), CENTER(1), RIGHT(2)" '(1 0 2 1 10 0 1)              ;Textalignment
LEFT/(CENTER)/(RIGHT)
```

```
SF-ADJUSTMENT _"Align Y: TOP(0), CENTER(1), BOTTOM(2)" '(1 0 2 1 10 0 1)              ; Textalignment
TOP/(CENTER)/(BOTTOM)
SF-TOGGLE    _"Background Transparent ON/OFF"      FALSE                    ; Background Transparent (ON)/OFF
SF-COLOR     _"Background Color"                   '(173 168 145)           ; Backgroundcolor '(RGB)
SF-TOGGLE    _"Gradient ON/OFF"                    TRUE                     ; Blend Color1 and Color2 (ON)/OFF
SF-COLOR     _"Gradient Color 1"                   '(255 0 0)               ; Color1 '(RGB)
SF-COLOR     _"Gradient Color 2"   '(0 255 0)                               ; Color2 '(RGB)
SF-VALUE     "Gradient Position (offset) X"        "3"                      ; Gradientposition Offset in PIXEL X
SF-VALUE     "Gradient Position (offset) Y"        "25"                     ; Gradientposition Offset in PIXEL Y
SF-ADJUSTMENT _"Gradient Width X (pixels)"         '(94 1 1000 1 10 0 1)    ; Width of Gradient
SF-ADJUSTMENT _"Gradient Height Y (pixels)"        '(2 1 1000 1 10 0 1)     ; Height of Gradient
SF-TOGGLE    _"Buttonsize Auto ON/OFF"             FALSE                    ; autoadjust Buttonsize for Text (ON)/OFF
SF-ADJUSTMENT _"Buttonwidth"                       '(100 1 1000 1 10 0 1)   ; Buttonwidth in PIXEL (if not autoadjust)
SF-ADJUSTMENT _"Buttonheight"                      '(30 1 1000 1 10 0 1)    ; Buttonheight in PIXEL (if not autoadjust)
SF-TOGGLE    _"Border ON/OFF"                      FALSE                    ; Border ON/(OFF)
SF-TOGGLE    _"Border Transparent ON/OFF"          FALSE                    ; Border is Transparent
SF-COLOR     _"Border Color"                       '(0 0 0)                 ; Border Color
SF-ADJUSTMENT _"Border Size X (pixels)"            '(1 1 1000 1 10 0 1)     ; Border Size in PIXEL X
SF-ADJUSTMENT _"Border Size Y (pixels)"            '(1 1 1000 1 10 0 1)     ; Border Size in PIXEL Y
SF-TOGGLE    _"Bump ON/OFF"                        TRUE                     ; Bump (ON)/OFF
SF-ADJUSTMENT _"Bumpsize (pixels)"                 '(3 1 1000 1 10 0 1)     ; Bumpsize
SF-TOGGLE    _"Button pressed ON/OFF"              FALSE                    ; Button pressed ON/(OFF)
SF-ADJUSTMENT _"Gif Palette  MAKE(0), WEB(2), MONO(3),"                                                    '(0 0 3 1
10 0 1)  , Gif Palette
SF-ADJUSTMENT _"Gif DitherType: NONE(0), FS(1), FS/LowBleed(2), FIXED(3)"   '(0 0 3 1 10 0 1)  ; Use Dithering for Palette
SF-ADJUSTMENT _"Gif Maximum Colors used"           '(256 1 256 1 10 0 1)    ; MaxColors in Palette
SF-TOGGLE    _"Run in Batch Mode"                  FALSE                    ; no GUI in Batch mode
SF-TOGGLE    _"Quit Gimp"                          FALSE                    ; Quit Gimp
)

(script-fu-register
 "script-fu-so-button-generator-get-file"
 _"<Toolbox>/Xtns/Script-Fu/SO/so-Button GetFile"
 ""
 "Dietmar Cordes (DC)"
 "21.11 2000"
 "Copyright StarOffice GmbH"
 ""
 SF-FILENAME  _"Filename"                  "c:\\testlang\\TestForButton.txt"    ; Input-File
 SF-FONT      _"Font"                      "-*-Arial Unicode MS-*-r-*-*-24-*-*-*-p-*-*-*"   ; Font
 SF-ADJUSTMENT _"Fontsize (pixels)"        '(16 2 1000 1 10 0 1)    ; Font size in PIXEL
 SF-COLOR     _"Textcolor"                 '(0 0 255)               ; Textcolor '(RGB)
 SF-TOGGLE    _"Antialising ON/OFF"        TRUE                     ; Antialiasing for Text (ON)/OFF
 SF-VALUE     "Textposition (offset) X"    "0"                      , Textposition offset in PIXEL X
 SF-VALUE     "Textposition (offset) Y"    "0"                      ; Textposition offset in PIXEL Y
 SF-ADJUSTMENT _"Align X: LEFT(0), CENTER(1), RIGHT(2)" '(1 0 2 1 10 0 1)          ; Textalignment
LEFT/(CENTER)/(RIGHT)
 SF-ADJUSTMENT _"Align Y: TOP(0), CENTER(1), BOTTOM(2)" '(1 0 2 1 10 0 1)          , Textalignment
TOP/(CENTER)/(BOTTOM)
 SF-TOGGLE    _"Background Transparent ON/OFF"      FALSE                    ; Background Transparent (ON)/OFF
 SF-COLOR     _"Background Color"                   '(173 168 145)           ; Backgroundcolor '(RGB)
 SF-TOGGLE    _"Gradient ON/OFF"                    TRUE                     ; Blend Color1 and Color2 (ON)/OFF
 SF-COLOR     _"Gradient Color 1"                   '(255 0 0)               ; Color1 '(RGB)
 SF-COLOR     _"Gradient Color 2"   '(0 255 0)                               , Color2 '(RGB)
 SF-VALUE     "Gradient Position (offset) X"        "3"                      , Gradientposition offset in PIXEL X
 SF-VALUE     "Gradient Position (offset) Y"        "25"                     ; Gradientposition offset in PIXEL Y
 SF-ADJUSTMENT _"Gradientwidth X (pixels)"          '(94 1 1000 1 10 0 1)    , Width of Gradient
 SF-ADJUSTMENT _"Gradientheight Y (pixels)"         '(2 1 1000 1 10 0 1)     ; Height of Gradient
 SF-TOGGLE    _"Buttonsize Auto ON/OFF"             FALSE                    ; autoadjust Buttonsize for Text (ON)/OFF
 SF-ADJUSTMENT _"Buttonwidth"                       '(100 1 1000 1 10 0 1)   ; Buttonwidth in PIXEL (if not autoadjust)
 SF-ADJUSTMENT _"Buttonheight"                      '(30 1 1000 1 10 0 1)    ; Buttonheight in PIXEL (if not autoadjust)
 SF-TOGGLE    _"Border ON/OFF"                      FALSE                    ; Border ON/(OFF)
 SF-TOGGLE    _"Border Transparent ON/OFF"          FALSE                    , Border is Transparent
 SF-COLOR     _"Border Color"                       '(0 0 0)                 ; Border Color
 SF-ADJUSTMENT _"Border Size X (pixels)"            '(1 1 1000 1 10 0 1)     , Border Size in PIXEL X
 SF-ADJUSTMENT _"Border Size Y (pixels)"            '(1 1 1000 1 10 0 1)     ; Border Size in PIXEL Y
```

```
SF-TOGGLE    _"Bump ON/OFF"                TRUE                          ; Bump (ON)/OFF
SF-ADJUSTMENT _"Bumpsize (pixels)"         '(3 1 1000 1 10 0 1)          ; Bumpsize
SF-TOGGLE    _"Button pressed ON/OFF"      FALSE                         ; Button pressed ON/(OFF)
SF-ADJUSTMENT _"Gif Palette: MAKE(0), WEB(2), MONO(3),"                                          '(0 0 3 1
10 0 1)  ; Gif Palette
SF-ADJUSTMENT _"Gif DitherType  NONE(0), FS(1), FS/LowBleed(2), FIXED(3)"   '(0 0 3 1 10 0 1)  ; Use Dithering for Palette
SF-ADJUSTMENT _"Gif Maximum Colors used"   '(256 1 256 1 10 0 1)         ; MaxColors in Palette
SF-TOGGLE    _"Quit Gimp"                  FALSE                         ; Quit Gimp
)

