(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
21 December 2000 (21.12.2000)

PCT

(10) International Publication Number
WO 00/77594 A2

(51) International Patent Classification[7]: G06F

(21) International Application Number: PCT/US00/16113

(22) International Filing Date: 12 June 2000 (12.06.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/329,769 10 June 1999 (10.06.1999) US

(71) Applicant: EC-ENABLER LTD. [US/US]; 300 East 51st Street, #11A, New York, NY 10022 (US).

(72) Inventor: HUGHES, Marvin, E.; Apt. 5L, 27 Claremont Avenue, Mt. Vernon, NY 10550 (US).

(74) Agents: MCKENNEY, Charles, E. et al.; Pennie & Edmonds LLP, 1155 Avenue of the Americas, New York, NY 10036 (US).

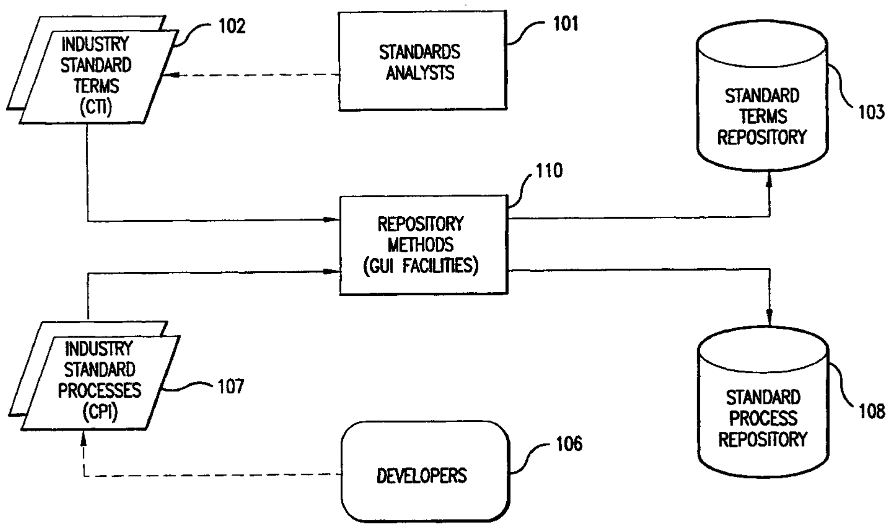
(81) Designated States (*national*): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States (*regional*): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, 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:
— *Without international search report and to be republished upon receipt of that report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: METHOD AND SYSTEM FOR TRANSFERRING INFORMATION



(57) Abstract: A method and system for application-to-application data exchange which provides data conversion from the format of a source application to the format of a target application upon receipt of data by the target application. To achieve compatibility among applications exchanging data, the preferred system uses a standard set of terms and process names for building metadata packets that inform both applications as to their respective data representation. A metadata packet includes a standard name and application specific data format, as well as an optional associated process name. Source metadata provided in connection with source application-specific data enables the conversion of the source format to the format compatible with the target. This method eliminates data conversion at the source application.

WO 00/77594 A2

METHOD AND SYSTEM FOR TRANSFERRING INFORMATION

INVENTOR: MARVIN E. HUGHES

FIELD OF THE INVENTION

This invention relates to transferring data from one computer application to another, including applications using different data formats.

BACKGROUND OF THE INVENTION

Electronic exchange of information is rapidly growing in significance for both businesses and individuals. Although communications infrastructure is available for transporting electronic messages, due to incompatible data formats of many applications, there are significant obstacles to exchanging electronic data dynamically, flexibly and easily. Paper-based transactions still persist even though they are slow and cumbersome, because paper documents are easily understood and available to most people engaged in commerce of any sort. This is not the case with computer data, because computer applications employing different data formats cannot interpret incompatible data.

To unify data formats employed by computer applications, the electronic data interchange (EDI) standard has been developed. This standard, however, has not been widely accepted because it does not effectively facilitate electronic transactions. The EDI standard enforces a specific data format and requires each participant to an electronic transaction to output its data in the format consistent with the standard. To conform to the standard, user's typically need to modify their applications and databases, which are inordinate tasks. To complicate the matter further, when the standard changes it is frequently necessary to alter user applications and convert their databases again to accommodate new features. Thus, the currently available standard is so cumbersome and expensive to implement and use that it does not meet the needs of a broad community of users that require electronic exchange of information.

Also, due to the great expense associated with modifying the existing standard, it is unduly rigid and does not dynamically adapt to the constantly changing commercial environment. Because the standard dictates the types of transactions that can be implemented through electronic data transfers, it severely limits business practices.

Accordingly, there is a need for a system and method of exchanging information among diverse applications that is based on a standard which is readily adaptable to changing commercial environments. Also, there is a need for a system that does not require complex, time consuming and error-prone modifications of the existing applications and databases in order to facilitate information exchange. Furthermore, there is a need for a standard and associated methods and system that can be readily adapted by a broad community of users who desire to exchange information.

SUMMARY OF THE INVENTION

The preferred embodiment of this invention provides a novel method and apparatus for readily and effectively exchanging electronic information between heterogenous applications. The preferred embodiment employs a new standard providing consistent names for data elements (e.g., data structure entries, fields of records, etc.) and associated processes. The standard enables users to define data relationships and specify data manipulation protocols so as to facilitate information exchange without changing existing computer applications, even if they use different data formats. In addition, the preferred embodiment minimizes the need for extensive "setup" time and arrangements before initiating electronic data exchanges among heterogeneous applications. Furthermore, the process-oriented standard of the preferred embodiment is well-suited for implementation using object technology and metadata management of open system architectures.

More specifically, the system and method of the preferred embodiment employ repositories of standard terms and standard process names. The standard terms (also referred to as "standard names") define data elements that are commonly transmitted by applications and the process names define processes commonly used in connection with such data elements, e.g., functions that validate data. For each data element that can be transmitted by an application, the preferred system builds a metadata packet entry that defines the data element such that it is readily "understood" and interpreted by other applications employing a different data format. A collection of such metadata packet entries forms a metadata packet that defines a data structure, a record, or another collection of related data. In the discussion below, all such collections of application's related data may be referred to as data structure.

Metadata packet entries include standard names coupled with application-specific data format definitions. If a given data element defined by a metadata entry is associated with a function (e.g., with a validation procedure), a metadata packet entry may also include such standard function names. The names (also referred to as "terms") in a metadata packet are readily understood by another application having access to the same standard repositories, and because application-specific data formats are defined as part of each metadata packet, incoming data can be readily converted to the format consistent with a recipient (target) application.

The process of building metadata packets is incomparably easier than modifying applications, as customarily done in the prior art, because the existing data structures of the application do not need to change. After metadata packets have been defined and stored for each communicating application, the applications can transfer data without regard for specific data formats used by the recipients.

To transmit information, the source application (i.e., the application that transmits data) sends both actual data elements formatted in accordance with the source - application format and the corresponding one or more metadata packets. (As noted, a metadata packet represents, for example, a data structure or a record). At the target end (e.g., at the system supporting the target application that receives data), the received source data can be readily converted for input to the target application because the source and target metadata packets use the same standard terms and their respective data formats are defined by metadata. In the preferred embodiment, the conversion of the data transmitted by the source application to the format compatible with the target application is target-data-structure driven. That is, target metadata is retrieved and matched with the corresponding source data structure defined by the source metadata. In the event that certain data elements required by the target application are not included in the source data structure defined by the source metadata packet, a default value is supplied during the data conversion. Thus, the resultant converted data is compatible with the target application.

