PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : **G06F 9/44**	**A1**	**(11) International Publication Number:** **WO 99/21086** **(43) International Publication Date:** 29 April 1999 (29.04.99)

(21) International Application Number: PCT/US98/22416

(22) International Filing Date: 22 October 1998 (22.10.98)

(30) Priority Data:
60/063,005 23 October 1997 (23.10.97) US

(71) Applicant: ALCATEL USA SOURCING, L.P. [US/US]; 1000 Coit Road, Plano, TX 75075 (US).

(72) Inventors: HURD, Donald, M., II; 1418 Abilene, Allen, TX 75013 (US). WELDON, Richard, S., Jr.; 5700 Caroline Court, Plano, TX 75093 (US).

(74) Agent: FISH, Charles, S.; Baker & Botts, L.L.P., 2001 Ross Avenue, Dallas, TX 75201-2980 (US).

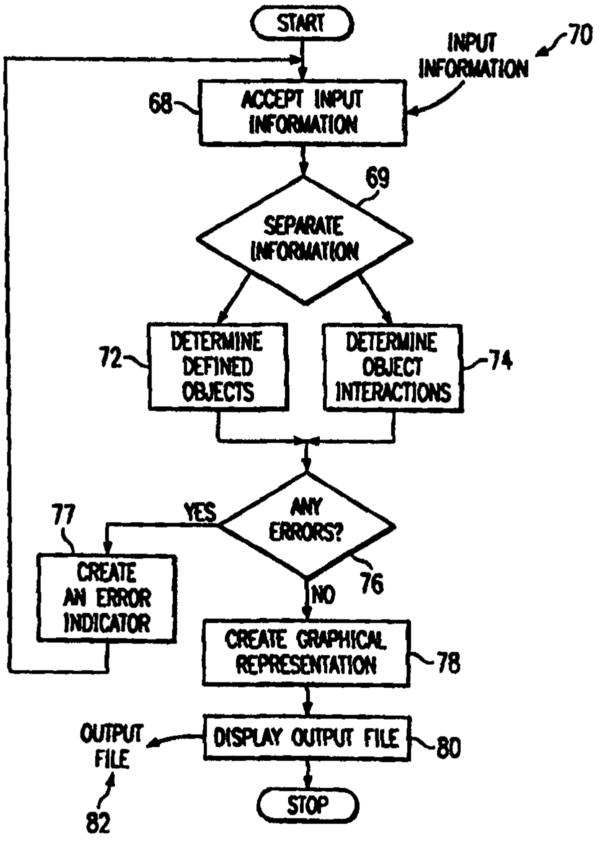
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, HR, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD AND SYSTEM FOR MODELING EVENTS IN A SOFTWARE PROGRAM

(57) Abstract

The present invention comprises a method and system for modeling events in a software program. The system accepts input information associated with a slice of time in an object-oriented software program in a first format. The system then separates the information into a first set of object information that determines the defined objects used in the software program, and a second set of interaction information that specifies the interactions between defined objects. After successfully accepting and separating the input information, the system formats the object information and the interaction information into a second format comprising a graphical representation of the slice of time in the object-oriented software program. The graphical representation of the slice of time in the object-oriented software program is then displayed.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania
AM	Armenia
AT	Austria
AU	Australia
AZ	Azerbaijan
BA	Bosnia and Herzegovina
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
BY	Belarus
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CN	China
CU	Cuba
CZ	Czech Republic
DE	Germany
DK	Denmark
EE	Estonia
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GE	Georgia
GH	Ghana
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IL	Israel
IS	Iceland
IT	Italy
JP	Japan
KE	Kenya
KG	Kyrgyzstan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakstan
LC	Saint Lucia
LI	Liechtenstein
LK	Sri Lanka
LR	Liberia
LS	Lesotho
LT	Lithuania
LU	Luxembourg
LV	Latvia
MC	Monaco
MD	Republic of Moldova
MG	Madagascar
MK	The former Yugoslav Republic of Macedonia
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
MX	Mexico
NE	Niger
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SG	Singapore
SI	Slovenia
SK	Slovakia
SN	Senegal
SZ	Swaziland
TD	Chad
TG	Togo
TJ	Tajikistan
TM	Turkmenistan
TR	Turkey
TT	Trinidad and Tobago
UA	Ukraine
UG	Uganda
US	United States of America
UZ	Uzbekistan
VN	Viet Nam
YU	Yugoslavia
ZW	Zimbabwe

METHOD AND SYSTEM FOR MODELING EVENTS IN A SOFTWARE PROGRAM

NOTICE

TECHNICAL FIELD OF THE INVENTION

This invention relates generally to the field of computer software and more particularly to a method and system for modeling events in a software program.

BACKGROUND OF THE INVENTION

Computer systems in general are well known. A typical system comprises a computer, keyboard, mouse, and a monitor. Further, the computer might comprise a CPU, and RAM and allows various software programs to be used. Software programs are well known and will not be described in detail. Briefly, a software program allows a computer to be customized to perform functions and services that a user demands. Software programs are created using various programing tools which might include, a programing language, editors, debuggers, and other tools to assist the programmer. Software has become so advanced, that programmers need a visual representation of their program to help them develop and finalize the software. This is especially true with object-oriented programming.

Object-oriented programs utilize multiple objects. Each object is a "black box" that receives and sends messages. Each object is capable of a specific task, and by programming software to send various messages to the objects, a user can have the objects perform various functions. The user is not concerned with how the object works, but rather is only concerned with what the object does. Accordingly, programming in an object-oriented language involves sending messages to and from various objects.

To simplify the task of programming in an object-oriented language, it is often desirable to visualize what objects are being used and how they interact. A use case is a mechanism for modeling the sequence of events and objects used in a software program.

Use case modeling is an analysis technique for eliciting, understanding, and defining (functional) system requirements. Use cases helps you focus on the usability of a system, that is *what* the users want the system to do for them. A use case model (together with business object definitions) is therefore the best foundation we know of, for writing a contract between customers and developers.

The use case model defines system requirements but does not deal with internal system structure. In theory this means that, based on a use case model, any sound design method -- structured or object oriented -- can be utilized to construct the system, as long as the product can perform all the use cases well (correct, flexible, good UI, etc.).

Object orientation represents the best practice for building high quality systems efficiently. The purpose of this paper is therefore to show how the use case model can be mapped to an object model. We do this without assuming any particular method for the object design process, though we will use the notation of OOSE since we are most familiar with that. When there is no risk for confusion we use the term object when speaking of instances as well as of classes.

Graphical use case programs are known in the art. One such program is Software Through Pictures®. To build a successful application its important to have a clear understanding of your business. You need development tools with the flexibility, scalability, and openness to change directions as fast and as often as your business needs change. Software Through Pictures® provides you this capability by providing an integrated multi-user environment sharing a common architecture and central repository. You choose the modeling method which provides your organization the best understanding of your business needs. Current graphical use case programs help a user develop software by displaying software in a graphic environment. Nevertheless, currently these programs are generally incapable of using other input and output formats, e.g., tabular. The graphical environment is difficult to operate and therefore greatly hinders the ability of programmers to visualize what is occurring in the software program. Additionally, many programmers would better visualize software programs if they had representations of the program in multiple formats.

Moreover, current graphical use case programs are difficult to edit and may not be capable of displaying looping, timing sequences, conditional messages, or repeated messages. The looping feature allows software programs to iterate through certain sequences of code in order to complete a task. For the timing sequence feature, software relies on interaction from other software components in a specific sequence. If the service or data is not provided in a certain amount of time, the user presumes the transaction failed and takes steps to recover. Conditional messages are messages that may or may not be sent or may be sent in different formats depending on certain conditions, including but not limited to states of hardware elements (including active, out of service, etc.) or states of software (including start up, shutting down, etc.). Repeated messages are messages between software that may need to be repeated, such as when the receiving software did not acknowledge receipt of the message. Also, when a user wishes to make a minor change, the entire graph has to be edited. Therefore, a procedure for making edits and changes to the input without having to recreate the input is desirable. Further, features such as looping, timing sequences, conditional messages and repeated messages are widely used in software programs and programmers would greatly benefit from having these features displayed. Moreover, high and low-level designers of computer software need use cases that can easily be edited and can display these needed features in a compact and informative manner.

SUMMARY OF THE INVENTION

Therefore, a need has arisen for a new method and system for modeling events in a software program that overcomes the disadvantages and deficiencies of the prior art.

According to an embodiment of the present invention, there is provided a method of developing a graphical representation of a slice of time in an object-oriented software program that includes accepting information associated with the software program in a first format. The information is separated into a first set of object information defining a number of objects used in the software program and a second set of interaction information specifying interactions between the defined objects. The object information and the interaction information are formatted into a second format comprising graphical representation of the slice of time in the object-oriented software program. The graphical representation of the slice of time in the object-oriented software program is then displayed.