(script-fu-register
 "script-fu-so-button-generator-get-file-wd"
 _"<Toolbox>/Xtns/Script-Fu/SO/so-Button GetFile (defaults)"
 ""
 "Dietmar Cordes (DC)"
 "21.11.2000"
 "Copyright StarOffice GmbH"
 ""
 SF-FILENAME  _"Filename"                  "c:\\testlang\\TestForButton.txt"          , Filename
 SF-TOGGLE    _"Quit Gimp"                 FALSE                         , Quit Gimp n
)
```

XML text element file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE OutputDir PUBLIC '-//StarOffice//DTD ResourceBundle 1 0//EN' 'XMLGIF DTD'>
<OutputDir path="WebApplication portal.web html.pics">
   <Template name="caption">
      <OutputFile name="upload_nav.gif">
           <Text xml:lang="de">Hochladen |</Text>
           <Text xml:lang="x-translate">Upload |</Text>
      </OutputFile>
      <OutputFile name="paste_nav.gif">
           <Text xml:lang="de">Einfügen</Text>
           <Text xml lang="x-translate">Paste</Text>
      </OutputFile>
      <OutputFile name="properties_nav.gif">
           <Text xml:lang="de">Properties |</Text>
           <Text xml lang="x-translate">Properties |</Text>
      </OutputFile>
   </Template>
          <Template name="clipboard">
                    <OutputFile name="fax_clip gif">
                              <Text xml lang="de">Fax</Text>
                              <Text xml lang="x-translate">Fax</Text>
                              <Text xml:lang="en-US">Fax</Text>
                              <Text xml:lang="pt">Fax</Text>
                              <Text xml lang="nl">Fax</Text>
                              <Text xml lang="fr">Fax</Text>
                              <Text xml lang="es">Fax</Text>
                              <Text xml lang="it">Fax</Text>
                              <Text xml:lang="da">Fax</Text>
                              <Text xml:lang="sv">Fax</Text>
                              <Text xml.lang="pl">Faks</Text>
                              <Text xml:lang="pt-BR">Fax</Text>
                              <Text xml.lang="tr">Fax</Text>
                    </OutputFile>
                    <OutputFile name="mail_clip gif">
                              <Text xml lang="de">Nachricht</Text>
                              <Text xml lang="x-translate">Mail</Text>
                              <Text xml:lang="en-US">Message</Text>
                              <Text xml:lang="pt">Mensagem</Text>
                              <Text xml:lang="nl">Bericht</Text>
                              <Text xml lang="fr">Message</Text>
```

```
                    <Text xml:lang="es">Mensaje</Text>
                    <Text xml:lang="it">Messaggio</Text>
                    <Text xml:lang="da">Meddelelse</Text>
                    <Text xml lang="sv">Meddelande</Text>
                    <Text xml.lang="pl">Wiadomosc</Text>
                    <Text xml.lang="pt-BR">Message</Text>
                    <Text xml:lang="tr">Message</Text>
                </OutputFile>
                <OutputFile name="print_clip gif">
                    <Text xml.lang="de">Drucken</Text>
                    <Text xml lang="x-translate">Print</Text>
                    <Text xml:lang="en-US">Print</Text>
                    <Text xml:lang="pt">Imprimir</Text>
                    <Text xml:lang="nl">Afdrukken</Text>
                    <Text xml lang="fr">Imprimer</Text>
                    <Text xml:lang="es">Imprimir</Text>
                    <Text xml:lang="it">Stampa</Text>
                    <Text xml:lang="da">Udskriv</Text>
                    <Text xml lang="sv">Skriv ut</Text>
                    <Text xml lang="pl">Drukuj</Text>
                    <Text xml:lang="pt-BR">Print</Text>
                    <Text xml:lang="tr">Print</Text>
                </OutputFile>
            </Template>
</OutputDir>
```

TRANSLATION OF TEXT

[0136] A technique for translating text strings is discussed below.

[0137] The discussion which follows relates to database application methods and programs for localizing software, translating texts contained therein and to support translators in localizing software and translating texts. It also relates to a translation system for localizing software, translating texts and supporting translators.

[0138] Conventional translation programs attempt to translate texts on a "sentence by sentence" basis. Such translation programs encompass the analysis of the grammatical structure of a sentence to be translated and the transfer of it into a grammatical structure of a target language sentence, by searching for grammar patterns in the sentence to be translated. The text to be translated may also be called the source language text. Translation is usually accomplished by analyzing the syntax of the sentence by searching

Translation of Text

[0138] A technique for translating text strings is discussed below.

[0139] The discussion which follows relates to database application methods and programs for localizing software, translating texts contained therein and to support translators in localizing software and translating texts. It also relates to a translation system for localizing software, translating texts and supporting translators.

[0140] Conventional translation programs attempt to translate texts on a "sentence by sentence" basis. Such translation programs encompass the analysis of the grammatical structure of a sentence to be translated and the transfer of it into a grammatical structure of a target language sentence, by searching for grammar patterns in the sentence to be translated. The text to be translated may also be called the source language text. Translation is usually accomplished by analyzing the syntax of the sentence by searching for main and subordinate clauses. For this purpose the individual words of the sentence need to be analyzed for their attributes (e.g., the part of speech, declination, plurality, and case). Further, conventional translation programs attempt to transform the grammatical form from the source language grammar to the target language grammar, to translate the individual words and insert them into the transformed grammatical form. If the translation program works correctly, the translated sentence exactly represents or matches the sentence to be translated. One difficulty is that the translation of many words in the text to be translated is equivocal and represents many different words in the target language. Therefore, as long as translation programs cannot use thesauruses specifically adapted for detecting the correct semantic expressions and meaning of sentences, according to the context, machine translations are imperfect.

[0141] Another drawback of the translation programs discussed above is that slang and slang grammar is not translatable, as slang uses grammar rules different from the standard grammar rules. Additionally the complex grammar rules are difficult to describe with mathematical laws and therefore require huge calculation costs even for simple sentences. Additionally, ambiguous source language sentences are especially difficult to translate into target sentences with the same ambiguous grammar structure.

[0142] Another drawback of the conventional translation programs is the fact that the translation performance is limited, with the exception of update options for the dictionaries and the grammar rules. Conventional translation programs are not adaptive, and repeat the same translation errors, as the user cannot change the grammar rules by himself/herself without risking a total breakdown of the translation apparatus.

[0143] Another problem is connected with software localization (i.e., the adaptation of software to a certain local, geographic or linguistic environment in which it is used). Localization must be done with internationally available software tools in order to ensure proper functionality with users from different cultural and linguistic backgrounds. This problem is also related to the fact that different translators work simultaneously on different aspects of software. So, for example, one team translates the text entries in the source code of software, another team translates the manuals, while a third team translates the "Help" files. To achieve a correct localized version of the software it is beneficial that the three teams use the same translations for common words so that the manuals, "Help" files, and the text elements in the software itself consistently match each other. To achieve this, the teams conventionally have to establish the keywords and the respective translations used in the software. This is time consuming and requires good communication among the teams.

[0144] An exemplary method described below localizes software by adapting the language of texts in source code of the software by extracting text elements from the source code. Next, untranslated text elements are searched in the extracted text elements. Translation elements are retrieved for each untranslated text element (e.g., by searching a database with pre-stored matching source text elements and respective translations). The next step is related to associating the searched untranslated element of the source language with the retrieved translation elements, if such translation elements are found.

[0145] For this example, the method supports a software localizer (or user) by offering one or more possible translations of a text element contained in the source code of a software program. In this way the user may simply perceive the translation proposals retrieved from a database. The method ensures the adaptability of different software to certain cultural, geographic or linguistic environments.

[0146] The method further includes the steps of validating the retrieved translation elements, and merging the source code with the validated translation elements. With these steps the user can easily decide if he/she wants to accept (validate) or discard the suggested translation. The validation of the suggested translation may be executed automatically, if the user has validated the same translation element in another section of the source code. Merging the source code with the validated translation elements indicates the enlargement of the collection of text elements for one language in the source code with the validated other language(s) translation elements. The merging may be executed after translation of the whole text.

[0147] The method further includes a compiling step to compile the merged source code to generate a localized software program in binary code. This software program preferably includes one or more text libraries in languages different from the language of the original source code. Compilation may be the last step in the generation of a localized software version, depending on the programming language. An additional step of binding may be executed when the programming language requires the binding of sub-modules or sub-libraries after compilation. In this case compilation is followed by one or more binding steps. Depending on the programming language, the compilation may include an assembling operation.