Accordingly, to communicate information, a source application does not perform any data conversion and does not even need to "know" what data format is compatible with the target application. Advantageously, the data structures in the source and target systems remain unchanged, while the metadata provides effective communication among applications.

It is apparent that the method and system of the preferred embodiment provides a dramatic improvement over current practices. The preferred standard uses only standard names and does not impose specific data formats. Due to its simplicity, the standard can dynamically change so as to stay current and consistent with business practices. Users can readily adapt to the changes in the standard by building new metadata packets and without changing their applications software. Another one of many advantages of the preferred method and system is that different applications that use incompatible data representations can communicate without converting data to another representation regardless of specific representations compatible with intended recipients. This mode of communication is possible because the transmitted data is converted at the target end of the data transfer based on the transmitted one or more metadata packets.

It should also be noted that the method and system of the preferred embodiment is not limited to supporting information exchange by remotely located source and target applications, wherein the corresponding source and target systems communicate over a network. It can, for example, be employed within the same system and within the same application. Also, as understood by a person skilled in the art, the preferred method and system are not limited to commercial transactions and can be employed in a vast variety of applications without any limitation to a specific area.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates establishing and maintaining repositories of standard terms and process names.

FIG. 2 illustrates the construction of metadata packets.

FIG. 3 illustrates the construction of metadata packets in further detail.

FIG. 4A and 4B illustrate an example of the metadata packet entries construction process.

FIGS. 5A and 5B illustrate the configuration of software components facilitating exchange of information of the preferred embodiment.

FIG. 6A-D illustrate the operation of the loader and agent manger.

FIG. 7A-C illustrates the functions performed by the process engine during data conversion.

FIGS. 8 and 9 provide examples of how metadata packets are constructed.

FIGS. 10 and 11 illustrate an example of a data transfer in accordance with the preferred embodiment.

FIG. 12 illustrates the manner in which object technology can be used to establish a convention for ordering relationships between transaction types and trading events.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The preferred embodiment employs repositories of standard terms (or names) and standard process names that enable applications having incompatible names and data formats to communicate with each other without converting their data structures to a different format. FIG. 1 illustrates a preferred way of constructing and maintaining these repositories of standard information. Standards analysts (see 101) are individuals who study various sources of relevant information and define and enhance the standard. They are preferably provided with a graphical user interface (GUI) 110 for entering and maintaining the lexicon of the standard. The terms and process names approved by the analysts are then stored in the term and process repositories, illustrated at 103 and 108, respectively. Updated repositories of standard terms and process names are periodically distributed to all the participants that use the preferred method of data exchange.

The standard terms stored in repository 103 reflect frequently used data elements and the process names stored in repository 108 identify the processes commonly used in connection with these data elements. For example, such processes may be used for data validation and manipulation. In deciding which terms to include in the standard, the analysts consider paper and electronic documents commonly used in commerce and other uses of data transfers, event logs, file specifications and other relevant sources. Software developers 106 may identify standard processes and supply their names for inclusion into the process name repository 108 by standards analysts 101. The repositories of standard terms 103 and process names 108 are preferably not linked and, therefore, provide independent collections of reference data. The names selected for the terms and process names of the standard preferably resemble natural language terms reflecting their intended use.

FIG. 2 illustrates how application-specific data structures are represented as metadata packets comprising standard terms and process names coupled with application-specific data formats. The metadata building process is implemented on a computer system, e.g., a personal computer, as a separate program or a collection of programs as understood by a person skilled in the art. To facilitate data exchange among heterogenous applications, each application intending to communicate with other applications should undergo the process of building metadata packets. Notably, the metadata building process does not translate data structures of the applications to a different format; only the names of data elements should be translated to the terms of the standard.

A specific user application is illustrated as 201. Data structures of an application are described, for example, using conventional record/file layouts, database table definitions or using other techniques known in the art. In FIG. 2, application data specification 200 describes the data structures of the application 201. The metadata building process is supported by a graphical user interface (GUI) 210 that facilitates correlating application-specific data elements with the standard terms and process names.

Application data specification 200, which for example can be derived from data names, specific data formats (e.g., lengths) and the overall data structure configuration (e.g., file organization), is entered by a user with an aid of GUI 210, and then stored as application description 204. Thereafter, standard terms and process names from the repositories 103 and 108 are matched with the application-specific definitions so as to construct metadata packets. Interface 210 facilitates the assignment of standard terms and process names to the data elements of the application. As a result, the system supporting this process generates one or more metadata packets comprising standard terms correlated to application-specific data formats and selected standard process names. The resultant metadata packets are stored as illustrated at 205.

FIG. 3 illustrates the process of building metadata packets discussed in connection with FIG. 2 in further detail. At 305 of FIG. 3, application definition module (ADM) presents graphical templates to a user building metadata packets who, first, enters application data specification 200, for example, using a keyboard. At 310, the ADM builds definitions of application-specific data structures on the basis of the user-supplied data and stores them as application description 204. The definitions stored as application description 204 include names of data elements, specifications of data formats (e.g., lengths of fields and the associated offsets), corresponding process names, and may also include other information that defines application data structures as known in the art.

Then, at 320, metadata building module (MBM) displays a template with the previously entered definitions of the application description as well as another template with standard terms and process names stored in the repositories 103 and 108. Preferably, the application description is organized by transaction type of the defined data structures. For example, the transaction type of a data structure can be "Purchase Order" as illustrated in the exemplary data structure 820 of FIG. 8. The standard repositories 103 and 108 also preferably identify transaction types that the standard names may relate to. Accordingly, the metadata building system preferably displays standard terms and process names that generally relate to the transaction type of the currently-displayed description of the application-specific data structure.

The user then assigns selected terms and process names from the standard repositories to the application-specific definitions using graphical prompts as known in the art. As a result, the standard terms and process names represent the lexicon in the particular application. At 330, for each application-specific term in the application definition, the metadata building module constructs an entry of a metadata packet comprising application-specific data specification joined with the corresponding standard terms and optional process names. The metadata packet entries corresponding to data elements of each application-specific data structure are then combined into a metadata packet. The packets are then stored as illustrated at 205.

As noted, the terminology of the standard is preferably selected so that the standard names resemble natural language thereby simplifying the process of matching application-specific and standard terms. As apparent from the above discussion, the standard terms (names) are selected based on the lexicon, without considering application-specific data formats. That is, only the terms used by the application are matched to the standard terms, but application-specific data formats do not need to be converted to another "standard" format. Also, it should be noted that the process of matching application-specific terms to standard terms so as to build metadata packets is not concerned with data structures employed by any intended recipient of information (target application). As understood by a person skilled in the art, the constructed metadata packets can also be employed for computer applications unrelated to electronic data transfer. The metadata discussed herein can, for example, be used for initiating and monitoring remote processing tasks, performing data display and retrieval functions that are currently performed by browsers, as well as for a variety of other applications as understood by a person skilled in the art.

FIGS. 4A and 4B provide an example of the metadata building process discussed above in connection with FIGS. 2 and 3. At 400, at least some of the application-specific data definitions that define data elements are displayed to the user. An example of such definitions is provided at 405. The definition of this example includes a company name "Co. Name," having application-specific data format defined as 70 and 10 and "Part No." formatted in this application as 15 and 5. Note that "Co. Name" and "Part No." are application-specific names and "70, 10" and "15, 5" are application-specific data formats. Next, at 410, the metadata building system selects a list of standard terms based on the transaction type of the application-specific data structure. In this example, these terms are "Company Name" and "Part Number", see 420.