The preferred embodiments of the present invention provide various technical advantages. For examples, the present invention enables the information content of a use case to be displayed in multiples formats. That is, information can be input into the use case through an ASCII file, a tabular input, a Framemaker file, or using the source code of an object-orientated software program. The system and method of the present invention will generate a graphical use case from the input.

Other features and aspects of the present invention will be apparent from the drawings and detailed description of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

For a more complete understanding of the present invention and the advantages thereof, reference is now made to the following description taken in conjunction with the accompanying drawings, wherein like reference numerals represent like parts, in which:

Figure 1 is an overview of one embodiment of the present invention.

Figure 2 is a flow chart illustrating the steps of a method for developing a graphical representation of a slice of time in an object-oriented software program according to one embodiment of the present invention.

Figure 3a shows an example of a tabular input for a use case according to one embodiment of the present invention.

Figure 3b shows an example of an ASCII input for a use case according to one embodiment of the present invention.

Figure 3c shows an example of a Framemaker input for a use case according to one embodiment of the present invention.

Figure 4a shows an example of the first page of an error file created when errors are encountered in the execution of the present invention.

Figure 4b shows an example of the second page of an error file created when errors are encountered in the execution of the present invention.

Figure 5 shows an example of the graphical output for the input from Figure 3a according to one embodiment of the present invention.

Figure 6a shows an example of a tabular input containing conditional messages, repeated messages, and looped messages for a use case according to one embodiment of the present invention.

Figure 6b shows an example of the first page of the graphical output containing conditional messages, repeated messages, and looped messages for the input from Figure 4a according to one embodiment of the present invention.

Figure 6c shows an example of the second page of the graphical output containing conditional messages, repeated messages, and looped messages for the input from Figure 4a according to one embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

The preferred embodiments of the present invention provide a method and system for developing a graphical representation of a slice of time in an object-oriented software program. According to a preferred embodiment, this is accomplished by providing a tabular interface for a user to input information on a software program.

Additionally, the system is capable of accepting other input formats. For example, an ASCII format can be used. Further, a Framemaker input format can be used. Framemaker is a documentation tool on UNIX. It provides similar capabilities that are in Microsoft Word. Its use is to create tabularized information consisting of rows that identify the objects/actors and rows indicating messages passed between objects/actors.

Figure 1 schematically depicts a computer system within which a method and system according to one preferred embodiment of the present invention can operate. Figure 1 shows a development network 10 comprising n developers, and n file servers connected via network NET1 which is, for example, a wide area network (WAN) or local area network (LAN). The development network, NET1, is connected to a system network comprising inter alia n clients and n servers connected through NET2 which is, for example, a WAN or LAN. Development network 10 is used, for example, to develop software systems. System network 12 is the network upon which the software systems developed on development network 10 operate. Development network 10 and system network 12 communicate, for instance, via a third network, NET3. NET3, may also comprise, for instance, a wide area or local area network.

The system and method of the present invention typically operate within development network 10. In one embodiment, the system and method of the present invention reside on development file server n so that its inventive features are easily accessible by component developers 1-n. Systems and methods according to preferred embodiments of the present invention are not limited in their operation to computer systems as shown in Figure 1, but may operate on any suitable computer system. For example, in another embodiment, the system and method of the present invention reside in RAM of a stand alone computer so that its inventive features are accessible to all authorized users of the stand alone computer.

Figure 2 depicts a flow chart demonstrating how the operation of the present invention occurs according to one embodiment of the present invention. The system accepts input information 70 at step 68. The system then separates the information at step 69 into a first set of object information, determining the defined objects used in the software program at step 72, and a second set of interaction information that specify the interactions between defined objects at step 74. After accepting and separating the information associated with the software program, the system determines if there were any errors that occurred during operation of the system, at step 76. If there were errors, the system creates an error indicator, at step 77, allows the error to be corrected and waits for the system to be run again. If there were no errors, the object information and the interaction information is formatted into a graphical representation of the slice of time in the object-orientated software, at step 78. The graphical representation of the slice of time in the object-orientated software program is then displayed in step 80 in the form of an output 82. Each of these steps will now be explained in more detail in conjunction with Figures 3-5.

At step 68, information associated with a particular slice of time in an object oriented software program is accepted as input. The input 70 can take almost any form. For example, input 70 can include a tabular input file, an ASCII file, a Framemaker file, or even software program source code. Figure 3a shows an example of a tabular input 20 according to a preferred embodiment of the present invention. Definition section 41 defines the framework of the software program to be displayed and includes a title row 21, blank row 23, note rows 27, and class row 24.

Title row 21 contains the title field 22 for the graphical representation of the use case. Blank row 23 is used to divide the title from the rest of the data. Note rows 27 are optional rows to display notes in the graphical output. Note fields 28 contain the notes that are to be displayed. Class rows 24 define the objects used in the use case of interest. The class fields 25 of the class rows 24 contain the names of the objects. The order of class rows 24 defines the order the objects are displayed in the graphical output.

Definition section 42 is used to define the messages occurring between the defined objects at the slice of time of interest and includes message header rows 26, a plurality of message rows 29, and label rows 40. Message header rows 26 are the headers for messages contained in the use case between objects. Data is not included in message header rows 26, but rather message header rows 26 are used to indicate that the data that follows will be messages. Message rows 29 contain data indicating the messages to be displayed. Message fields 32 are the actual messages to be displayed. "From fields" 33 contain the starting points for the messages and "to fields" 34 contain the ending points for the messages. Label rows 40 are used to create labels to be displayed before the message that follows the label row. The contents of the label are contained in the data stored in label field 41. In any field, except from fields 33 or to fields 34, a backslash ("\") is used to force text onto multiple lines.

According to another embodiment of the present invention, an ASCII file is used as input 70. ASCII file 150, shown in Figure 3b, contains the same input information as tabular input 20 but in ASCII code. Some aspects of tabular input 20 that are used for structure, however, are not required in ASCII file 150. For example, the blank row 23 and the message header row 26 are not required in the ASCII file 150. An additional feature of using an ASCII file input is that the system can output a graphical output as in Figure 5, a tabular output (looking exactly like the tabular input from Figure 3a), or both outputs.

According to another embodiment, a Framemaker file is used as input 70. A Framemaker file input is configured to contain the same information as tabular input 20 and ASCII file 150. Similar to ASCII file 150, the Framemaker file input does not contain the structured information used in tabular input 20. Figure 3c shows an example of a Framemaker input for a use case according to one embodiment of the present invention.

According to another embodiment the actual source code for the object oriented software program is used as input 70. That is, instead of reformatting the information of the software program into another format, the source code of the object-orientated software program could be used as input 70.

After accepting the information associated with the software program, in step 68, the system parses through the software program. In step 69, the system separates the information into object information that defines the objects used in the slice of time of the object-orientated software program, and interaction information that specifies the interactions between defined objects.

In step 72, the defined objects in the input information are determined. The defined objects are the objects that form the skeletal structure of the software program being displayed. Reference to the skeletal structure is used to indicate the background of the program, for example the title, the notes, and the objects. In Figure 3a, the information in title row 21, note rows 27 and class rows 24 form the skeletal structure. The list of defined objects, however, can include any number of different types of objects that form the skeletal structure of the software program being displayed. The defined objects can be determined in a number of ways. For example, according to a preferred embodiment, the defined objects are determined from the structure of input 70. For example, when tabular input 20 is used, the defined objects are determined as the content of class fields 25 of class rows 24. That is, the "CLASS" indication in class rows 24 serves as a "tag" to facilitate determination of defined objects. A similar technique is used in conjunction with ASCII file input 150 shown in Figure 3b and a Framemaker file input. If source code is used as input 70, the method of Figure 2 determines the defined objects using the class definitions from the source code.

In step 74, the object interactions are determined. The object interactions describe how different objects in the software program interact, enabling a user to visualize what is occurring in the software program at a given slice of time. For example, in Figure 3a, the object interaction information includes the information in message rows 29 and label rows 40. The list of object information, however, can include any number of different types of information that demonstrate how the objects in the software program interact at any given slice of time. The object interactions are determined in any of a number of ways. In a preferred embodiment, similar to that described above, the object interactions are determined from the structure of the input file. For example, when tabular input 20 is used, the object interactions are determined from the content following message header row 26. A similar technique is used in conjunction with other input formats.

After determining the defined objects and object interactions, the method determines if there are any errors in the information. If there are errors in the input information, an error indicator is created. One example of an error indicator is a use case error file 400 shown in Figures 4a-b. Figure 4b is the second page of the error file 400. The error file 400 is useful for identifying what errors had occurred. Error file 400 is especially useful for indicating errors in the input 70. Error file 400 is output in ASCII format. The error file 400 contains all of the information in input 70, but with error comments 402 and 404 added. In error message "receiver class does not exist" 402, the user entered as an option to field 207 "to class" 401. The class "to class" had not been previously defined in any of the class rows 24. Therefore the error message "receiver class does not exist" 402 was output in the error file 400. In error message "sender class does not exist" 404, the option from field 206 "from class" 403 had been entered. The class "from class" had not been previously defined in any of the class rows 24. Therefore the error message "sender class does not exist" 404 was output in the error file 400. After outputting the error file 400, the system allows the input to be corrected and run again. Although the error file 400 is shown in Figure 4, the error indicator can include any means for signifying that an error has occurred. For example, the error indicator could be a visual display or an audible sound.