[0148] Further, the method includes an adaptation of the language of texts related to the software (e.g., manuals and Help files) for illustrating the use of the software and supporting the user accordingly. Software is often provided with additional texts such as "Help " files, manuals, and packaging prints. Such texts are provided with translations matching the translations used in the source code of the software. The translation of software related texts may be executed by the additional steps of searching untranslated

elements of the software related text, retrieving translation elements for each untranslated text element of the software related text on the basis and in accordance with the association of the retrieved source code translation elements, and associating the untranslated elements of the software related text with the retrieved translation elements. Thereby it is ensured that a consistent translation is performed for both the translation of source code texts and software related texts.

[0149] The step of retrieving translation elements for each untranslated text element of the software related text on the basis of the association of the retrieved source code translation elements may include previous validation, merging or compilation of the associated source code translation elements, with a preference on the validated entries.

[0150] The method may be executed in a local area network (LAN), wherein different teams of translators access and/or retrieve the translation suggestion from the same table and/or dictionary and/or library and/or storage unit. To simplify the access and to standardize the translation of a software, the storage unit may be related to a single language software localization (i.e., there is one translation sub-dictionary for each language in the localized software version). This feature may be combined with a composed translation dictionary structure, so that a number of keywords are stored in and retrieved from a localization specific database, by which keywords of minimum requirements are defined. Text elements not containing the keywords may be retrieved from a common database.

[0151] Another exemplary method is provided by which—dependent on or irrespective from the translation of software source code—text elements are translated. The method utilizes text elements of the software related text and their respective translations stored in a database. First, all untranslated text elements are searched in the software related text. Next, translation elements are retrieved for an untranslated text element (e.g., by searching a database with pre-stored matching source text elements and respective translations). Associating the searched untranslated element of the source language with the retrieved translation elements is done, if and after such translation elements are found.

[0152] In this example the method supports a translator by offering one or more possible translations of a source text element. The user or translator may easily decide if he wants to accept or discard the suggested translation. The main difference with conventional translation programs is that the grammar of the source text element is not checked. This is not necessary, as only the text structure of the source language has to match a previously translated text element. It is similar to using an analphabet (e.g., an illiterate) for comparing a pattern (the source text element) with previously stored patterns, in a card index and attaching a respective file card (with the translation element) at the place where the pattern was found in the source text. Then, a translator may accept or discard the index card, instead of translating each text element by himself, thereby improving his translation speed. The performance of the pretranslation is therefore only dependent on the retrieval speed of the file card and on the size of the card index, and not on the linguistic abilities of the user performing the translation. In another example two or more users may sweep the card index at the same time. Additionally, in another example, the users can take newly written index cards from another user and sort them into the card index.

[0153] Further, the associating is performed by entering the retrieved translation elements into a table. The use of tables simplifies the processing of the text elements and enables the developers to use database management systems (DBMS) to realize the example in a computer. The use of tables further simplifies the access of retrieved translation elements, if they are retrieved from a database.

[0154] The use and the interaction of tables in a database are well known and easily implemented tasks in databases, especially if the tables include pre-defined translated phrases, compared to an ambiguous grammar analysis. The transfer to tables of a database offers additional benefits, as comments, markings and translation related background information, such as language code and translation code may be stored in additional columns in the table. With the "patterns" stored in columns the job of users performing the translation in the previous example is simplified by taking the next pattern in the column.

[0155] The user may accept or discard entries in the pretranslation table by interaction. The interaction with a user has various advantages. First, a user may decide for himself if he wants to accept the proposed entry. Second, if multiple entries are proposed the user can select the best one, or may select one of the proposed lower rated translation suggestions and adapt it to the present text to be translated. The user may choose whether he wants to select one of the proposed translation elements in the pre-translation table, or whether he wants to translate the text element manually.

[0156] The translation method may include updating the dictionary database with the validated translation element. By updating the database with new text elements and translation entries the database is extended and improved. This extension may be executed automatically (e.g., by automatically updating the dictionary database with the validation of a translation element). The updating of the database may be executed according to different procedures. For example, the database update may be executed by transferring all validated translation elements from the user device to the database, so that the database itself may determine whether that validated translation element is already stored, or is to be stored as a new translation element. As another example, the database may be updated by transferring only newly generated database entries to the database. It may be advantageous that a user, a user device, or the database checks whether a validated text element/translation element pair is already present in the database, to prevent a large number of similar or identical entries, which is preferably avoided. An additional validation process initiated from a second translator may be performed to update the translation database to prevent invalid translation elements from being dragged into the translation database. The second validation may be easily be implemented, as the translator for the second validation has only to determine whether a translation element matches the respective text element or not. It may also be possible to discard certain translation proposals as invalid by a user input, to prevent the system from offering this translation element a second time. This feature may be embodied as a "negative key ID"-entry in a predetermined column of the translation table.

[0157] An index may be generated in an additional step indicating the grade of conformity of an element of the text and of the dictionary database. This index is useful, if the text element to be translated and the text entries stored in the database differ only slightly, so that a translator may easily derive a correct translation from a slightly different database entry. The depiction of an index describing the matching of an untranslated text element and a translated text element may be described as fuzzy matches. The generated index may be described as a "fuzziness" index of the matching element.

[0158] The number of entries in the pretranslation table of one text element is related to the grade of matching. The number of matching translations to be retrieved from the translation database has not been limited in the preceding discussion, so that more than one matching entry of the database may be retrieved. If more than one entry may be found, the number of retrieved entries may be limited to a predetermined value of the matching grade index. Alternatively, the number of retrieved database entries may be limited by a predetermined value. Both limits avoid the possibility that a human translator has to process a large number of translation proposals. The number of translation proposals may be limited by a predetermined value (e.g., of the matching index). If the predetermined matching index is set to zero, the user in the above example would retrieve the whole database for each source text element in the table.

[0159] A translation entry of the source language text element entry is accepted as a valid translation automatically, if the index indicates an exact match of the untranslated text element with the source language text element entry. This option automates the translation support method described previously to an automatic translator for text elements with exactly matching database entries. This automatic translation method may economize the translator, if the texts to be translated are limited to a small lingual field combined with a very large database. Very large databases suggest a centralized use of the translation method in computer networks such as local area networks or the internet, thus enabling fast generation of a very large universal translation database, if any net user can access the database for translation tasks improving the database with each accepted translation. To prevent abuse of the database, the entries may be marked as authorized by the translators of the provider.

[0160] The method may include marking translated or untranslated or partially translated text elements. The markings may be used to indicate the status of the translation, distinguishing between different grades of translations (e.g., fully, partially, or untranslated). The markings may be incorporated in the text, or may be added in a separate column into the translation table. The markings may be used to indicate whether the specific table entry has been translated by a translator, or is an automatically generated proposal. The markings are especially useful if the front-end used by the translator depicts the table entries as conventional text without a visible table structure. The markings may be colored text, colored background, bold or oblique depiction of the text, bold or colored tables or frames.

[0161] Markings of the table entries may indicate a count of the number of times this database entry has been used or has been accepted by a translator. The markings may indicate the differences between the translation suggestion and the actual text, to simplify the work of the translator. The marking may indicate a single word differing in the database entry and in the translation. The marking may indicate the quality of the suggested translation. The marking may indicate the similarity to pre-translated database entries, so that the user may recognize whether the user selected an unusual grammar or word translation. A marking may be applied to the source text, indicating a grammar structure, so that the main clause is marked with one color and the subordinate clauses in another color. Additionally, the flexion of subordinate clauses to words in the main sentences may be indicated.

[0162] Single text elements may be stored in any logical form in the database. For example, a text may be stored as a tuple of key identification numbers of sentences, and/or the texts and sentences may be stored as a tuple of key identification numbers of words and/or grammar references. The texts, words and/or grammar references may be stored as single texts in the database.

[0163] The various text elements described above may be combined in a single translation method. In this method the database is first searched for text sections, then for sentences and finally for words. The three searches may be executed successively with or without intermediate user interactions. In the first case the user first checks the suggested text sections, then the computer searches for translations of sentences, to be checked by the user and finally, searches for single words to be finally checked by the user. The computer may successively check the three hierarchic levels of text elements, preferably only continuing in a lower hierarchic level if no suggestion has been found in the higher hierarchic level, leading to a mix of suggested text section-, sentence- and word translations. To clarify such a mix of suggestions a front-end of the translation support program may mark the hierarchic level of the suggested translation. This technique may be compared with cutting down a groove: You can first cut all trees, then cut all branches, or you could cut down each tree, cut its branches, cut down the next tree, and so on. Similarly, the translation program may first compare all text sections with the database, then all sentences, and next all words, or may compare a text section, next compare all sentences of the text section, if the text section could not be found in the database, continuing by checking the database for all sentences in that text section, and continuing by searching all words in the database of sentences that could not be found in the database. The search for text sections or for words may be economized. With these combined features a translator can easily utilize a database to generate a pretranslation simply and relieving the user of manually searching paper dictionaries and grammar books for translation.