At 425, the user selects the terms from the repository of standard terms as, for example, illustrated as 430. At 435, the user graphically relates the selected standard terms to the corresponding terms in the application definition (see 440). Then, at 445 (FIG. 4B), each selected standard term is used as a part of the corresponding metadata packet entry. The metadata packet entry for the data element relating to "Company Name" is illustrated as 450 with the standard term itself shown at 451. At 455, the standard term data is coupled with the application-specific data definition. That is, data format 452 has been stored in the packet entry 450 so that at this point the metadata packet entry 450 contains a standard term correlated to application-specific data format. Next, at 465 the user optionally selects relevant process names from the repository of standard process names which are presented as a list of process names (see 470). These process names will be invoked by a process engine, described in detail below, of the target application. Although this option is available, it is not necessary to include any process names in any metadata packet element. The optional assignment of process names completes the assembly of this exemplary metadata packet entry. In this example, at 475, the system adds user-selected process names to the packet entry 450 (see 453 and 454). The completed metadata packet entry remote process 450 is then included in the appropriate metadata packet which can be retrieved at any time for inclusion in an electronic data transfer or for any other purpose as understood by a person skilled in the art.

FIGS. 5A and 5B illustrate preferred software configuration supporting data transfer between a source and a target applications. FIG. 5A illustrates the source side and FIG. 5B illustrates the target side of the transfer. In this illustration, each side is a computer system executing the source and target applications respectively. These applications can use incompatible data formats. (As noted, the same methodology can be used for applications that reside on the same system).

Both systems include software components of the preferred embodiment supporting the preferred transfer, receipt and interpretation of data. In this discussion it is assumed that metadata packets have already been built for both applications. Software components illustrated in FIGS. 5A and 5B supporting data transfer and interpretation can be combined into the same process or can be different programs depending on the specifics of a particular implementation as understood by a person skilled in the art. Also, as understood by a person skilled in the art, multiple copies of at least some of the components illustrated in FIGS. 5A and 5B can be employed. Preferably, the software installed at both ends of the transfer include source and target functionality.

In FIG. 5A metadata packets 505 and data structure(s) 510 of the source application are accessed by loader 521 and agent manager 522 illustrated as block 520, to facilitate the packetizing of information to be forwarded via a communications network to the target application. The agent manager software, 522, assures that the correct data has been assembled for the selected function of data transfer. The function refers to the intended use of the data by the target application. For example, the function can be file transfer and the associated data translation, data display, or interaction with a remote process. The agent manager identifies and retrieves the appropriate metadata packets that are needed to describe the data being transmitted for the desired function. The metadata is then stored as shown at 427. The loader 521 records the address information of the target application and retrieves the appropriate data structure(s) of the application 510 for transmission to the target application. In addition, if only a subset of the application data should be transmitted, the loader invokes process engine 500 to isolate the desired subset. The process engine 500 identifies the desired subset using the processing as discussed in connection Fig. 5B depicting the target side. The loader may also invoke third party utility functions 523 to process the data before it is transmitted. The data to be transmitted is stored as illustrated at 525. The loader 521 then combines the metadata packets 427 assembled by the agent manager 522 with the data 525 provided by the application and initiates the communications session. Data encryption and decryption utilities and network routing requests, as known in the art, are also included in the functions supported by the loader 521. One or more files containing metadata packet(s) and the corresponding source data are then transmitted to the target system executing the target application using communication interface 530. The function of the data transmission is also included in the transmitted data. For example, the function can be identified as DT-file transfer and data translation; DP-data display; RP-interaction with a remote process.

The preferred software components executing at the target system are illustrated in connection with FIG. 5B. As noted, software components supporting the preferred data transfer and receipt are preferably included in a single package. Thus, although these software components are shown under different reference numbers in FIG. 5B, they are preferably a different copy of the software which support both the transfer and receipt capabilities.

At the target system, the source data and metadata packets are received at communication interface 535. See FIG. 5B. The received data and metadata, as illustrated at 537 and 540, respectively, are disassembled and stored for further processing by the loader 546. In the capacity of a recipient, the loader 546 and agent manager 547 collectively illustrated as 545 perform additional functions discussed below.

The agent manager of the target system validates the existence of a function supported by the target application for which the data transfer was received. It should be noted that different applications use the received data in different ways. An application may read the received data as a file, or display the data, or use it to interact with a remote process (e.g., to supply parameter/task list to a remote process), or use it for another purpose as known in the art. As noted, this intended use of the data is referred to as the function of the transmitted data. The loader 546 preferably maintains a function queue where it enters the function of incoming data and its storage location. The agent manager retrieves metadata packets (see 550) of the target application that correspond to the received packets on the basis of the transaction type of the transmitted data, and invokes the appropriate portion of the process engine, illustrated as 560, to perform data conversion for the indicated function.

The source data, originating at 510, is converted in accordance with the target metadata specification 550 to the target application data 555. The data conversion process at the target system employs an output-driven mapping process. That is, first the terms in the target application are selected and then matched with the terms employed by the source as discussed in more detail below.

FIGS. 6A-D and 7A-C illustrate the processes performed by the loader, agent manager, and process engine at the receiving end. It should be noted that, as understood by a person skilled in the art, these modules can be implemented as three separate programs, or may be combined into a single program or partitioned in any other way known in the art. Also, as noted, multiple copies of these components can be employed. A person skilled in the art will choose an appropriate configuration for a given implementation.

The processing illustrated in FIG. 6A begins at 600 wherein the target data communications facility (e.g., 535) receives metadata packets and source data structures (see 605, see also, 537 and 540). Upon arrival at the target system, the received metadata packets and associated data structures of the source application are examined by the loader 546 so as to determine that the information has been properly received. Thus, flow of control is transferred to 610 where proper receipt of the data transmission is validated (see 610). If an error is detected, an appropriate message identifying the error is returned to the sender (see 615). The loader module also performs decryption, if necessary, as well as any additional tasks that are needed to properly receive the transmitted data, as known in the art. Each metadata packet received from the source application is then stored in the received metadata directory of the target system (see 620 and 625). The received source data structures are stored in the received data directory at the target system (see 630 and 635 of FIG. 6B). The loader also determines the function of the transmitted data (e.g., file transfer and data translation, display, interaction with a remote process). As noted, the target computer may have various applications that receive data transmitted from external sources. After the loader has determined the function of the received data, this function and the identification of the storage for the received data are entered in the function queue. When the requested function has been identified and added to the queue the loader invokes the agent manager (see 547). As understood by a person skilled in the art, the system may maintain several queues between the loader and agent manager.

Referring to FIG. 6C, the agent manager reads function queue at 660 and retrieves validation criteria for the received data based on its function. The agent manager then retrieves received information from the data directly in accordance with the specification provided in the queue and, at 665, validates that the received information meets the requirements needed to assure proper deployment of resources for further processing. If invalid data has been detected, the source application is informed accordingly by a return message (see 666 - 668). This validation of the received data and metadata is preferably performed upon the receipt of data because the data conversion process can be performed at a later time, not necessarily upon receipt. Thus, if an error is detected, the source has to be notified while it retains information about the particular data transfer.

After the received data has been validated, the agent manager invokes the capabilities of the process engine in accordance with the function of the received data (see 680 and 685) and initiates a session monitor process (see 690) to record statistics and log exceptional activity during the operation of the engine. The execution of the agent manager is then returned to block 660 where it continues to handle new data.