If no errors are found at step 76, the system formats the object information and the interaction information into a graphical representation of the slice of time in the object-orientated software program at step 78. The output 82 demonstrate what objects in the software program are being used and how the objects are interacting.

After formatting the information at step 78, output 82 demonstrating the graphical representation of the slice of time in the object-orientated software program is displayed at step 80. Output 82 can also be saved as a permanent file either on a hard disk or a floppy diskette. Figure 5 shows an example of a graphical output 100 created from tabular input 20 and displayed during step 80. The graphical output 100 displays graphically what each object in the software program is and how each objects interacts with other objects at a fixed time during the software program.

The defined objects form the skeletal structure of the software program at a given slice of time being displayed. Title 101 is displayed at the bottom of the graphical output and contains the same information found in title field 22. If the user has used any note rows 27, information in note fields 28 are placed on the top of the graphical output. Dashed lines 104 of the graphical output indicate objects 103 in the program. The objects 103 are derived from the information found in the class fields 25 on the class rows 24. Each of objects 103 form a column in the graphical output. The objects 103 are displayed from left to right in the order in which the class row 24 appears in the input 70. That is, the object 103 appearing on the far left of the display was derived from the first class row 24 listed in the input 70. The object 103 appearing to the immediate right of the first object 103 was derived from the second class row 24 listed in the input 70. This ordering continues such that the object 103 displayed on the far right of the display was derived from the last class row listed in the input 70. This structure allows a user to easily visualize all of the objects being used at a fixed time, and in a fixed order. This forms the skeletal structure of the software program at a given slice of time being displayed.

The object interactions indicate how different objects in the software program interact, enabling a user to visualize what is occurring in the software program at a given slice of time. Messages 105 demonstrate how the objects interact with each other. An arrow 106 is placed between the two objects interacting; the arrow starts at the object identified from field 33 in the message row 29 and ends at the object identification to field 34 in the message row 29. The message 105 is placed between the objects above and below the arrow 106. A label 107 is placed before any message 105 contained in a message row 29 that was proceeded by a label row 40 in the input 70. The contents of label 107 is the information from the label field 41 in the label row 40.

The system and method of the present invention are capable of producing and displaying many non-standard features that are useful in visualizing what is occurring in a software program at a given slice of time. Some of these features are demonstrated in Figures 6a-c. Figure 6a shows another example of a tabular input 200 according to a preferred embodiment of the present invention. Only features not explained in conjunction with Figures 1-5 will be explained in conjunction with Figures 6a-c. The first non-standard feature is a repeat message. Repeat row 201 is used to denote that a message or set of messages are to be indicated as being repeated for a specified number of times. The repeated row 203 indicates the end of the messages or set of messages to be repeated. Accordingly, only the messages between the repeat row 201 and the repeatend row 203 will be repeated. Repeat field 202 indicates the number of times the message or set of messages are to be repeated. In this example, the message are to be repeated "For XX" times. Repeats can be nested as shown using a nested repeat row 218. Nested repeats are used to depict a repeat within a first repeat. A nested repeat is formed when a nested repeat row 218 is placed after a first repeat row 201 but before the repeatend row 203 for that first repeat row 201. Nested repeat field 220 indicates the number of times the message or set of messages are to be repeated, for example "For X" times.

The second non-standard feature is an option. An option can be used to denote that a message or set of messages are performed conditionally. All messages between the option row 205 and an optionend row 208 are indicated as being performed conditionally. The option from field 206 indicates where the starting object is and the option to field 207 indicates the location of the ending object. Option fields can also be nested to depict an option within another option, such as with nested option row 216. Nested option row 216 is an option that is placed after a first option row 205, but before the optionend row 208 for that first option row 205.

A third non-standard feature that can be demonstrated using the method and system of the present invention is looping. Looping is demonstrated in the message row 29 by using a class row 29 having the same from field 33 and to field 34.

A fourth non-standard feature that can be demonstrated using the method and system of the present invention is the display of timing sequences. Timing sequences are for actions that are required to occur in a specified amount of time. For example, if a software component does not acknowledge a message sent in 10 nanoseconds, the sender can make some assertions as to the state of the component to which communication was attempted. The diagram generated shows the timing constraints (for example, 10 nanoseconds) and describes actions to take should the timing constraints be violated.

Other features that can be illustrated using the method and system of the present invention are spaces and page breaks. Space row 204 is optional and for esthetic purposes only. Space row 204 creates a white space between messages to make them easier to read. Additionally, a page break can be inserted in the graphical output by using a break row (not shown).

Figures 6b and 6c show the graphical output 300 developed from the tabular input shown in Figure 6a. Repeat message 301 is a repeated message and is created from the information between the repeat row 201 and repeatend row 203. "For XX," indicated by numeral 302 and written above messages 301 and 311, indicates that messages 301 and 311 are repeated "XX" times. Nested repeat messages 306 is a repeat message within a first repeat message and is created from the information between nested repeat row 218 and nested repeatend row 219. "For X," indicated by numeral 207 and written above messages 308 and 312, indicates that messages 308 and 312 are repeated "X" times.

Options are depicted with a shaded area 303. The first option is displayed with shaded area 304 and is created by an option row 205. The shaded area 304 is bound by the option from field 206 and the option to field 207. The shaded area 304 contains all the messages between option row 205 and optionend row 208. Nested option 306 is displayed using a lighter shaded area 305 and contains all the messages between nested option row 216 and nested optionend row 217.

A looping arrow 310 denotes looping and is created by a message 209 with the same from field 33 and to field 34. Note that the output 300 is broken into a first page (Figure 6b) and a second page (Figure 6c). This occurs because of the length of the output. The output could also be intentionally split using a break row (not shown).

A system for representing objects and how objects interact in a software program has been described, and specific embodiments thereof have been presented with reference to a UNIX ® environment. Nevertheless, the embodiments presented are exemplary, and the invention is not limited to the embodiments presented or to operation in the UNIX® environment. For example, other types of environments, for instance, a DOS® environment, could be established using procedures similar to those explained above and could be given analogous functionality to the invention. Additionally, for purposes of simplifying the detailed description of the invention, the preferred embodiments of the present invention are explained with reference to specific inputs with specific characteristics, such as a tabular input with three rows. The invention is not, however, limited in operation by any of the specific characteristics of the inputs used in the embodiments. Further, the method and system according to preferred embodiments of the present invention are not limited to operation within computers and software programs as the examples listed with Figures 1 to 6. Other variations of the present invention are possible and are within the scope of the invention defined by the following claims.

WHAT IS CLAIMED IS:

1. A system for developing a graphical representation of a slice of time in an object-oriented software program comprising:

input means for accepting information associated with the software program in a first format;

means for separating the information into a first set of object information defining a number of objects used in the software program and a second set of interaction information specifying interactions between the defined objects;

means for formatting the object information and the interaction information into a second format comprising graphical representation of the slice of time in the object-oriented software program; and,

means for displaying the graphical representation of the slice of time in the object-oriented software program.

2. The system of claim 1 wherein the first format comprises a tabular format.

3. The system of claim 1 wherein the first format comprises a Framemaker format.

4. The system of claim 1 wherein the first format comprises an ASCII format.

5. The system of claim 1 wherein the first format comprises source code of the object-orientated software program.

6. The system of claim 1 wherein the graphical presentation comprises a graphical display that demonstrates how the objects operate and interact.

7. The system of claim 1 further comprising means for displaying errors in the object information and the interaction information.

8. The system of claim 1 wherein the graphical representation comprises information on looping.

9. The system of claim 1 wherein the graphical representation comprises information on conditional messages.

10. The system of claim 1 wherein the graphical representation comprises information on repeated messages.

11. The system of claim 1 wherein the graphical representation comprises information on timing sequences.

12. The system of claim 1 wherein the second format comprises a graphical format.

13. The system of claim 1 wherein the second format comprises a tabular format.

14. A method for developing a graphical representation of a slice of time in an object-oriented software program:

inputting information associated with the software program in a first format;

separating the information into object information defining a number of objects used in the software program and interaction information specifying interactions between the defined objects;

formatting the object information and the interaction information into a second format comprising a graphical representation of the slice of time in the object-oriented software program; and,

displaying the graphical representation of the slice of time in the object-oriented software program.