[0164] The method further includes processing text elements and translated text elements stored in the database. The processing may include a preprocessing technique to transfer an arbitrary source text to a database table, including such steps as converting different text files to a file format used in the database to simplify the retrieval of matching database entries. Similarly, translated text in the database may easily be transformed to an arbitrary file format. The transfer of the text to a table requires splitting the source text into single text elements. The processing may include a post-processing technique (e.g., by applying statistics to

extract translation rules). By means of statistics, by analyzing the behavior pattern of a user, a translation device may be enabled to translate sentences automatically. Thus, the method may include analyzing the user input on particular difference patterns between source text and database entries, enabling generation of translation suggestions assembled from database entries and user input patterns. The method may further include transferring operations texts to and from a database, affording useful options such as simplification of the pretranslation of the source code text, as the source code text may be entered in the format of text elements in a database compatible form such as a table.

[0165] The method may further include sorting the text elements. A sorting operation could be performed prior to or after the generation of translation suggestions. Thus, a user may sort the text elements such as the text elements of the source text into alphabetic order or sort the source text for certain "keywords". A special application of a sorting operation may be sorting the sentences according to grammar structures or the length of sentences or the number of words or characters used in a sentence. By sorting grammar structures, the method may be used to simplify the translation and the use of the method, as the translator starts with simple grammar structures working himself/herself through to the complex grammar structures. This enables a user to practice with simple sentences to prepare for complex sentences. The user may adapt the order of the sorting by transferring a text element to the end of a sequence in the ordering.

[0166] An exemplary software tool includes program code portions for carrying out the steps of the methods described above, when the program is implemented in a computer program. The software tool may be implemented in a database management program (DBMS) to offer the user additional translation features. The software tool may be implemented in a translation program to offer a combination of the features from conventional features and features according to the examples described above. The combination of a conventional translation program and a program according to these examples enable a user to use an intelligent and adaptive translation program, the performance of which increases with the number of translated sentences. A combination of a conventional translation program and an example as described above may utilize an additional grammar filter, to check the source text and the user input to prevent errors in the translation, and to improve the quality of a translation by marking the translated passage in the text with an "equivocal source text"-mark.

[0167] A dictionary database may contain the data of a sentence based dictionary. The database, especially a database containing a sentence based translation dictionary may be applied in translation devices, or to extend the capabilities of an existing conventional translation device.

[0168] The system may include a sentence based dictionary for translation purposes. The sentence based dictionary is different from conventional dictionaries in that the translated units are not words and but whole sentences. Such a dictionary, if printed as a conventional book, may be larger than the "Encyclopedia Britannica" and may appear a bit bulky, but it would require only a single access per sentence, without the need of checking the grammar. The dictionary may be incorporated in an electronic book, simplifying the access, reducing the weight and improving the user interface of the dictionary. The electronic dictionary may include a scanner for simplified source language text input.

[0169] Another example provides a network system for enabling and/or supporting at least one user to localize software by translating texts in the software by means of a database. The network system includes at least one of the translation or software localization apparatuses described previously, and at least one translation database for exchanging data with the at least one apparatus. The software to be localized may be transferred to the apparatuses via the network.

[0170] **FIG. 9***a* depicts a flowchart depicting exemplary steps for translating strings. There are shown the steps of a method for localizing software by changing, adapting or translating the language in a source code of the software (e.g., by translation) wherein the method includes extracting text elements from the source code of the software (step **901**); searching untranslated elements of the extracted text elements (step **903**); retrieving translation elements for each untranslated text element (step **905**); and associating the untranslated elements of the source code with the retrieved translation elements (step **909**).

[0171] In this basic method, the translation elements are retrieved from a database **907**. The associated untranslated elements and the translation elements can then be transferred to a user interface to be displayed on a screen **915**. This basic method for translating texts can be extended to the method described below.

[0172] **FIG. 9***b* shows a block diagram illustrating an exemplary apparatus for localizing software by changing, adapting or translating the language in a source code of software (e.g., by translation).

[0173] The apparatus **920** includes a component for extracting **922** text elements from source code **930** of the software. The component for extracting **922** may include a data processing device or a program module executed at a data processing device. Moreover, the component for extraction **922** may be realized by a code module containing instructions for extracting. The text elements may be extracted from the source code of the software according to the step **901** shown in **FIG. 9***a*.

[0174] Further, the apparatus **920** includes a component for searching **924** untranslated elements of the extracted text elements. The component for searching **924** includes a data processing device or a program module executed at a processing device. Moreover, the component for searching **924** may realized by a code module containing instructions for extracting. The untranslated elements of the extracted text elements may be searched according to step **903** shown in **FIG. 9***a*.

[0175] Further, the apparatus **920** includes a component for retrieving **926** translation elements for each untranslated text element. The component for retrieving **926** may include a data processing device or a program module executed at a processing device. Moreover, the component for retrieving **926** may include a code module containing instructions for extracting. The translation elements are retrieved from a database **907**. The translation elements are retrieved for each untranslated text element according to step **905** shown in **FIG. 9***a*.

[0176] Further, the apparatus **920** includes a component for associating **928** the untranslated elements of the source code with the retrieved translation elements. The component for associating **928** includes a data processing device or a program module executed at a data processing device. Moreover, the component for associating **928** includes a code module containing instructions for extracting. Untranslated elements of the source code are associated with the retrieved translation elements according to step **909** shown in **FIG. 9***a*.

[0177] **FIG. 10** depicts a flowchart of an exemplary software localization and related text translation method. In **FIG. 10** the method is divided by a dotted line separating a branch **1002** to **1014** describing the localization of the software from a branch **1022** to **1032** describing the translation of software related texts. The method depicted in **FIG. 10** starts with a start element **1000**. The start element provides the presence of software source code and/or a software related text. As described in **FIG. 9***a*, both branches start with an extraction step **1002**, **1022** to extract text elements from the source code and/or from software related text. Next, the extracted text elements are searched for untranslated text elements (steps **1004**, **1024**). In the next steps **1006**, **1026** translation elements are retrieved for each untranslated text element from a database **1050**. Both branches may access the same database **1050** to retrieve the translation elements. Accessing the same database **1050** provides the advantage that both translations are based on the same set of keyword translations, thus avoiding confusing results such as the manual description of an icon indicating "Memory" while the same icon in the user interface of the software indicating "Storage". Thus, this method minimizes the occurrence of inconsistencies between the software and the manuals by using the same unambiguous expressions for describing the same notions.

[0178] The steps **1008**, **1028** describe the association of the text elements and the translation elements. As described below, this may include the use of pre-translation tables as depicted in the **FIGS. 15***a*, **15***b*, **16***a* and **16***b*. It is to be noted that the software source code translation and the software related text translation may be executed with the same or with different translation tables. The steps **1010**, **1030** describe a validation of translation elements. The validation may be executed automatically (or manually by user input). If none of the proposed translation elements can be validated, the method requests a user input of a valid translation element, in step **1016**. With the user input of a valid translation, the method can access a new pair of translation elements, that may be utilized in the translation of the other branch. This updating (depicted as the arrow between step **1016** and the database **1050**) results in a growing database **1050**, with an increasing number of entries and an increasing precision of the translation proposals. One possible updating procedure is depicted in **FIG. 13**. It should further be noted that the method of both of the branches may be executed in an interleaving fashion.

[0179] It should further be noted that the temporal distribution of the execution of the steps is not important. Therefore, the steps of the flowchart may be executed by first localizing the software, and then translating the software related texts, wherein the translation of the texts is determined by the translations previously used for the localization. The steps of the flowchart may also be executed by first translating the software related texts, and then localizing the software, wherein the localization of the software is determined by the translation previously used for translating the software related texts. The steps of the flowchart may also be executed by simultaneously translating the software related texts and localizing the software, wherein the first validation of a translation element determines the following translations. This example enables two teams to work simultaneously on localizing software and its manuals, economizing the operations of determining a translation catalogue of keywords.

[0180] If all translation elements are valid, these are merged (in the first branch) with the source code of the software (step **1012**), and the source code of the software is compiled (or assembled) to a localized software version (step **1014**). If all translation elements are valid in the second branch they are exchanged with the related text elements of the software related text (step **1032**). Then the localization of the software is terminated (step **1090**).