The operation of the process engine for each received metadata packet and the associated application-specific data is illustrated in FIGS. 7A-C. The process engine 600 loads a source metadata packet (see 700), the corresponding target metadata packet (see 705) and the corresponding transmitted source application-specific data (see 710). The corresponding source and target metadata packets are selected based on the source packet transaction type. Then, the process engine performs data conversion by mapping from the target data structure to the received source data structure.

A metadata packet includes one or more entries specifying data elements and optionally, it may also include one or more group level definitions. The group level definitions are file headers and other information of general nature. If they are used, group level definitions appear in the beginning of the packet. Each of the definitions corresponds to one or more entries representing data elements that appear thereafter. The entries representing data elements belonging to a group level definition can be ascertained from the group level definition. It should also be noted that a metadata packet preferably (but not necessarily) includes a transaction type of the packet.

In the discussion below, the processing of the process engine is illustrated assuming that the packets include one or more group level definitions. This processing technique can be modified by a person skilled in the art for other conventions used in packet construction.

At 720 the process engine determines if a given group level definition entry in the target packet exists in the transmitted source metadata. If the definition does not exist, at 725, the default data is provided as target data for the data elements of this group level definition. If the given group level definition entry has been found, the packet entries corresponding to the group level definition are identified and the source data elements (if they exist) are converted based on the metadata. If at 730, a target metadata packet entry does not exist in the source packet, default mapping (735) is performed. That is, default data is provided for the missing data element corresponding to the missing metadata entry. Otherwise, the target packet entry and the corresponding source packet entry are used for mapping the source data element to the target data (see 740). That is, the source data is converted to the target data representation. As a part of the mapping process, the process engine may execute one or more processes specified by metadata as illustrated at 738. In the event of default mapping as discussed above, the default processing functions may also be applied.

More specifically, at 745 the process engine checks if additional metadata packet entries of the target packet belong to the group level definition that is currently being processed. In other words, at 745 it is checked if additional target data elements that have not been processed belong to this group level definition. If so, flow returns to 730 and the next data element corresponding to the next target metadata entry is processed. Otherwise, the system checks at 750 whether additional group level definitions exist and, if so, flow returns to 720 to process the data elements corresponding to the next group level definition entry. Otherwise, the conversion process terminates.

It should be noted that if metadata does not employ group level definition, the metadata entries can be processed sequentially so as to create data elements for input to the target application. In this case, as discussed before, the process is driven by the target metadata so that the target entries are created either by default mapping if the corresponding source elements do not exist or by data conversion from the source data element to the target data element based on the corresponding metadata packet entries.

Because the metadata entries of the target application are considered first, this procedure assures integrity of results, i.e., that all the necessary elements of the target data are specified when the data is provided to the target application. The target-driven execution as discussed herein assures that the preferred method is applicable to a wide range of applications.

FIGS. 8 through 11 provide an example of the operation of the preferred embodiment described generally from a user perspective. User 800 in FIG. 8 wishes to transmit a purchase order represented consistently with a purchase order file specification of his/her source application. Therefore, to use the preferred system and method the user needs to build metadata packets for this purpose.

These packets then would support a multitude of subsequent purchase order transmissions. To build appropriate packets, first, user 800 enters into his/her system application-specific data structures preferably using the graphical user interface (see 210) as discussed above. In this example, the purchase order file specification (i.e., the purchase order data structure) of the source application of the user 800 is illustrated at 820. In this example, data structure 820 includes several fields of data identified by "Field Name" (e.g., "Comp. Name"), and each field is encoded using the source-application-specified format defined as"Offset" and 'Length". As discussed above, the entry of the application data structure is preferably supported by visual and textual prompts. Although this example shows a specific data structure formatted in a specific way as understood by a person skilled in the art, the preferred system and method is not limited to any specific data representation or any specific use of data (e.g., transaction types).

Next, the user assigns standard terms, illustrated as 825, from the repository 103 to the application-specific definitions 820. Preferably, this is done by the system displaying a list of standard terms and a user associates them with the names used in the application preferably with a pointing device. It should be noted that the terms and process names of the preferred standard do not include synonyms so that each term uniquely identifies the corresponding data type, even though synonyms may exist in a data structure of a given application. As discussed above, the displayed standard terms are preferable selected based on the transaction type of the application-specific data structure, which in this example is a purchase order.

To facilitate the terms assignment process, a list of terms commonly found in user environments may be displayed to a user in connection with each standard term. As noted, to assure unambiguous interpretation during the data conversion process, the standard has only one name for each supported data element. For example, "Company Name" (see 825) is the name adapted by the standard. The corresponding data elements used by applications may have different names. For example, in the applications illustrated in connection with FIGS. 8 and 9, "Comp. Name" (see 820) and"Co. Name" (see 915) have the same meaning as "Company Name." The names employed by the applications (see, e.g., 820 and 915) may appear in a synonym list provided in connection with a given standard term, e.g., "Company Name". This list is used as a tool facilitating the correlation of the standard and application-specific names, but, as noted, the standard itself does not have synonyms.

During the process of matching standard and application-specific terms, the user may also assign process names from the standard repository 108 to selected metadata packet entries. An example of such standard process names is illustrated as 828. The resultant metadata entries of this example are illustrated as 830. They form a metadata packet for the purchase order transaction type. The packet entries include standard data names, application-specific data formats and optionally selected standard process names. Thus, application-specific data structure 820 has been represented in the metadata such that it can be readily understood by other applications having access to the standard repositories 103 and 108.

As part of data transfer, the generated metadata packet 830 is then passed to the system network facility generally illustrated on this drawing as 840 by the agent manager and loader as discussed above. The application-specific data of the source application is illustrated in this example as 835 and it is also passed to the network facility by the loader as discussed above. Also, a user of the source application may choose to forward a subset of the output information to the target application. As noted previously, the subset may be produced by invoking the process engine to generate the subset using the steps typically performed in connection with the target application. Also as noted previously, the present method and system can be used for processing data within one system, and not only for the purpose of exchanging data between systems. In this example, the purchase order transaction type associated with the data structure at issue in FIG. 8 belongs to the class ORDER, which includes a pair of transaction types: purchase order and order entry (see FIG. 12 and the associated discussion).

The recipient of the purchase order also has built its metadata packet that specifies the data accepted by an order entry application. As illustrated in FIG. 9, user 910 defined application-specific data structure shown at 915 using the procedure discussed above. The standard terms and process names used for order entry are illustrated at 920 and 923. In this example, using the procedure discussed above, the target system built metadata packet 925.

FIG. 10 illustrates metadata packets for the source (purchase order) 830 and the target (order entry) 925. As apparent from this illustration, the commonality of terms, not formats, facilitates successful data conversion. Thus, using the output-driven method (i.e., the method where target metadata is considered first), as discussed above, the process engine converts the received purchase order data structure in the format of the source application into the structure compatible with the order entry input of the target application. As a result, incompatible data was transmitted, received and used by the target application without any translation performed by the source system.

As summarized in FIG. 11, the source (sender) application 1100 and the target (receiver) application 1110 were enabled to consummate an electronic delivery of data without disturbing their respective native environments. Thus, the preferred methods and system facilitates conducting a trading event using a standard language 1115 and performing the requisite data validation and manipulation processes, see, e.g., 1120 and 1130. The process engine has converted the source data structure into the data representation required by the target application. Metadata received by the target application can be interpreted so as to facilitate file translation (see 1145), deliver information to a web site (see 1150), initiate and direct remote processing operations (see 1155) or perform any other functions conceived by application developers.