15. The method of claim 14 wherein the first format comprises a tabular format.

16. The method of claim 14 wherein the first format comprises a Framemaker format.

17. The method of claim 14 wherein the first format comprises an ASCII format.

18. The method of claim 14 wherein the first format comprises source code of the object-orientated software program.

19. The method of claim 14 wherein the graphical representation comprises a graphical display that demonstrates how the objects operate and interact.

20. The method of claim 14 further comprising displaying errors in the object information and the interaction information.

21. The method of claim 14 wherein the graphical representation comprises information on looping.

22. The method of claim 14 wherein the graphical representation comprises information on conditional messages.

23. The method of claim 14 wherein the graphical representation comprises information on repeated messages.

24. The method of claim 14 wherein the graphical representation comprises information on timing sequences.

25. The method of claim 14 wherein the second format comprises a graphical format.

26. The method of claim 14 wherein the second format comprises a tabular format.

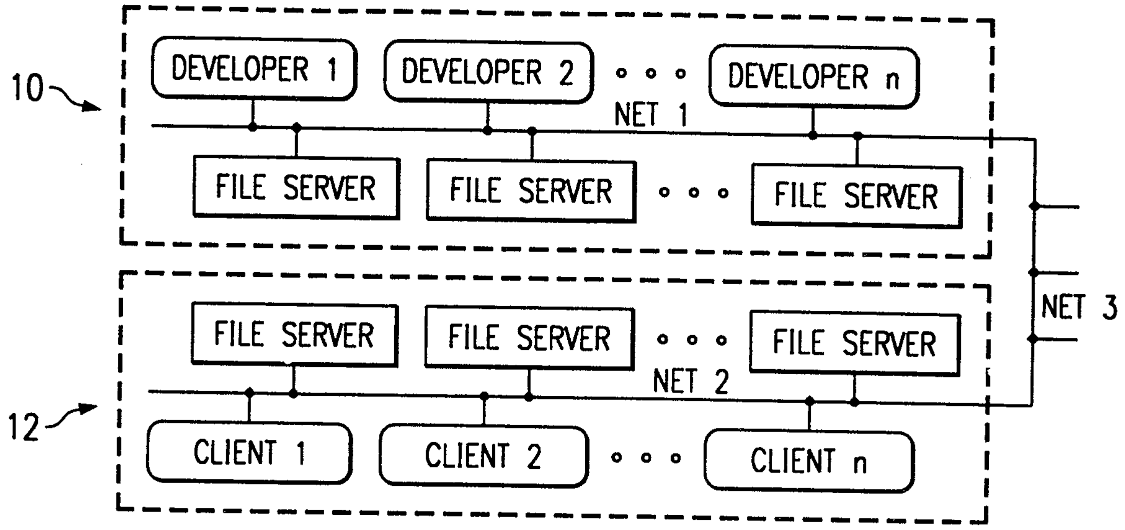

10
DEVELOPER 1
DEVELOPER 2
DEVELOPER n
NET 1
FILE SERVER
FILE SERVER
FILE SERVER
NET 3
FILE SERVER
FILE SERVER
FILE SERVER
NET 2
12
CLIENT 1
CLIENT 2
CLIENT n

FIG. 1

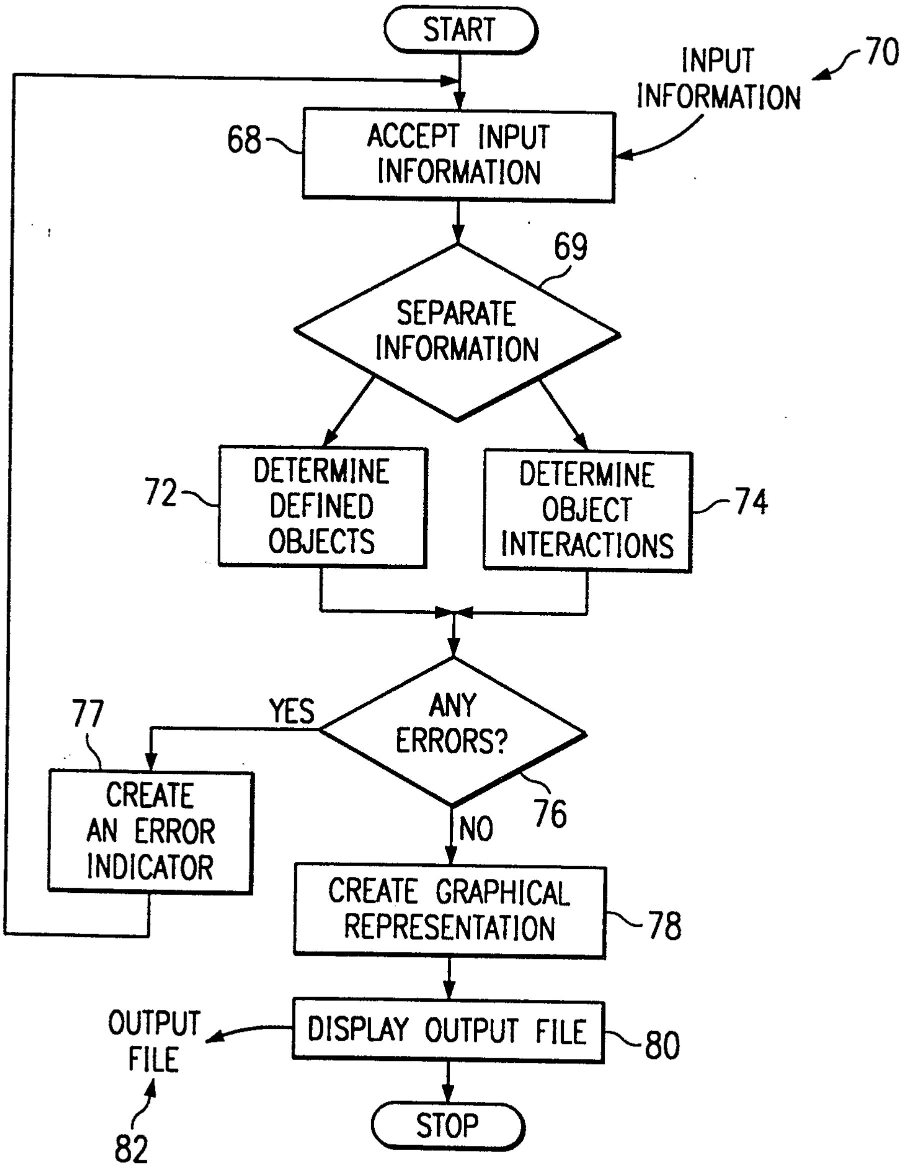
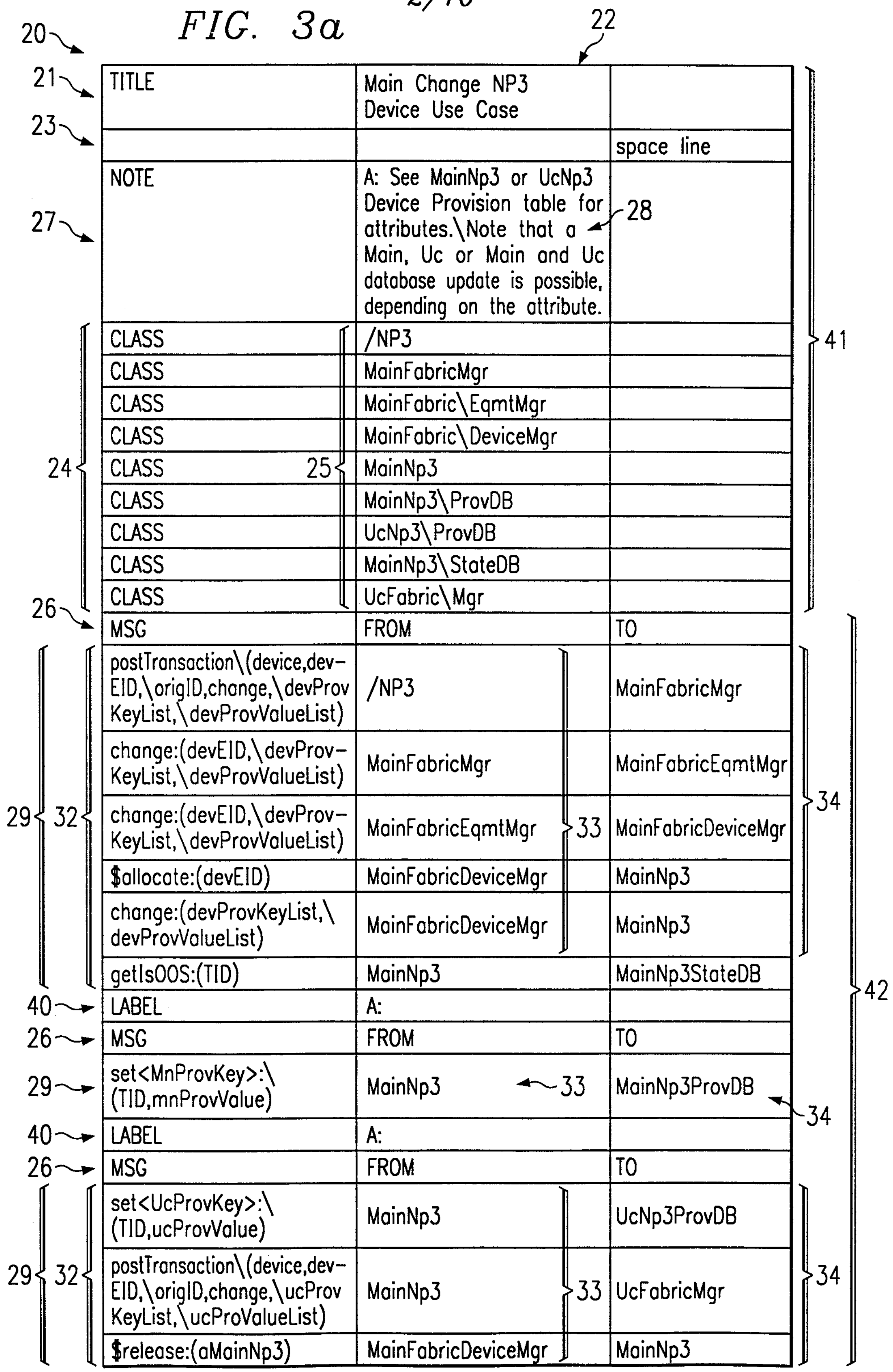
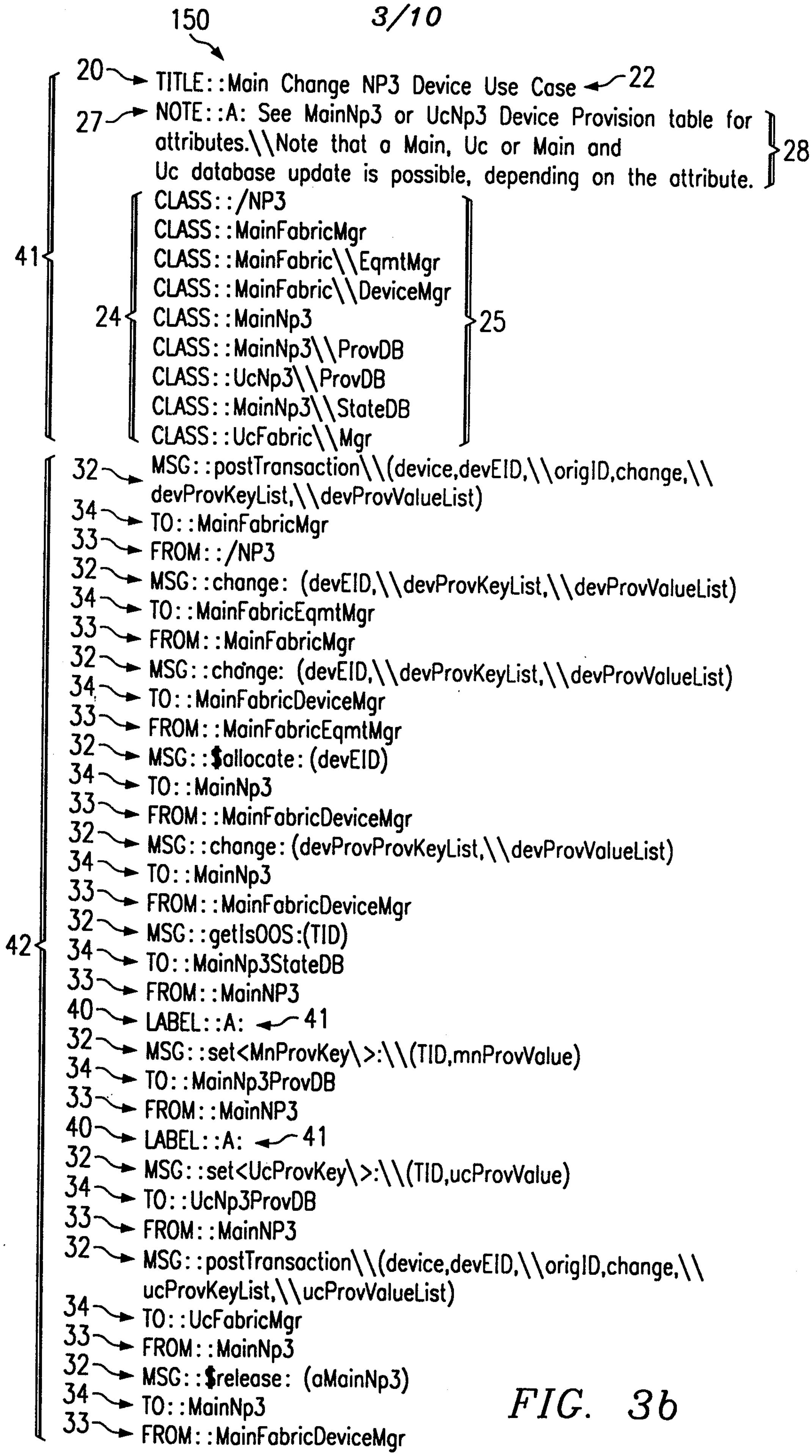