[0181] **FIG. 11***a* depicts a flowchart depicting the steps of an exemplary method for translating elements of source language text. The method include searching untranslated elements of the source language text (step **1102**), retrieving translation elements for each untranslated text element (step **1104**), and associating the searched untranslated elements of the source language with the retrieved translation elements (step **1108**).

[0182] In this basic method, the translation elements are retrieved from a storage unit **1106**. The associated untranslated elements and the translation elements are then transferred to a user interface to be displayed on a screen **1110**. This basic method for translating texts can be extended to the method described in **FIG. 12**. To extend this method to a software localizing method, the storage unit **1106** may have been previously provided with different keyword translations from a software localization, and can be used to translate manuals and help files.

[0183] **FIG. 11***b* depicts a block diagram illustrating an example of an apparatus for translating elements of a source language text. The apparatus **1120** includes a component for searching **1122** untranslated elements of the software related text **1130**. The component for searching **1122** may include a data processing device or a program module executed at a data processing device. Moreover, the component for searching **1122** may realized by a code module containing instructions for extracting. Untranslated elements of the source language text may be searched according to step **1102** shown in **FIG. 11***a*.

[0184] Further, the apparatus **1120** includes a component for retrieving **1124** translation elements for each untranslated text element. The component for retrieving **1124** may include a data processing device or a program module executed at a processing device. Moreover, the component for retrieving **1124** may include a code module containing instructions for extracting. Translation elements for each untranslated text element may be retrieved from a database **1106** according to step **1104** shown in **FIG. 11***a*.

[0185] Further, the apparatus **1120** includes a component for associating **1126** the untranslated elements of the source code with the retrieved translation elements. The component for associating **1126** may include a data processing device or

a program module executed at a data processing device. Moreover, the component for associating **1126** may be realized by a code module containing instructions for extracting. The searched untranslated elements of the source language may be associated with the retrieved translation elements according to step **1108** shown in **FIG. 11***a*.

[**0186**] **FIG. 12** depicts a flowchart depicting steps of an exemplary method of translating text. The start box **1210** of the flowchart presumes the presence of a source language text, a database with stored source language text elements and related translated target text elements, and presumes that the source and the target language are already entered. First, a system program or a user selects the type of text element which the present source language text has to be split into (step **1212**). The text element includes any language or text unit (e.g., text sections, sentences, or words). After the splitting of the source text into text elements, the text elements are entered in fields of a translation table (see **FIG. 14**). Then an untranslated source text element is searched (step **1214**), followed by checking the database for a matching source text elements (step **1216**). If a matching text element is not found, the text element remains untranslated and a respective field in the pretranslation table remains empty or is marked with an "untranslatable" mark (step **1224**) Then, the system searches for the next untranslated source text element (step **1222**).

[**0187**] If a matching text element is found in the database, the system copies the related translation to a pretranslation field in the translation table (step **1218**). The text elements stored in the database can be exact matches or fuzzy matches. An exact match results when the source text element and the database text element are identical. A fuzzy match results when the source text element and the database text element only differ in one or a few words or punctuation marks. Fuzzy matches can be marked with an index indicating the fuzziness of the match (i.e., the quality of the match). In both cases the related translations are copied to a pre translation table. In the case of fuzzy matches or in the case of more than one exact match, the pretranslation table entries may be sorted (e.g., for their matching quality) to minimize time spent by a user searching for the best match. More than one translation of a single entry is possible as translations of single words can be equivocal, and so the translation of whole sentences are also equivocal. Next, the method searches for the next untranslated source text element (step **1222**). If a next text element is found, control returns to step **1216**, to loop steps **1216** to **1222**, until a next text element is not found.

[**0188**] If the system does not find a next source text element, the method determines whether the actual text element is the smallest text element available (step **1226**). If the actual text element is not the smallest available, a smaller text element is selected (step **1227**) and the control returns to step **1214**, to loop steps **1216** to **1224**, until no next smaller text element is found. If no smaller text element is found the translation is terminated (step **1228**).

[**0189**] **FIG. 13** depicts a flowchart of an exemplary dictionary database updating method. The flowchart represents a detailed execution example of the arrow between the element **1016** and the database **1050** in **FIG. 10**. As this figure is related to **FIG. 10**, the element **1016** is depicted as used in **FIG. 10**. In **FIG. 10** the element is located between the steps **1010**, **1030** and the steps **1012**, **1032**. These connections are indicated with the two arrows and the respective reference numerals. The validating step **1016** leads to a new validated pair of a text element and a respective translation element. To increase the translation database **1050**, the two elements are provided with a preliminary key identification number (key ID) (step **1300**). Next, the pair of elements is transferred to the database **1050** (step **1302**). Next, the database analyzes both elements to generate a new key ID (in step **1304**). Finally, the text element and the translation element are stored (step **1306**). With a new entry stored in database **1050**, the translation database is growing and enables the database to improve the grade of matching of retrieved translation elements.

[**0190**] The preliminary key ID may be unnecessary if the database itself generates a key ID. Further, the steps of updating the database may only include the updating of statistical information. The database may only be informed about the number indicating how often a known translation element has been used to translate a certain text element. The statistical information may increase the efficiency to provide a translation program with an improved standard database.

[**0191**] **FIG. 14** depicts a section of a translation table for use in the example discussed above for a translation method. The first three columns of the translation table as shown represent the minimum required information for a translation table. The table as shown contains four columns: In the first column the table contains a key identification (ID), to identify the entry in the translation table. The key ID may be a single key, or a main ID with attached sub-IDs. The second column contains language IDs. The language IDs indicate information about the language in use, and may be represented as in the present table by the characterizing numbers of the telephone area code of a particular country (e.g., "1" for U.S. English and "49" for German). The language ID may be a single key, or a main ID with attached sub-IDs, indicating dialects. The third column of the table contains text elements (words) in the different languages. One problem with translations is that synonyms lead to equivocal translations. To simplify the work of the user the table may contain a fourth column, providing additional information about how to use the present word. The fourth column may contain synonyms or background information of the table entry in the third column.

[**0192**] It may be noted that the different keywords may comprise different columns with different ID numbers (not shown). Different columns may contain the sorting index in the language of the word, for sorting the text element entries according to various sorting rules (e.g., alphabetic, number of characters, and grammar information). Additional columns can contain information related to the actual software number, to enable the users of the system to distinguish whether a translation is validated for the present software localization or has been validated for another software localization. The table may further include different columns to relate the present entry to different other languages, so that only one table with n−1 additional columns is needed for each language, to provide translations between all n languages. Instead of n·(n−1) tables to connect each language with every other, another possible table structure with single tables for each language and a central (e.g., n-dimensional) relation matrix (or tensor) would enable the method to use transitive translations. So if a text element has never

been translated from a first language to a third language, the system may determine that the text element has been already validated in a translation to a second language, and has been validated in a translation from the second language to the third language. Thus, the system may be able to derive a translation proposal via the second language. The described examples may also utilize an object oriented database structure for the dictionary database.

[0193] The translation table may further include information about the origin of the text or the translation, if the software localization system supports translation specific keyword databases.

[0194] **FIG. 15***a* depicts three user interaction interfaces ("screenshots") embodied on a computer device. The first screenshot **1540** shows a computer program or device depicting all found matches down to a matching of 84%. The screenshot **1540** comprises four columns. The first column represents a key ID of the text element (sentence) in the text. The second column represents the language ID of the text element entry. The third column represents the database text elements entries. To simplify the decision which of the pretranslation table entries matching the source text element most, are shown in the source language, and therefore all entries in the language column are all "1". The fourth column can represent two kinds of information. For a source text element the fourth column represents a status. The status of the source text element in the first line and third column of screenshot **1540** is "untranslated". In the case of pretranslation entries (line 2 to 5) the fourth column represents a "Quality" value, indicating how exactly the source language entry (line 2 to 5) in the pretranslation table matches the source text element (in line 1). The depicted percentage values may be determined according to the formula:

$$x = \frac{b_{common}}{a_{ste}} \cdot 100,$$

[0195] wherein a_{ste} represents the number of characters of the source text element and b_{common} represents the number of characters in common and common in sequence to the source text element and the source language database entry.

[0196] To simplify use, the differing characters are depicted bold, italic and are surrounded with a box. Other markings of the differences may be applied similarly. In screenshot **40** the field in the third column, second line is surrounded by a bold line to indicate that this translation suggestion is marked. With a user input the user may select the surrounded translation suggestion by a "mouseclick" an "enter" or by contact with a touch screen.