It should be noted that purely lexical qualities of the preferred standard simplify the exchange and proper interpretation of data. Other than consistency of vocabulary, there are no other requirements with respect to transmitted data so that, for example, format, structure, context, and manipulation remain the properties of application environments. That is, data representation of the application environment is not effected. Thus, in the preferred embodiment, it is possible to readily envelope, transport and transform information between diverse applications.

FIG. 12 provides an example of how object technology can facilitate the implementation of the preferred embodiment. Electronic transactions have structural elements that can be defined in object-oriented terms. An example of such an object definition is as follows:

Transaction Events ----> Object Class

-Standard Order

Transaction Type ----> Object

- Purchase Order/Order Entry ----> Object Property

- Buyer/Seller ----> Object Property

- Sender/Receiver ----> Object Property

- Source/Target ----> Object Property

- National Language ----> Object Property

Transaction Process ----> Object Method

A transaction set describes transaction types that are used to conduct a specific event between trading partners. In the example provided above, the purchase order issued by the system of FIG. 8 and the order entry data generated by the system of FIG. 9 are complementary components of the transaction set, Standard Order. As discussed above, the properties associated with transaction types provide a consistent definition of the trading relationship across a variety of trading events. Functions and operations performed on transaction types correspond to the application of methods to objects. Data elements (fields, character strings, bit maps, etc.) within transaction types are processed by the process engine and/or by external methods invoked by it.

As understood by a person skilled in the art, the terminology of this specification should be interpreted broadly. For example, the term "data structure" should not be construed as a data structure of a specific language or system because it generally relates to any collection of items of information (e.g., data elements). Similarly, "metadata" generally relates to any data describing other data as understood from the previous discussion. It is also understood that a metadata packet broadly defines a collection of information (e.g., a data structure) and each entry in the packet describes an item of information in such a collection. The data formats of various application data structures and the like should also be construed broadly as any data representations as understood by a person skilled in the art. Other terminology employed herein (for example, applications, process, system, transaction type, communication, function, etc.) should also be interpreted broadly as understood by a person skilled in the art.

The present invention is not to be limited in scope by the specific embodiments described herein. Indeed, modifications of the invention in addition to those described herein will become apparent to those skilled in the art from the foregoing description and accompanying figures. Doubtless, numerous other embodiments can be conceived that would not depart from the teaching of the present invention, whose scope is defined by the following claims.

CLAIMS:

1. A method of providing data from a source computer application to a target computer application comprising:

receiving source data having a representation consistent with the source application at a target system;

receiving a source metadata packet corresponding to the source data at the target system, wherein the metadata packet includes at least one entry comprising a standard name corresponding to at least one name used by the source application and a definition of a related data representation used by the source application;

retrieving a target metadata packet; and

converting the source data to a representation compatible with the target application based on the source and the target metadata packets.

2. The method of claim 1 wherein the step of converting comprises examining an entry in the target metadata packet before the corresponding entry in the source metadata packet is examined.

3. The method of claim 1 further comprising using default data if there is no source metadata packet entry that corresponds to the target metadata packet entry.

4. The method of claim 1 wherein the source metadata packet further comprises at least one process name selected from a repository of standard process names.

5. The method of claim 4 wherein the step of converting further comprises executing a process corresponding to the process name.

6. A method of building a metadata packet for a computer application comprising:

storing application definition comprising application-specific names and application-specific data formats;

matching the application-specific names in the definition to standard names; and

storing one or more metadata packet entries each of which comprises a name selected from the standard names and one of the application-specific data formats.

7. The method of claim 6 further comprising storing at least one standard process name as part of at least some of the metadata packet entries.

8. The method of claim 6 wherein the step of matching comprises using a graphical user interface.

9. The method of claim 6 further comprising displaying suggested names that may correspond to one of the standard names.

10. A method of converting source data, provided by a source application, for input to a target application comprising:

retrieving a target metadata packet comprising at least one standard name;

identifying a source metadata packet, comprising at least one standard name, corresponding to the target metadata packet;

converting the source data defined by the source metadata packet to a target representation defined by the target metadata packet so as to obtain input data; and

if an entry in the metadata target packet does not exist in the source metadata packet, providing default data as a part of the input data.

11. The method of claim 10 further comprising executing at least one process identified by a standard process name in the source metadata packet.

12. The method of claim 10 wherein the source and target applications are executing on different computers communicating over a network.

13. The method of claim 10 wherein the source and target applications are executing on the same computer.

14. Computer memory storing a metadata packet useful for data conversion at a target end of data transmission, comprising:

standard data name stored as part of the packet in the memory;

representation of an application-specific data format stored as part of the packet in the memory; and

at least one standard process name stored as part of the packet in the memory.

15. The memory of claim 14 wherein the packet comprises a plurality of entries.

16. The memory of claim 15 wherein at least some of the entries represent data elements.

17. The memory of claim 16 wherein at least some of the entries represent group level definitions.

18. A method of receiving and interpreting source data for a target application comprising:

receiving source data from a source application;

receiving source metadata relating to the source data;

receiving a function of the source data transmission;

retrieving a target metadata in accordance with the function of the source data transmission; and converting the source data to a target data based on the source and the target metadata.

19. The method of claim 18 wherein the function is file transfer.

20. The method of claim 19 wherein a file formatted in accordance with the target application is generated as a result of the step of converting.

21. The method of claim 18 wherein the function is a display of the received data.

22. The method of claim 18 further comprising receiving the source data over the Internet at a remote computer and displaying the target data at the remote computer.

23. The method of claim 18 wherein the function is controlling or initiating a remote process.

24. The method of claim 18 wherein the step of converting includes generating a visual representation of the received source data consistent with the target application.

25. The method of claim 23 wherein the step of converting includes generating parameters/task list for input to the target application and wherein the target application is a remote process.

26. A system for providing data from a source computer application to a target computer application comprising:

means for receiving source data having a representation consistent with the source application at a target system;

means for receiving a source metadata packet corresponding to the source data at the target system, wherein the metadata packet includes at least one entry comprising a standard name corresponding to at least one name used by the source application and a definition of a related data representation used by the source application;

means for retrieving a target metadata packet; and

means for converting the source data to a representation compatible with the target application based on the source and the target metadata packets.

27. The system of claim 26 wherein the means for converting comprises means for examining an entry in the target metadata packet before the corresponding entry in the source metadata packet is examined.

28. The system of claim 25 further comprising means for using default data if there is no source metadata packet entry that corresponds to the target metadata packet entry.

29. The system of claim 25 wherein the source metadata packet further comprises at least one process name selected from a repository of standard process names.

30. The system of claim 29 wherein the means for converting further comprises means for executing a process corresponding to the process name.

31. A system of building a metadata packet for a computer application comprising:

means for storing application definition comprising application-specific names and application-specific data formats;

means for matching the application-specific names in the definition to standard names; and

memory for storing one or more metadata packet entries each of which comprises a name selected from the standard names and one of the application-specific data formats.

32. The system of claim 31 further comprising memory for storing at least one standard process name as part of at least some of the metadata packet entries.

33. The system of claim 31 wherein the means for matching includes a graphical user interface.

34. The system of claim 31 further comprising means for displaying suggested names that may correspond to one of the standard names.

35. A system for converting source data, provided by a source application, for input to a target application comprising:

means for retrieving a target metadata packet comprising at least one standard name;

means for identifying a source metadata packet, comprising at least one standard name, corresponding to the target metadata packet; and

means for converting the source data defined by the source metadata packet to a target representation defined by the target metadata packet so as to obtain input data, wherein if an entry in the metadata target packet does not exist in the source metadata packet, the means for converting provides default data as a part of the input data.