START
INPUT INFORMATION
70
68
ACCEPT INPUT INFORMATION
69
SEPARATE INFORMATION
72
DETERMINE DEFINED OBJECTS
DETERMINE OBJECT INTERACTIONS
74
77
YES
ANY ERRORS?
CREATE AN ERROR INDICATOR
76
NO
CREATE GRAPHICAL REPRESENTATION
78
OUTPUT FILE
DISPLAY OUTPUT FILE
80
82
STOP

FIG. 2

FIG. 3a


20
21
22
23
27
28
24
25
26
29
32
33
34
41
42
40
26
29
33
34
40
26
29
32
33
34

TITLE	Main Change NP3 Device Use Case	
		space line
NOTE	A: See MainNp3 or UcNp3 Device Provision table for attributes.\Note that a Main, Uc or Main and Uc database update is possible, depending on the attribute.	
CLASS	/NP3	
CLASS	MainFabricMgr	
CLASS	MainFabric\EqmtMgr	
CLASS	MainFabric\DeviceMgr	
CLASS	MainNp3	
CLASS	MainNp3\ProvDB	
CLASS	UcNp3\ProvDB	
CLASS	MainNp3\StateDB	
CLASS	UcFabric\Mgr	
MSG	FROM	TO
postTransaction\(device,devEID,\origID,change,\devProvKeyList,\devProvValueList)	/NP3	MainFabricMgr
change:(devEID,\devProvKeyList,\devProvValueList)	MainFabricMgr	MainFabricEqmtMgr
change:(devEID,\devProvKeyList,\devProvValueList)	MainFabricEqmtMgr	MainFabricDeviceMgr
$allocate:(devEID)	MainFabricDeviceMgr	MainNp3
change:(devProvKeyList,\devProvValueList)	MainFabricDeviceMgr	MainNp3
getIsOOS:(TID)	MainNp3	MainNp3StateDB
LABEL	A:	
MSG	FROM	TO
set<MnProvKey>:\(TID,mnProvValue)	MainNp3	MainNp3ProvDB
LABEL	A:	
MSG	FROM	TO
set<UcProvKey>:\(TID,ucProvValue)	MainNp3	UcNp3ProvDB
postTransaction\(device,devEID,\origID,change,\ucProvKeyList,\ucProValueList)	MainNp3	UcFabricMgr
$release:(aMainNp3)	MainFabricDeviceMgr	MainNp3