[0197] With the selection the displayed table changes to screenshot **1542**: all unselected translations suggestions are deleted from the screen, and the third line of the table contains the translation of the database entry. Therefore, line 3 column 1 contains the same key ID as the other two lines, and line 3 column 2 contain the language ID of the target language. Line 3 column 3 contains the translation of the source language database entry, and column 4 indicates an exact match of the source text element and the source language database entry. The translation suggestion is marked with a bold frame to indicate that the translation suggestion is selected.

[0198] With a second user input the depicted table changes to table **1544**. The table contains only one line: In the first field, the key ID of the source text element, in the second field the language ID, in the third field the translated text element, and finally in the fourth field the status indicating "fully translated". The first two columns of the tables **1540**, **1542**, **1544** may be economized. The makings may be different. The method described above may skip the screenshot **1542**, if the quality of the selected suggestion is 100%.

[0199] **FIG. 15***b* illustrates schematically a flow diagram according to the user interaction shown in **FIG. 15***a*. The untranslated text may be the same as shown in **FIG. 15***a* and references to **FIG. 15***a* will be made to complete the flow diagram of **FIG. 15***b*. **FIG. 15***b* illustrates the steps of the user iterative interface with respect to the example presented in **FIG. 15***a*.

[0200] In step **S1**, an untranslated text may be processed. The text may be "The computer is standing on the table". The untranslated text may be a text according to language "1".

[0201] In step **S3**, the database may be accessed for retrieving translation elements for the untranslated text of the same language, herein all related translation elements of the language "1". The retrieving may include a matching step wherein the matching step may result in exact matches or in fuzzy matches.

[0202] In step **S5**, the retrieved related elements may be received by the computer executing the user interaction interface. The matching quality value according to the formula

$$\text{formula } x = \frac{b_{common}}{a_{ste}} \cdot 100$$

[0203] is determined for the received related translation elements.

[0204] In step **S6**, the untranslated text, the translation related elements is sequenced in a list by the user interaction interface according to the list **1540** depicted in **FIG. 15***a*. All retrieved entries are listed according to their matching quality.

[0205] In step **S7**, the user may select an entry of the presented list **1540** shown in **FIG. 15***a*. Since the second list entry may have assigned a quality of 100%, the untranslated text and the translating related element match exactly. Due to the matching quality the user may select the second entry of the list, indicated by the bold surrounding lines according to the depicted list **1540** in **FIG. 15***a*.

[0206] In step **S9**, the translation of the translation related element selected in step **S7** may be retrieved from the database. The respective key-ID of the selected translation related element may be used for retrieval. The retrieved translation may be a translation of the selected translation related element into the language "49".

[0207] In step **S11**, the translation of language "49" of the selected translation related element may be received by the user interactive interface from the database. A list may be prepared presenting the untranslated text, the selected trans-

lation related element and the translation to the user desired language, herein a translation from the language "1" which is the language of the untranslated text to the language "49". The matching quality has been determined in step **S5** so that a new determination may be skipped. The matching quality value of the translation related element may be assigned to the corresponding retrieved translation since the determination of the above defined matching quality may be determined only in combination with text of the same language.

[**0208**] In step **S12**, the respective list comprising the key-ID, the language code, the corresponding element and the status or quality, respectively, may be presented in a list illustrated as list **1542** in **FIG. 15***a*.

[**0209**] In step **S13**, the user confirms the translation of the untranslated text by selecting the translation. The selection may be indicated by bold surrounding lines which is shown in list **1542** in **FIG. 15***a*.

[**0210**] As the matching quality of the untranslated text and the translation related element retrieved from the database indicates a value of 100% the translation is an exact translation. The respective entry is available in the database such that no additional entry may have to be added to the translation database.

[**0211**] In step **S14**, the translation of the untranslated text has been done successfully and is finished. The user interactive interface may present the next untranslated text for translating the user and may start again with step **S1**.

[**0212**] The example shown is related to a 100% matching translation. Thus, the exact translation of the untranslated text was found in the database. Therefore, no further database entries may need to be included in the database. **FIG. 16***a* and **FIG. 16***b* present an example of a non-exactly matching untranslated text involving the generation of a new database entry.

[**0213**] **FIG. 16***a* depicts another set of user interaction interfaces ("screenshots"). **FIG. 16***a* shows an example of how the examples described above may be embodied on a computer device. The first screenshot **1650** shows a screenshot of a computer program or device depicting all found matches in a matching range of 66% down to 61%. The screenshot **1650** comprises four columns. As in **FIG. 15***a*, the first column represents a key ID of the text element (sentence) in the text. The second column represents the language of the text element entry. The third column represents the source text element and database text elements entries. To simplify the decision which of the pretranslation table entries match the source text element most closely, the pre-translation entries are shown in the source language, and therefore all entries in the language column are "1". Similar to **FIG. 15***a*, the fourth column can represent two kinds of information. For a source text element the fourth column represents a status. The status of the source text element in the screenshot **1650** is "untranslated". For pretranslation entries (line 2 to 3) the fourth column represents a "Quality" value, indicating the quality of the match between the source language entry in the pretranslation table and the source text element.

[**0214**] The depicted percentage values are determined according to the same formula as in **FIG. 15***a*. The formula used in **FIG. 15***a* may be modified by replacing a_{ste} by a_{dte} representing the number of characters of the database text element. It should be noted that the formula is not limited to the use of characters, but can be applied to words and sentences. The formula may also comprise other terms using grammar structure related values to characterize the matching. Different from **FIG. 15***a*, the retrieved translation suggestions reach a maximum match of only 66%. The closest source text entry in the table is marked with a bold frame.

[**0215**] With a selection of the bold frame by user input, the system depicts the screenshot **1652**. The screenshot **1652** depicts the source text element in the first line, the retrieved database text entry in the second line, and the translation of the database entry in the third line. As the system can detect and mark the differences between the source text element and the database entry, the system marks the differences in the translation of the database entry. All marked elements are depicted bold, italic, and with a small frame in the figure.

[**0216**] Next the user may edit the translation entry pretranslation table in an additional operation by keyboard input. Advanced systems may utilize voice recognition algorithms. In the pretranslation table as shown, the user accepted the proposed translation by a user input.

[**0217**] By accepting the proposed translation matching only 66% with the source text element, the system only translates the matching part of the source text element. The screenshot **1654** represents a partially translated element in the pretranslation table. The table **1654** may further depict the source text element and the retrieved database entry translation. In the first column the table depicts the key ID "1" of the source text element. In the second column the table depicts no entry, as a partially translated sentence is not related to a single language. The entry in the second column can be "1", or "1-49" as part of the text element remains untranslated. In the third column the partially translated text is depicted, with a bold, italic underlined untranslated part depicted in capitals. The untranslated sentence is surrounded by a dotted bold frame to indicate that the present state of the sentence requires additional processing.

[**0218**] To reach the next step, the user may select another text element size to post-process the entry in the preprocessing table, or the user may first translate all other text elements and proceed then with a smaller text element.

[**0219**] The next screenshot **1656** depicts a second translation stage with smaller text elements. In this stage the text elements are words and/or word/article combinations. The present table depicts a newly generated key ID **127** in the first column, to distinguish the different stages of the translation algorithm. In the second column first line, the language ID has not been changed, as the sentence is still only partially translated, while in the second line the retrieved database entry is marked with a "1" and in the third line the translation suggestion is marked as the target language with a "49". In the third column the untranslated part of the sentence is marked as in table **1654**, its related database entry and its translation suggestion retrieved from a translation database.

[**0220**] With a user input accepting the proposed translation, the depicted table changes to table **1658**, containing only one line: In the first field, the key ID of the source text element (in this case the sentence), in the second field the language ID "49", in the third field the translated text element, and finally in the fourth field the status indicating "fully translated".

[0221] With the step of accepting the suggested translation proposal, the system generates a new database entry **1660** containing the source text element and the accepted translation. For efficient retrieval of the pair, the entries include key IDs and language IDs. The new database entry is not necessarily depicted, as the storage operation can be executed automatically.

[0222] The translation database may include different 100% matching database entries, due to different equivocal translations.

[0223] **FIG. 16***b* illustrates schematically a flow diagram according to the user interaction shown in **FIG. 16***a*. The untranslated text may be the same as shown in **FIG. 16***a* and references to **FIG. 16***a* will be made in order to complete the flow diagram of **FIG. 16***b*. **FIG. 16***b* illustrates the operations of the user iterative interface with respect to the example presented in **FIG. 16***a*.

[0224] In step **S20**, an untranslated text may be processed. The text may be "The computer is standing on the table". The untranslated text may further include a text according to language "1".