36. The system of claim 35 further comprising means for executing at least one process identified by a standard process name in the source metadata packet.

37. The system of claim 35 wherein the source and target applications are executing on different computers communicating over a network.

38. The system of claim 35 wherein the source and target applications are executing on the same computer.

39. A system for receiving and interpreting source data for a target application comprising:

means for receiving source data from a source application;

means for receiving source metadata relating to the source data;

means for receiving a function of the source data transmission;

means for retrieving a target metadata in accordance with the function of the source data transmission; and

means for converting the source data to a target data based on the source and the target metadata.

40. The system of claim 39 wherein the function is file transfer.

41. The system of claim 39 wherein the function is a display of the received data.

42. The system of claim 39 wherein the function is controlling or initiating a remote process.

43. The system of claim 39 further comprising means for receiving the source data and metadata over the Internet at a remote computer for display at the remote computer.

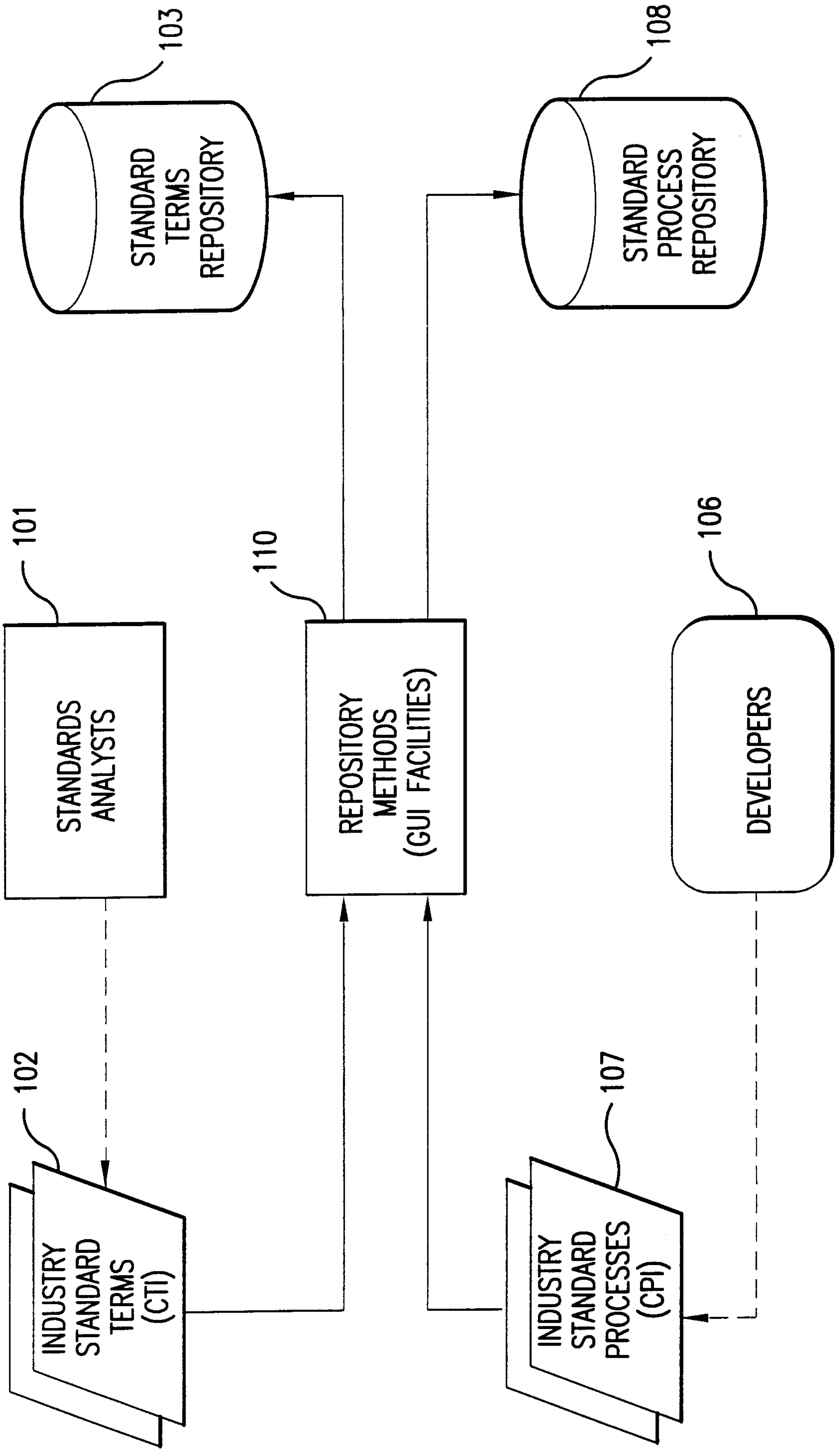
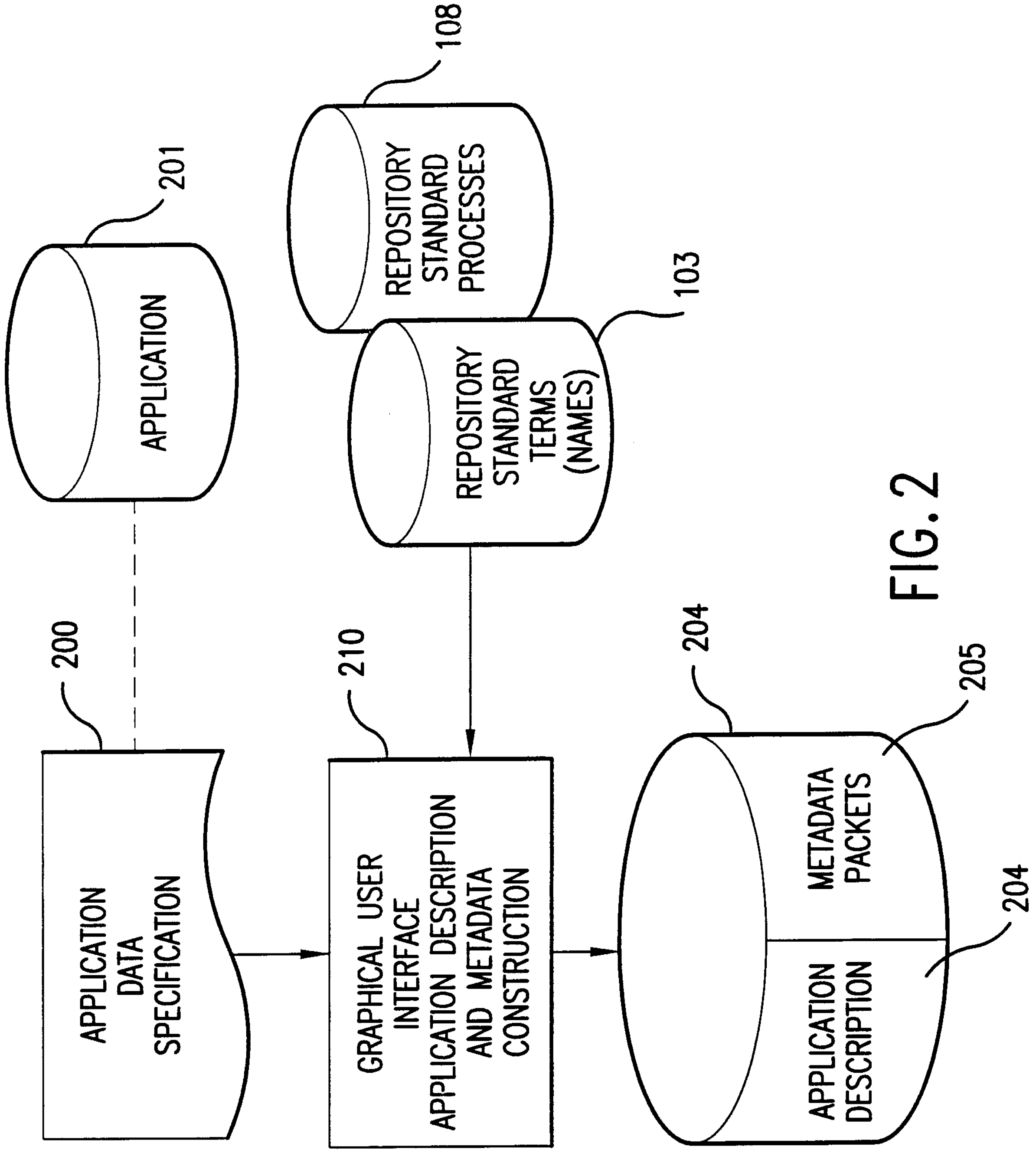
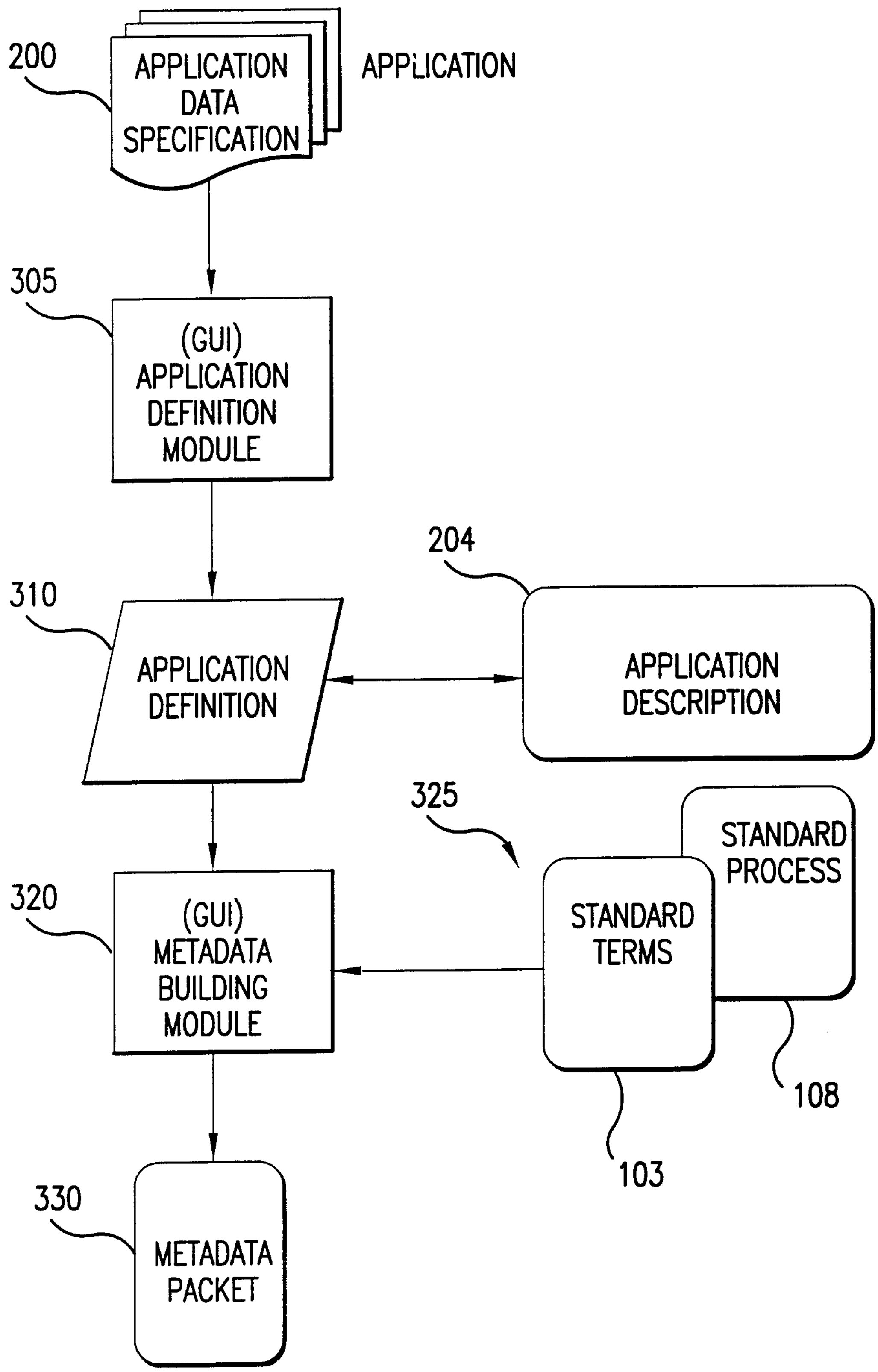