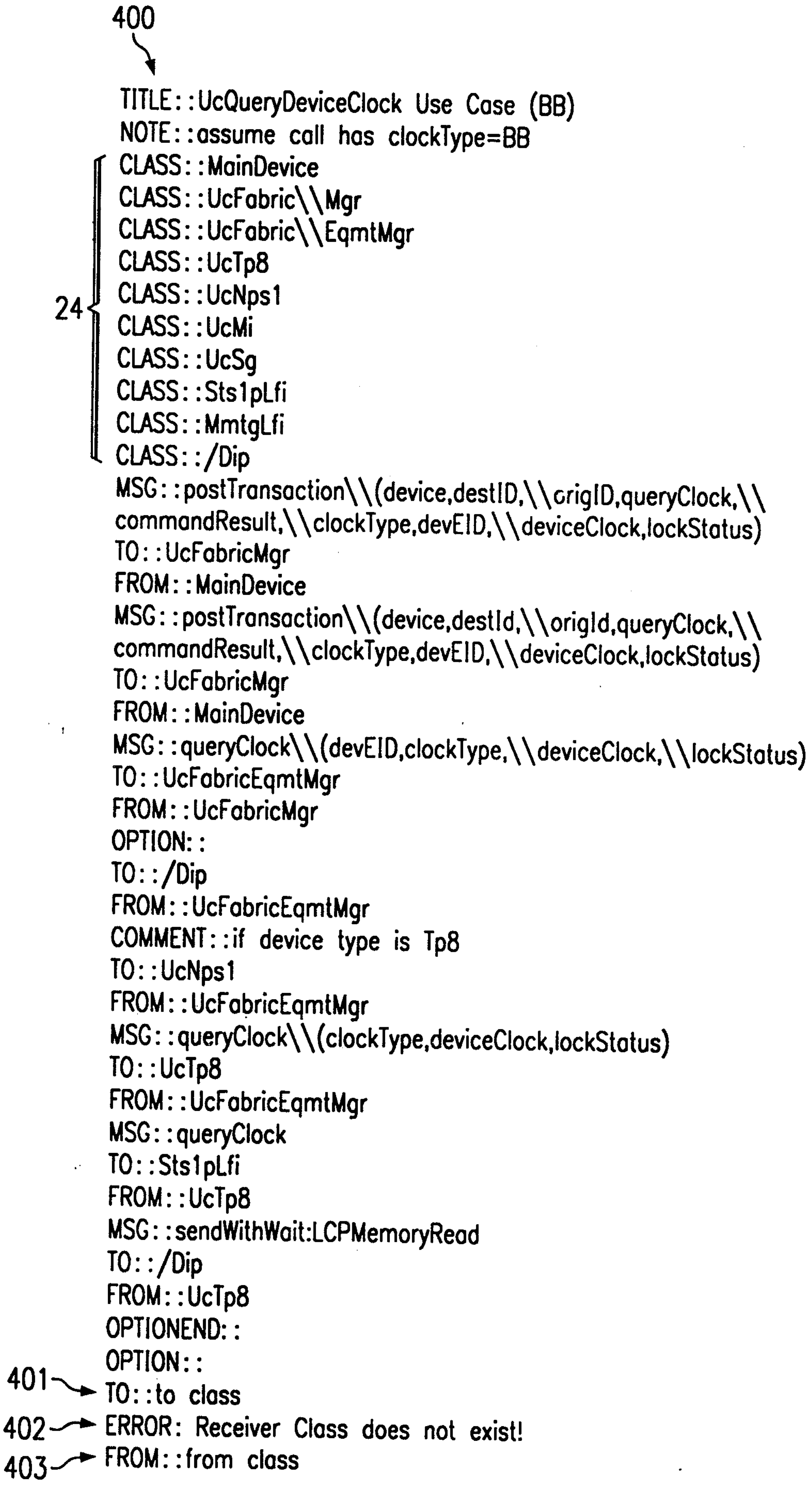


```
TITLE::Main Change NP3 Device Use Case
NOTE::A: See MainNp3 or UcNp3 Device Provision table for
attributes.\\Note that a Main, Uc or Main and
Uc database update is possible, depending on the attribute.
CLASS::/NP3
CLASS::MainFabricMgr
CLASS::MainFabric\\EqmtMgr
CLASS::MainFabric\\DeviceMgr
CLASS::MainNp3
CLASS::MainNp3\\ProvDB
CLASS::UcNp3\\ProvDB
CLASS::MainNp3\\StateDB
CLASS::UcFabric\\Mgr
MSG::postTransaction\\(device,devEID,\\origID,change,\\
devProvKeyList,\\devProvValueList)
TO::MainFabricMgr
FROM::/NP3
MSG::change: (devEID,\\devProvKeyList,\\devProvValueList)
TO::MainFabricEqmtMgr
FROM::MainFabricMgr
MSG::change: (devEID,\\devProvKeyList,\\devProvValueList)
TO::MainFabricDeviceMgr
FROM::MainFabricEqmtMgr
MSG::$allocate: (devEID)
TO::MainNp3
FROM::MainFabricDeviceMgr
MSG::change: (devProvProvKeyList,\\devProvValueList)
TO::MainNp3
FROM::MainFabricDeviceMgr
MSG::getIsOOS:(TID)
TO::MainNp3StateDB
FROM::MainNP3
LABEL::A:
MSG::set<MnProvKey\>:\\(TID,mnProvValue)
TO::MainNp3ProvDB
FROM::MainNP3
LABEL::A:
MSG::set<UcProvKey\>:\\(TID,ucProvValue)
TO::UcNp3ProvDB
FROM::MainNP3
MSG::postTransaction\\(device,devEID,\\origID,change,\\
ucProvKeyList,\\ucProvValueList)
TO::UcFabricMgr
FROM::MainNp3
MSG::$release: (aMainNp3)
TO::MainNp3
FROM::MainFabricDeviceMgr
```

FIG. 3b

TITLE	Create OTM UC	
		space line
NOTE		
CLASS	System Level Mgt I/F	
CLASS	System Level Config Mgr	
CLASS	System Level Slot MO	
CLASS	System Level OTM Class	
CLASS	System Level OTM MO	
CLASS	System Level POM	
MSG	FROM	TO
<Create OTM	System Level Mgt I/F	System Level Config Mgr
Validate & Create>	System Level Config Mgr	System Level Slot MO
Validate & Create	System Level Slot MO	System Level OTM Class
OK	System Level OTM Class	System Level Slot MO
Instantiate	System Level Slot MO	System Level OTM MO
Write Persistent Data	System Level OTM MO	System Level POM
OK	System Level POM	System Level OTM MO
OK	System Level OTM MO	System Level Slot MO
OK	System Level Slot MO	System Level Config Mgr
OK	System Level Config Mgr	System Level Config Mgr
OK	System Level Config Mgr	System Level Mgt I/F

FIG. 3c

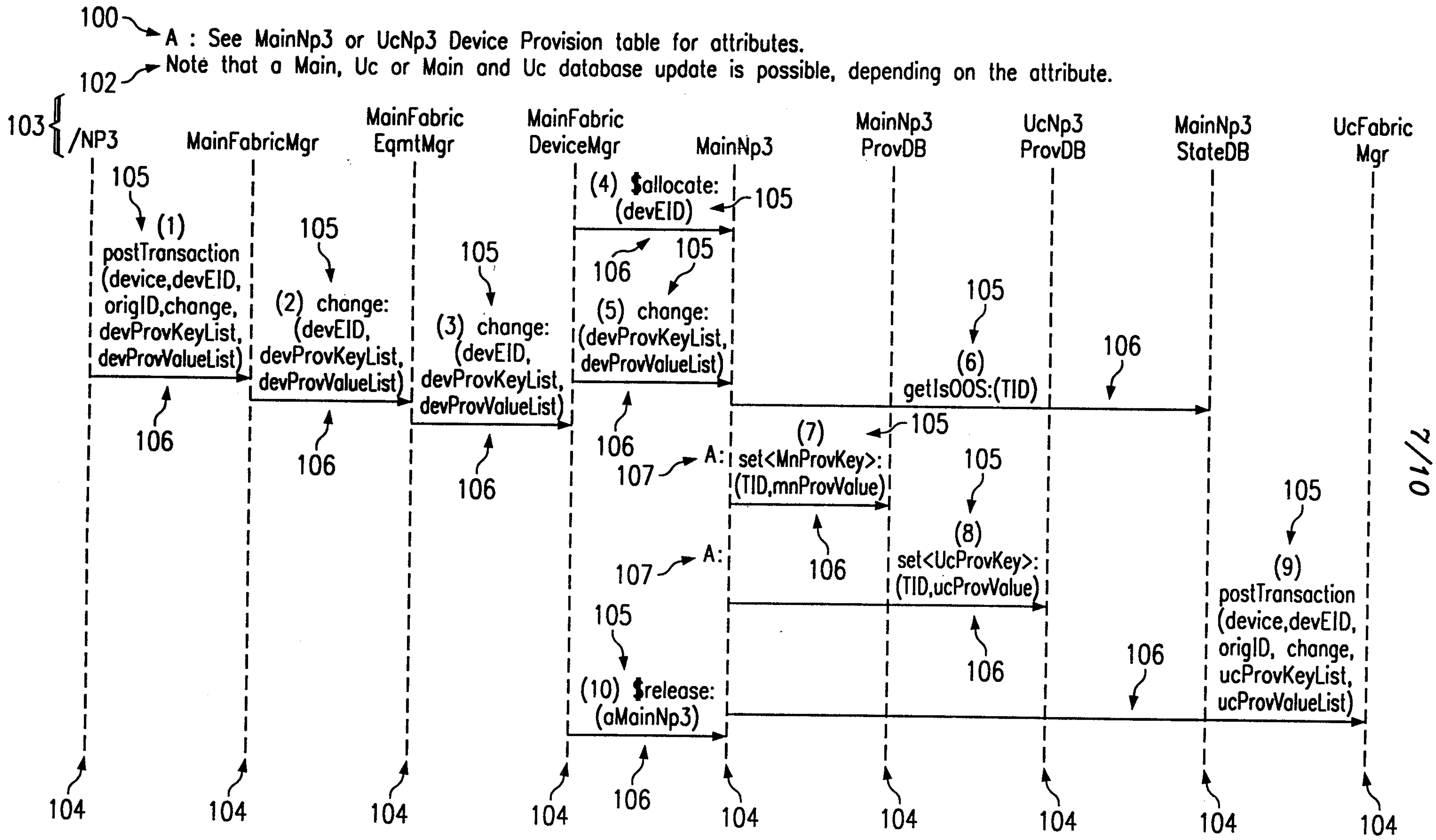


```
400
TITLE::UcQueryDeviceClock Use Case (BB)
NOTE::assume call has clockType=BB
   CLASS::MainDevice
   CLASS::UcFabric\\Mgr
   CLASS::UcFabric\\EqmtMgr
   CLASS::UcTp8
24 CLASS::UcNps1
   CLASS::UcMi
   CLASS::UcSg
   CLASS::Sts1pLfi
   CLASS::MmtgLfi
   CLASS::/Dip
MSG::postTransaction\\(device,destID,\\origID,queryClock,\\
commandResult,\\clockType,devEID,\\deviceClock,lockStatus)
TO::UcFabricMgr
FROM::MainDevice
MSG::postTransaction\\(device,destId,\\origId,queryClock,\\
commandResult,\\clockType,devEID,\\deviceClock,lockStatus)
TO::UcFabricMgr
FROM::MainDevice
MSG::queryClock\\(devEID,clockType,\\deviceClock,\\lockStatus)
TO::UcFabricEqmtMgr
FROM::UcFabricMgr
OPTION::
TO::/Dip
FROM::UcFabricEqmtMgr
COMMENT::if device type is Tp8
TO::UcNps1
FROM::UcFabricEqmtMgr
MSG::queryClock\\(clockType,deviceClock,lockStatus)
TO::UcTp8
FROM::UcFabricEqmtMgr
MSG::queryClock
TO::Sts1pLfi
FROM::UcTp8
MSG::sendWithWait:LCPMemoryRead
TO::/Dip
FROM::UcTp8
OPTIONEND::
OPTION::
401 TO::to class
402 ERROR: Receiver Class does not exist!
403 FROM::from class
```