[0225] In step **S22**, the database may be accessed for retrieving translation elements for the untranslated text of the same language, herein all related translation element of the language "1". The retrieving may include a matching step wherein the matching step may result in exact matches or in fuzzy matches.

[0226] In step **S24**, the retrieved related elements may be received by the computer executing the user interaction interface. The matching quality value according to the formula

$$x = \frac{b_{common}}{a_{dte}} \cdot 100$$

[0227] is determined for the received related translation elements.

[0228] In step **S25**, the untranslated text, the translation related elements is sequenced in a list by the user interaction interface according to the list **1650** depicted in **FIG. 16***a*. All retrieved entries are listed according to their matching quality. The list may present the key-ID, the language the untranslated text or the translation related elements, respectively, and the matching quality. The non-fitting text parts of the translation related elements in relation to the untranslated text may be indicated by italic characters.

[0229] In step **S26**, the user may select an entry of the presented list **50** shown in **FIG. 15***a*. In this example no entry of the list shows a matching quality of 100% indicating that no exact match was retrieved from the database. The second entry of list **50** shown in **FIG. 15***a* may have a matching quality value of 66%. The user may select the best matching entry, herein the second list entry which may be indicated by bold surrounding lines.

[0230] In step **S28**, the translation of the translation related element selected in step **S26** may be retrieved from the database. For retrieving the respective key-ID of the selected translation related element may be used. The retrieved translation may be a translation of the selected translation related element to the language "49".

[0231] In step **S30**, the translation of language "49" of the selected translation related element may be received by the user interactive interface from the database. A list may be prepared presenting the untranslated text, the selected translation related element and the translation to the user desired language, herein a translation from the language "1" which is the language of the untranslated text to the language "49". The matching quality has been determined in step **S24** so that a new determination may be skipped. The matching quality value of the translation related element may be assigned to the corresponding retrieved translation since the determination of the above defined matching quality may be determined only in combination with text of the same language.

[0232] In step **S31**, the respective list including the key-ID, the language code, the corresponding element and the status or quality, respectively, may be presented in a list illustrated as list **1652** in **FIG. 16***a*. The non-fitting text parts of the translation related elements in relation to the untranslated text may be indicated by italic characters.

[0233] In step **S32**, the user confirms the translation of the untranslated text by selecting the translation. The selection may be indicated by bold surrounding lines which is shown in list **1652** in **FIG. 16***a*.

[0234] In step **S33**, as no exact matching translation related element was found in the database, a mixed language translation result may be presented to the user by the user interactive interface. This can be seen in list **1654** shown in **FIG. 16***b*. The text "The computer" may have not been translated. However, the remaining part of the sentence may have been translated by the operations described above.

[0235] In step **S34**, the user may select the untranslated text part for further translation. This is shown in list **1654** depicted in **FIG. 16***a*.

[0236] The translation may be continued by attempting to translate an untranslated text part, defining this part as untranslated text and performing translation operations as described above. The untranslated text "The computer" of the language "1" may be processed as described below.

[0237] In step **S35**, a translation related element according to the untranslated text part may be retrieved from the database. The retrieval may include a matching step wherein the matching step may result in exact matches or in fuzzy matches. The translation related elements of the language "1" according to the untranslated text part are searched.

[0238] The retrieval of translation related elements with respect to the untranslated text part may only return a single matching translation related element. In this case, the respective translation (language "49") may be automatically retrieved from the database since no further translation related elements may be presented to the user for selecting.

[0239] In step **S37**, after the retrieval of the translation related element and the translation into the language "49" thereof a list of the partially translated text, the translation related element according to the untranslated text part and the translation may be prepared. The matching quality value may be determined. The determination of the quality may be derived regarding the untranslated text part and the retrieved

translation related element. The translation may be assigned the same matching quality value.

[0240] To indicate the new translation operation with respect to the untranslated text part, a new key-ID may be assigned to the list entries. As indicated in list **1656** shown in **FIG. 16***b*, the key-ID may be "127".

[0241] In step **S38**, the list of the partially untranslated text, the retrieved translation related element and the translation (language "49") may be shown to the user. The respective list can be seen in list **1656** shown in **FIG. 16***a*. The partially translated text may be assigned no language ID. Status information of the partially translated text may indicate a percentage value according to the translated text part.

[0242] In step **S39**, the user may select the second list entry of the list **1656** shown in **FIG. 16***b*. The selection may be indicated by bold surrounding lines.

[0243] In step **S40**, the combination of the two independent translation operations of the untranslated text of step **S20** may lead to the complete translation thereof. The complete translation may be presented by the user interactive interface to the user. The key-ID of the translation may show the former key-ID value of step **S31**. The status information indicates a successful translation. The respective list **1658** is shown in **FIG. 16***a*.

[0244] In step **S42**, as the untranslated text was not found in the database a new database entry may be generated. The contents of the new database entry is shown in list **1660** of **FIG. 16***b*. The key-ID may be given by an available key-ID of the database and may be generated automatically as a result of the generation of the database entry. The user interactive interface may present the next untranslated text for translation by the user and may start again with step **S20**.

[0245] A new entry may have been generated for the untranslated text presented in step **S20**. Further occurrence of this untranslated text may lead to the retrieval of a translation related element of a matching quality value of 100%. The operations of the user interactive interface for exact matches are described above in detail in **FIG. 15***a* and **FIG. 15***b*.

[0246] The continuous addition of new database entries according to the example described above may ensure that untranslated text is translated in the same manner to the respective translation language, minimizing the usage of synonyms or synonym text constructions which may irritate and confuse a reader of the translated text.

[0247] **FIG. 17** depicts an apparatus for translating texts according to another example by means of a database and a database management system, comprising a module **1774** to communicate with a database **1778**, a module **1775** for exchanging data, and a processing device **1772** for retrieving elements from a database.

[0248] There is also depicted an exemplary realization of the apparatus **1770**. The apparatus **1770** is connected via a communication module **1774** to a network **1776**. Via the network **1776** the apparatus **1770** can communicate with other devices (not shown) to exchange data (e.g., the server **201** of **FIG. 2**). The network **1776** can be the internet, or any other wide area, or local area network. The communication module **1774** is connected to a Central Processing Unit (CPU) **1772**, to execute the exchange and filtering of data. The CPU **1772** is connected to a data exchange interface **1775** (e.g., a user interaction interface). The data exchange interface **1775** can be connected to a keyboard and a display, or via a network to a terminal device to interact with a user. The database **1778** contains the translation data to be accessed, via the communication module **1774** and the network **1776**. To be able to translate texts correctly, the CPU **1772** must be able to receive the source text and to relate the source text with the data stored in the database **1778**. The source text may be received via the data exchange interface **1775**, or via the communication module **1774** from the network **1776**. To simplify the reception of the source text, the user interaction device **1773** and the database **1778** may be incorporated in a single device which may additionally be a portable device (see **FIG. 18**).

[0249] **FIG. 18** depicts a sentence based electronic translation apparatus. The translation apparatus is embodied as an interactive electronic dictionary **1880**. The electronic dictionary can be an electronic book including a display **1881** for depicting a pretranslation table containing a source text, translation proposals, a translated text and translation related information. In the translation table the first column **1885** indicates a key ID, to identify every text element in the source text. The second column **1886** contains source text elements and translation proposals. The third column **1887** contains matching indices to indicate the similarity between the source text entry and the source text element. The fourth column **1888** contains status symbols to indicate whether the source text element requires further processing or translating. The user can interact with the dictionary by a scroll wheel **1883**, keys, a scroll roll **1884** and thumb index menu keys **1890**. A user can pre-select a character or source text element with the thumb index **1890**, scroll the alphabet or the source text elements with the scroll wheel **1883**, and select translation proposals with the keys and the scroll roll **1884**. The electronic dictionary can further contain an interface **1889** to communicate with a network to update its database or exchange data with another database. The dictionary may further upload new source texts via the interface **1889**, or print translated texts on a printer connected to the interface. The electronic dictionary (ED) may further comprise a built in scanner to recognize source texts autonomously. The dictionary may further comprise an extension slot, to connect the ED with handheld scanners, mobile communication devices, or wireless networks. Thus, the ED may be used as a translation device for custom officers, to translate foreign bills of loading. The ED may also be used as a conventional electronic book. An electronic book application has the advantage that users may read a book in its original language, with the option to translate single sentences.