FIG. 4a

```
404  ERROR: Sender Class does not exist!
     COMMENT::if device
     TO::UcNps1
     FROM::UcFabricEqmtMgr
     MSG::queryClock\\(clockType,deviceClock,lockStatus)
     TO::UcNps1
     FROM::UcFabricEqmtMgr
     MSG::queryClock
     TO::MmtgLfi
400  FROM::UcNps1
     MSG::sendWithWait:LCPMemoryRead
     TO::/Dip
     FROM::UcNps1
     OPTIONEND::
     SPACE::
     SPACE::
     OPTION::
     TO::/Dip
     FROM::UcFabricEqmtMgr
     COMMENT::if device type is Mi
     TO::UcNps1
     FROM::UcFabricEqmtMgr
     MSG::queryClock\\(clockType,deviceClock,lockStatus)
     TO::UcMi
     FROM::UcFabricEqmtMgr
     MSG::isMaster
     TO::UcMi
     FROM::UcMi
     MSG::queryClock
     TO::MmtgLfi
     FROM::UcMi
     MSG::sendWithWait\\:LCPMemoryRead
     TO::/Dip
     FROM::UcMi
     OPTIONEND::
401  OPTION::
     TO::to class
402  ERROR: Receiver Class does not exist!
403  FROM::from class
     ERROR: Sender Class does not exist!
404  COMMENT::if device type is SG
     TO::UcNps1
     FROM::UcFabricEqmtMgr
     MSG::queryClock\\(clockType,deviceClock,lockStatus)
     TO::UcSg
     FROM::UcFabricEqmtMgr
     MSG::queryClock
     TO::MmtgLfi
     FROM::UcSg
     MSG::sendWithWait:LCPMemoryRead
     TO::/Dip
     FROM::UcSg
     OPTIONEND::
```

FIG. 4b


100
A : See MainNp3 or UcNp3 Device Provision table for attributes.
102
Note that a Main, Uc or Main and Uc database update is possible, depending on the attribute.
103
/NP3
MainFabricMgr
MainFabric EqmtMgr
MainFabric DeviceMgr
MainNp3
MainNp3 ProvDB
UcNp3 ProvDB
MainNp3 StateDB
UcFabric Mgr
105
(1) postTransaction (device,devEID, origID,change, devProvKeyList, devProvValueList)
106
105
(2) change: (devEID, devProvKeyList, devProvValueList)
106
105
(3) change: (devEID, devProvKeyList, devProvValueList)
106
(4) $allocate: (devEID)
105
106
105
(5) change: (devProvKeyList, devProvValueList)
106
105
(6) getIsOOS:(TID)
106
107
A:
(7) set<MnProvKey>: (TID,mnProvValue)
105
106
105
(8) set<UcProvKey>: (TID,ucProvValue)
106
107
A:
105
(10) $release: (aMainNp3)
106
106
105
(9) postTransaction (device,devEID, origID, change, ucProvKeyList, ucProvValueList)
104
104
104
104
104
104
104
104
104

101 Main Change NP3 Device Use Case

FIG. 5

FIG. 6a

200	TITLE	Create OTM UC – Nested Repeats		
			space line	
	NOTE			
	CLASS	System Level/Mgt I/F		
	CLASS	System Level/Config Mgr		
	CLASS	System Level/Slot MO		
	CLASS	System Level/OTM Class		
	CLASS	System Level/OTM MO		
	CLASS	System Level/POM		
	MSG	FROM	TO	
	Create OTM	System Level Mgt I/F	System Level Config Mgr	
	Validate & Create	System Level Config Mgr	System Level Slot MO	
	Validate & Create	System Level Slot MO	System Level OTM Class	
	OK	System Level OTM Class	System Level Slot MO	
	Instantiate	System Level Slot MO	System Level OTM MO	
	Write Persistent Data	System Level OTM MO	System Level POM	
	OK<this is a test>	System Level POM	System Level OTM MO	
201	REPEAT	For XX 202		
	OK(parm1,parm2,parm3, parm4,parm5)	System Level OTM MO	System Level Slot MO	
	THIS IS A TEST	System Level OTM MO	System Level Slot MO	
203	REPEATEND			
	OK(parm's)	System Level Slot MO	System Level Config Mgr	
	OK	System Level Config Mgr	System Level Config Mgr	
204	SPACE			
205	OPTION	System Level Mgt I/F 206	System Level POM	207
	COMMENT	FROM	TO	
	This is a test of OPTION processing.	System Level Mgt I/F	System Level Slot MO	
201	REPEAT	For XX 202		
	MSG	FROM	TO	
	Test	System Level Mgt I/F	System Level Config Mgr	
	Test	System Level Config Mgr	System Level POM	
	LABEL	A:		
216	OPTION	System Level Mgt I/F 206	System Level Slot MO	207
	COMMENT	FROM	TO	
	This is second OPTION block.	System Level Mgt I/F	System Level Slot MO	
218	REPEAT	For X 220		
	MSG	FROM	TO	
	This is test 2	System Level Mgt I/F	System Level Slot MO	
	Testing the parser	System Level Mgt I/F	System Level Slot MO	
219	REPEATEND			
217	OPTIONEND			
	Trying to get a second page	System Level Slot MO	System Level POM	
203	REPEATEND			
208	OPTIONEND			
29	This should be on the second page	System Level POM 33	System Level POM	34
209	What's up Doc	System Level Mgt I/F	System Level OTM MO	
	It's a Bird	System Level OTM MO	System Level OTM Class	