[0250] **FIG. 19** depicts a network system with an apparatus as described in **FIG. 17** or **18** and at least one database **1917**. The apparatus can be a server **1930** or client devices **1911**, **1912**, **1914**, **1915**, **1922**, or **1931**. The database may include the database **1917**, or any other database integrated in one of the client devices **1911-1915**, **1922**, **1931** or the server **1900**. The client devices are connected via networks **1910**, **1930**, **1920** to the server **1900**. It is assumed that the server **1900** contains an internal database (not shown) or can at least access an external database **1917** or a database in one of the client devices **1911-1917**, **1922**, **1931**. In the figure the server is directly connected to two networks, a local area

network (LAN) **1930** and a wide area network (WAN) **19310**. In the LAN **1930** only one client is depicted, the personal computer **1931**. The WAN **1910** is connected to a personal computer **1911**, a personal digital assistant (PDA) **1912**, a browser **1914**, laptop **1915**, a database **1917** and a gateway **1918** to a communication network **1920**. The gateway **1918** indirectly connects a communication device **1922** (e.g., a mobile telephone) with computation abilities with the server **1900**. Thus, all the client devices **1911-1917**, **1922**, **1931** are connected to the server **1900**.

[**0251**] All client devices may comprise user interfaces, CPUs and memories or databases. To execute the method described above, the client devices **1911-1915**, **1922**, **19331** and/or the sever **1900** are enabled to access a translation database. Thereby, the exact location of the translation database is not important. So the database **1917** may include the translation database and offer the client devices **1911-1915**, **1922**, **1931** access to the stored translation elements. Alternatively, each client device **1911-1915**, **1922**, **1931** includes its own translation database enabling each device to translate text elements received from an internal or remote device. The method described above is executable as long as a translation database is accessible (e.g., via the networks **1910**, **1920**, **1930**). In the simplest case the client devices **1911-1915**, **1922**, **1931** themselves include a translation database. In the simplest case the client devices are autonomous software localizing devices being connected to the network only to access new software to be localized. A very complex case includes accessing a database **1917** via a server **1900** which is accessing distinct databases to retrieve text elements to be translated.

[**0252**] As the data of the database is usually retrieved by a database management system (DBMS) the data source may be able to use relational algorithms between tables. The DBMS can store the text elements in a directory table and the translation elements in a related user specific translation table. Thus, the data source of the DBMS may utilize relational algebra to interconnect the text elements with the translation elements. The DBMS user interface enables the user to translate text elements by accessing pre-translation tables. The translation elements can be selected and/or validated in each field of the pre-translation table by a user input. The user input can be a keyboard input, a mouse input and the like. In a last step the validated text elements and validated translation elements are stored in the database, to be automatically retrieved if the user accesses the database with a DBMS to retrieve the text element next time.

[**0253**] Although aspects of methods and systems consistent with the present invention are described as being stored in memory, one having skill in the art will appreciate that all or part of methods and systems consistent with the present invention may be stored on or read from other computer-readable media, such as secondary storage devices, such as hard disks, floppy disks, and CD-ROM; a carrier wave received from a network such as the Internet; or other forms of ROM or RAM either currently known or later developed. Further, although specific components of the system for localization have been described, one skilled in the art will appreciate that a data processing system suitable for use with methods, systems, and articles of manufacture consistent with the present invention may contain additional or different components.

[**0254**] It is noted that the above elements of the above examples may be at least partially realized as software and/or hardware. Further, it is noted that a computer-readable medium may be provided having a program embodied thereon, where the program is to make a computer or system of data processing devices execute functions or steps of the features and elements of the above described examples. A computer-readable medium may include a magnetic or optical or other tangible medium on which a program is embodied, but can also be a signal, (e.g., analog or digital), electromagnetic or optical, in which the program is embodied for transmission. Further, a computer program product may be provided comprising the computer-readable medium.

[**0255**] The foregoing description of an implementation of the invention has been presented for purposes of illustration and description. It is not exhaustive and does not limit the invention to the precise form disclosed. Modifications and variations are possible in light of the above teachings or may be acquired from practicing of the invention. For example, the described implementation includes software but the present invention may be implemented as a combination of hardware and software or in hardware alone. Note also that the implementation may vary between systems. The invention may be implemented with both object-oriented and non-object-oriented programming systems. The claims and their equivalents define the scope of the invention.

What is claimed is:

1. A method in a data processing system for displaying screen objects in a plurality of different languages, each screen object having a text string in a first language, and an associated immutable image file, and associated display characteristics of the associated immutable image file, the method comprising the steps of:

translating the text strings into the plurality of different languages;

automatically generating a text string file containing the text strings in the different languages and desired locations in a directory structure;

using an image manipulation program to generate an immutable image file for each of the text strings in the text string file by using the text strings and the associated display characteristics; and

storing the generated immutable image files at the desired locations in a directory structure such that the different language of the text string in each of the immutable image files is indicated.

2. The method of claim 1, wherein each of the associated immutable image file and the generated immutable image file includes a file in a format of at least one of .gif, bitmap (.bmp), .tiff, .jpg, .png, .xpm, .tga, .mpeg, .ps, .pdf, .pcx, and a graphic format.

3. The method of claim 1, wherein the image manipulation program includes GIMP (GNU Image Manipulation Program).

4. A method in a data processing system for localizing an immutable image file containing text in a first language, the method comprising the steps of:

translating the text from the first language into a second language that is different from the first language; and

automatically generating a translated immutable image file containing the text in the second language.

5. The method of claim 4, wherein the step of automatically generating further includes the step of initiating execution of a script in association with an image manipulation program, to instruct the image manipulation program to input the text in the second language and template information.

6. The method of claim 5, further comprising the step of storing the translated immutable image file at a desired location, wherein the desired location is determined based on location information provided as input to the image manipulation program by the script.

7. The method of claim 6, further comprising the step of accessing, by a user, the translated immutable image file stored at the desired location.

8. The method of claim 5, wherein the template information includes at least one of image size, image shape, image color, position of a text element within the image for the translated immutable image file, and a previously generated immutable image file.

9. The method of claim 4, wherein the step of automatically translating is performed by a translator program associated with a pretranslation database.

10. The method of claim 4, wherein the translated immutable image file includes a file in a format of at least one of .gif, bitmap (.bmp), .tiff, .jpg, .png, .xpm, .tga, .mpeg, .ps, .pdf, .pcx, and a graphic format.

11. The method of claim 4, further comprising the step of identifying the text in the first language by parsing source code of a user interface program.

12. A data processing system for localizing an immutable image file containing text in a first language, comprising:

means for translating the text from the first language into a second language that is different from the first language; and

means for automatically generating a translated immutable image file containing the text in the second language.

13. A computer-readable medium encoded with instructions that cause a data processing system for localizing an immutable image file containing text in a first language to perform a method comprising the steps of:

translating the text from the first language into a second language that is different from the first language; and

automatically generating a translated immutable image file containing the text in the second language.

14. The computer-readable medium of claim 13, wherein the step of automatically generating further includes the step of initiating execution of a script in association with an image manipulation program, to instruct the image manipulation program to input the text in the second language and template information.

15. The computer-readable medium of claim 14, wherein the method further comprises the step of storing the translated immutable image file at a desired location, wherein the desired location is determined based on location information provided as input to the image manipulation program by the script.

16. The computer-readable medium of claim 14, wherein the template information includes at least one of image size, image shape, image color, and position of a text element within the image for the translated immutable image file, and a previously generated immutable image file.

17. The computer-readable medium of claim 13, wherein the translated immutable image file includes a file in a format of at least one of .gif, bitmap (.bmp), .tiff, .jpg, .png, .xpm, .tga, .mpeg, .ps, .pdf, .pcx, and a graphic format.

18. A data processing system for localizing an immutable image file containing text in a first language, the system comprising:

a memory comprising an immutable image file creation system that translates the text from the first language into a second language that is different from the first language, and automatically generates a translated immutable image file containing the text in the second language; and

a processor for running the immutable image file creation system.

19. The data processing system of claim 18, wherein the step of automatically generating further includes the step of initiating execution of a script in association with an image manipulation program, to instruct the image manipulation program to input the text in the second language and template information.

20. The data processing system of claim 19, wherein the image file creation system further stores the translated immutable image file at a desired location, wherein the desired location is determined based on location information provided as input to the image manipulation program by the script.

21. The data processing system of claim 19, wherein the template information includes at least one of image size, image shape, image color, and position of a text element within the image for the translated immutable image file, and a previously generated immutable image file.

22. The data processing system of claim 18, wherein the translated immutable image file includes a file in a format of at least one of .gif, bitmap (.bmp), .tiff, .jpg, .png, .xpm, .tga, .mpeg, .ps, .pdf, .pcx, and a graphic format.

* * * * *