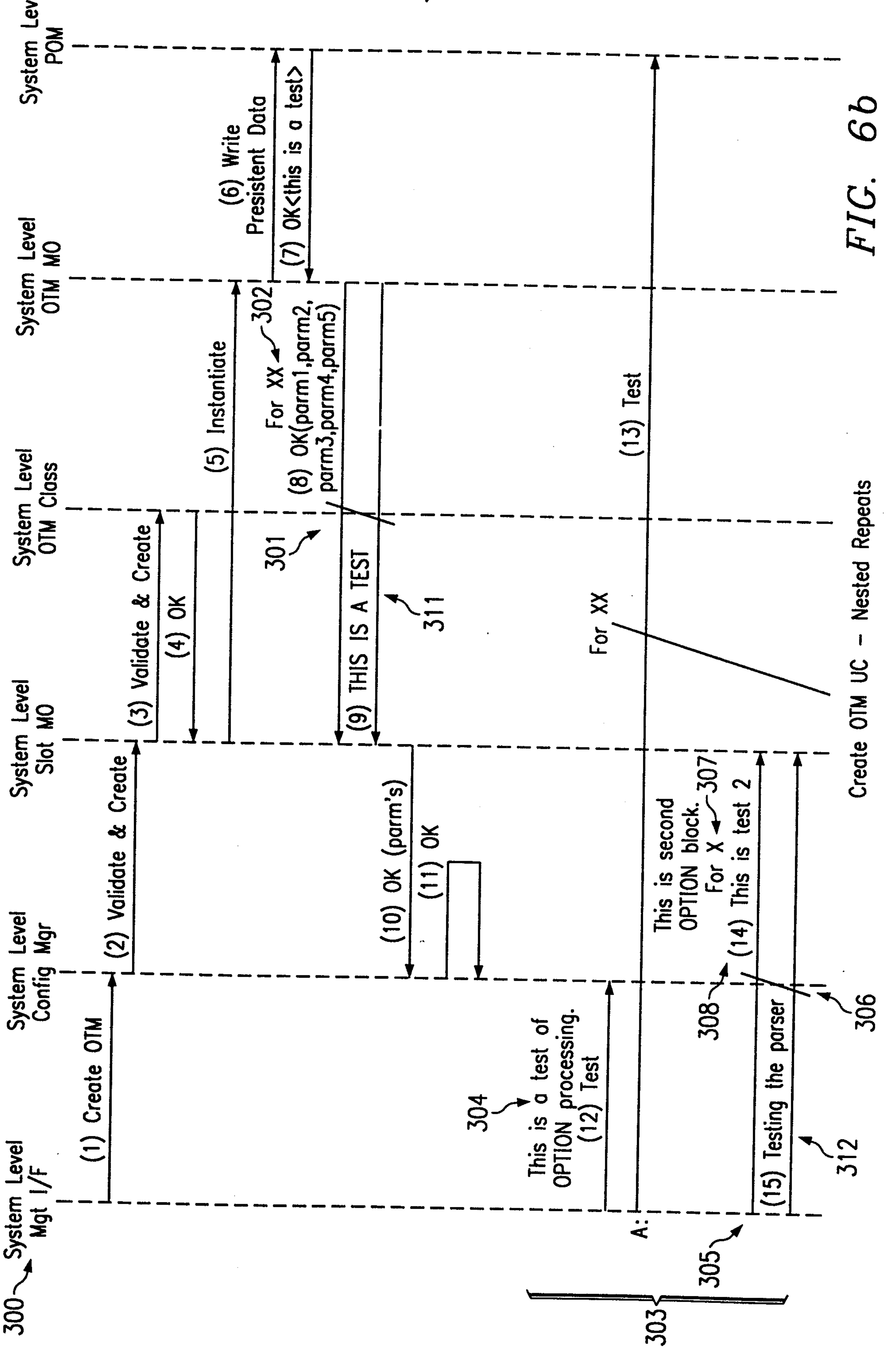
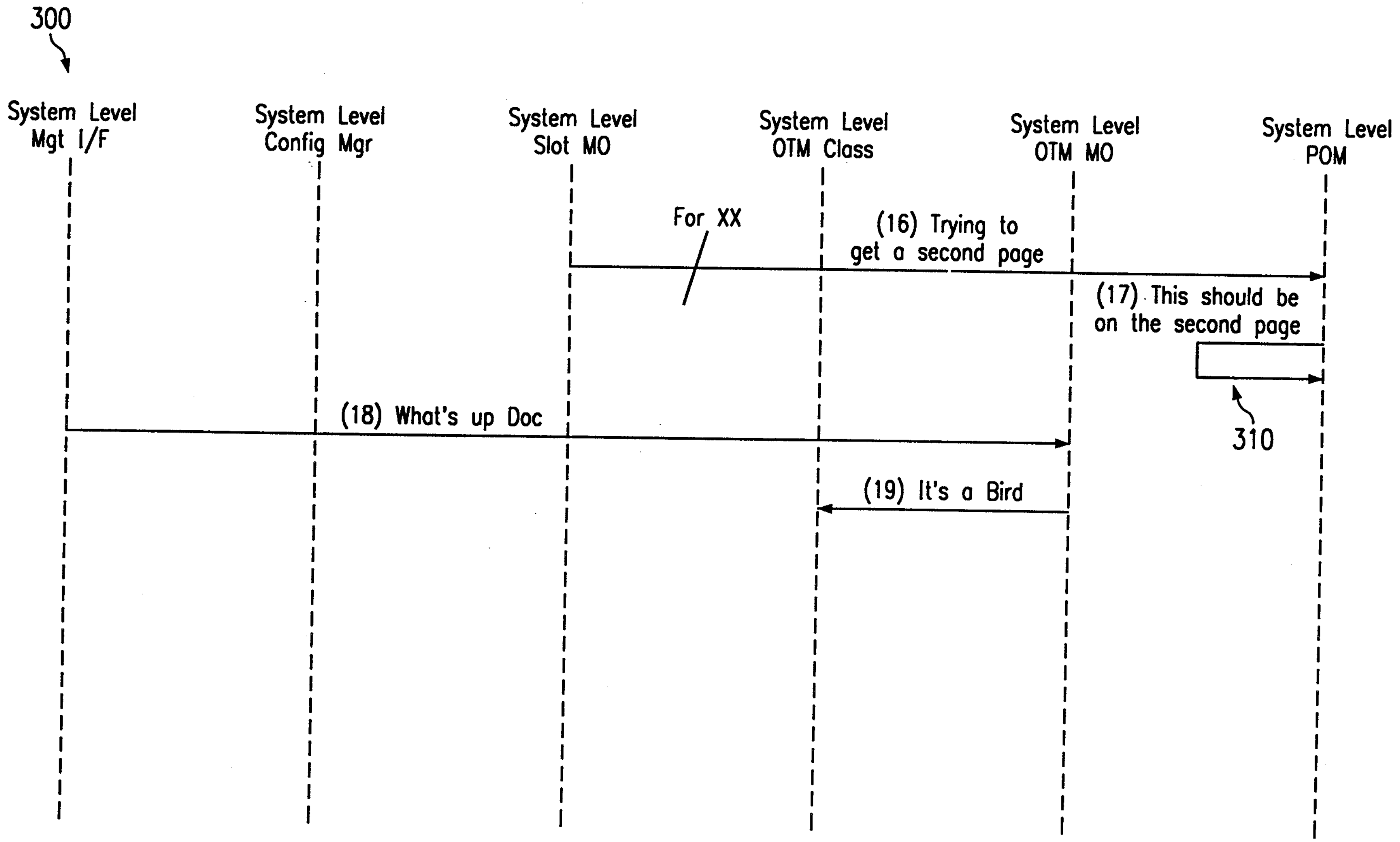

300
System Level Mgt I/F
System Level Config Mgr
System Level Slot MO
System Level OTM Class
System Level OTM MO
System Level POM
(1) Create OTM
(2) Validate & Create
(3) Validate & Create
(4) OK
(5) Instantiate
(6) Write Presistent Data
(7) OK<this is a test>
For XX
302
(8) OK(parm1,parm2, parm3,parm4,parm5)
301
(9) THIS IS A TEST
311
(10) OK (parm's)
(11) OK
304
This is a test of OPTION processing.
(12) Test
A:
(13) Test
For XX
This is second OPTION block.
307
For X
(14) This is test 2
308
306
(15) Testing the parser
312
305
303
Create OTM UC - Nested Repeats

FIG. 6b



Create OTM UC - Nested Repeats

FIG. 6c

INTERNATIONAL SEARCH REPORT

rnational Application No
PCT/US 98/22416

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/44

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

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

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	EP 0 503 944 A (IBM) 16 September 1992 see page 2, line 35 - page 3, line 9 ---	1-26
A	EP 0 727 740 A (NCR INT INC) 21 August 1996 see column 1, line 37 - column 2, line 16 see column 12, line 31 - line 46 ---	1-26
A	US 5 638 539 A (GOTI JUAN C) 10 June 1997 see column 1, line 45 - column 2, line 13 see column 3, line 40 - column 4, line 19 ---	1-26
A	ROMAN G -C ET AL: "A TAXONOMY OF PROGRAM VISUALIZATION SYSTEMS" COMPUTER, vol. 26, no. 12, 1 December 1993, pages 11-24, XP002002622 see the whole document -----	1-26

□ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

° Special categories of cited documents :

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

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

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

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

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

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

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

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

"&" document member of the same patent family

Date of the actual completion of the international search 10 March 1999	Date of mailing of the international search report 17/03/1999
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Authorized officer Brandt, J

INTERNATIONAL SEARCH REPORT

Information on patent family members

ɜrnational Application No
PCT/US 98/22416

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
EP 0503944	A	16-09-1992	JP	2022752 C	26-02-1996
			JP	6195189 A	15-07-1994
			JP	7060374 B	28-06-1995
EP 0727740	A	21-08-1996	JP	8314704 A	29-11-1996
			US	5644728 A	01-07-1997
US 5638539	A	10-06-1997	US	5875330 A	23-02-1999