103
STANDARD TERMS REPOSITORY
108
STANDARD PROCESS REPOSITORY
101
STANDARDS ANALYSTS
110
REPOSITORY METHODS (GUI FACILITIES)
106
DEVELOPERS
102
INDUSTRY STANDARD TERMS (CTI)
107
INDUSTRY STANDARD PROCESSES (CPI)

FIG. 1



APPLICATION
201
APPLICATION DATA SPECIFICATION
200
GRAPHICAL USER INTERFACE APPLICATION DESCRIPTION AND METADATA CONSTRUCTION
210
REPOSITORY STANDARD PROCESSES
108
REPOSITORY STANDARD TERMS (NAMES)
103
204
METADATA PACKETS
205
APPLICATION DESCRIPTION
204

FIG. 2



200
APPLICATION DATA SPECIFICATION
APPLICATION
305
(GUI) APPLICATION DEFINITION MODULE
310
APPLICATION DEFINITION
204
APPLICATION DESCRIPTION
325
STANDARD PROCESS
STANDARD TERMS
320
(GUI) METADATA BUILDING MODULE
108
103
330
METADATA PACKET

FIG. 3

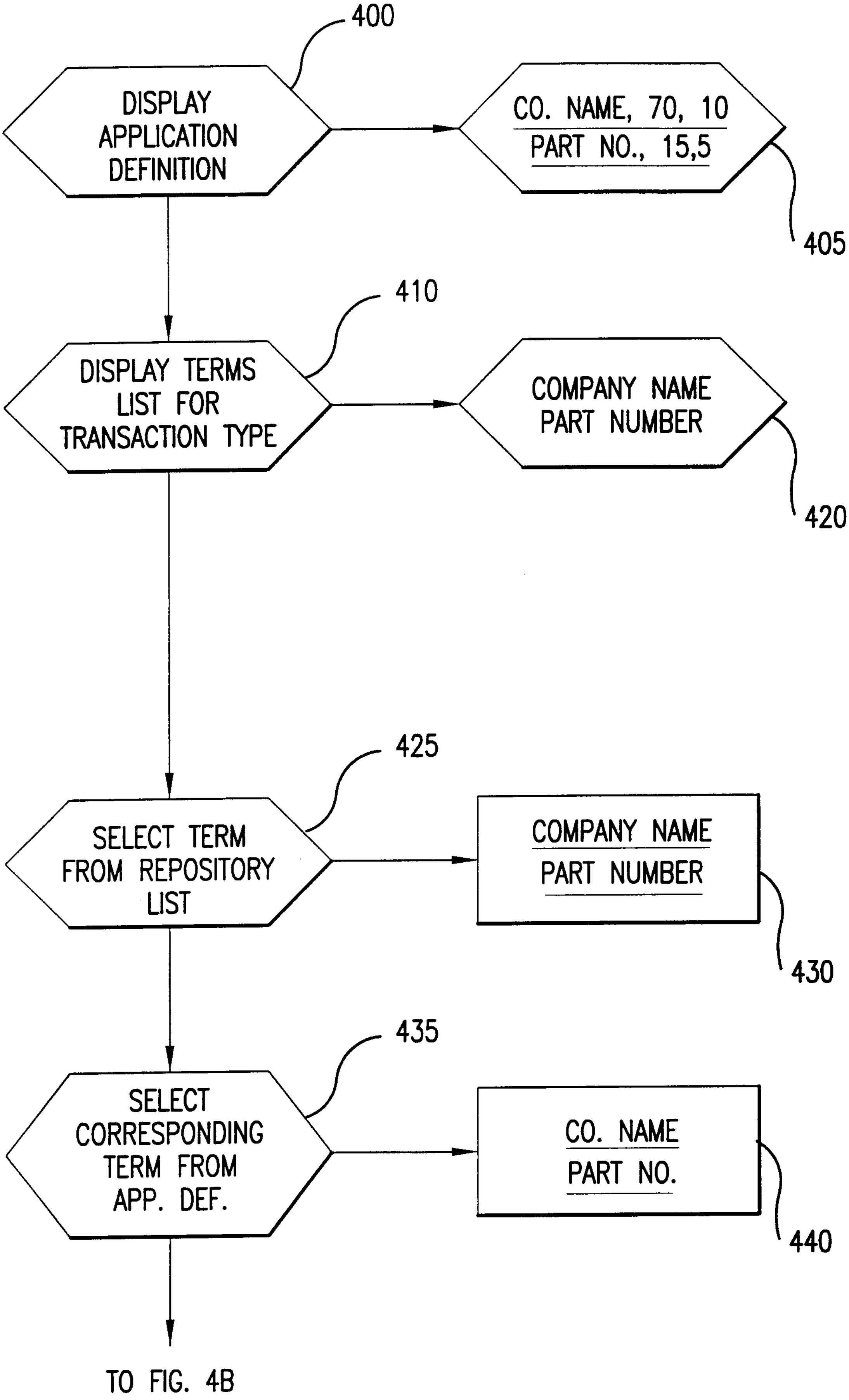

400
DISPLAY APPLICATION DEFINITION
CO. NAME, 70, 10
PART NO., 15,5
405
410
DISPLAY TERMS LIST FOR TRANSACTION TYPE
COMPANY NAME
PART NUMBER
420
425
SELECT TERM FROM REPOSITORY LIST
COMPANY NAME
PART NUMBER
430
435
SELECT CORRESPONDING TERM FROM APP. DEF.
CO. NAME
PART NO.
440
TO FIG. 4B

FIG. 4A

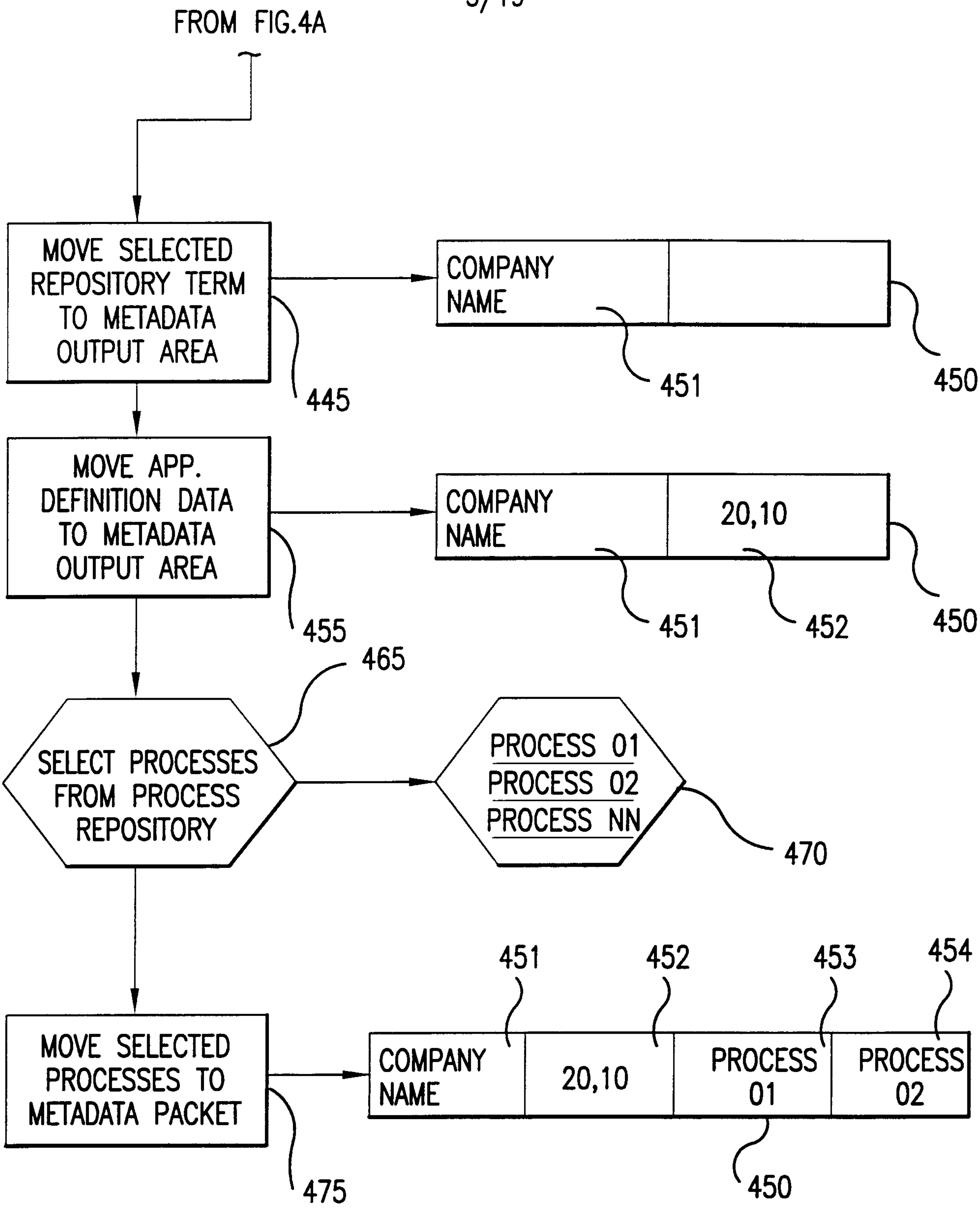

FROM FIG.4A
MOVE SELECTED REPOSITORY TERM TO METADATA OUTPUT AREA
445
COMPANY NAME
451
450
MOVE APP. DEFINITION DATA TO METADATA OUTPUT AREA
455
COMPANY NAME
20,10
451
452
450
465
SELECT PROCESSES FROM PROCESS REPOSITORY
PROCESS 01
PROCESS 02
PROCESS NN
470
451
452
453
454
MOVE SELECTED PROCESSES TO METADATA PACKET
475
COMPANY NAME
20,10
PROCESS 01
PROCESS 02
450

FIG. 4B

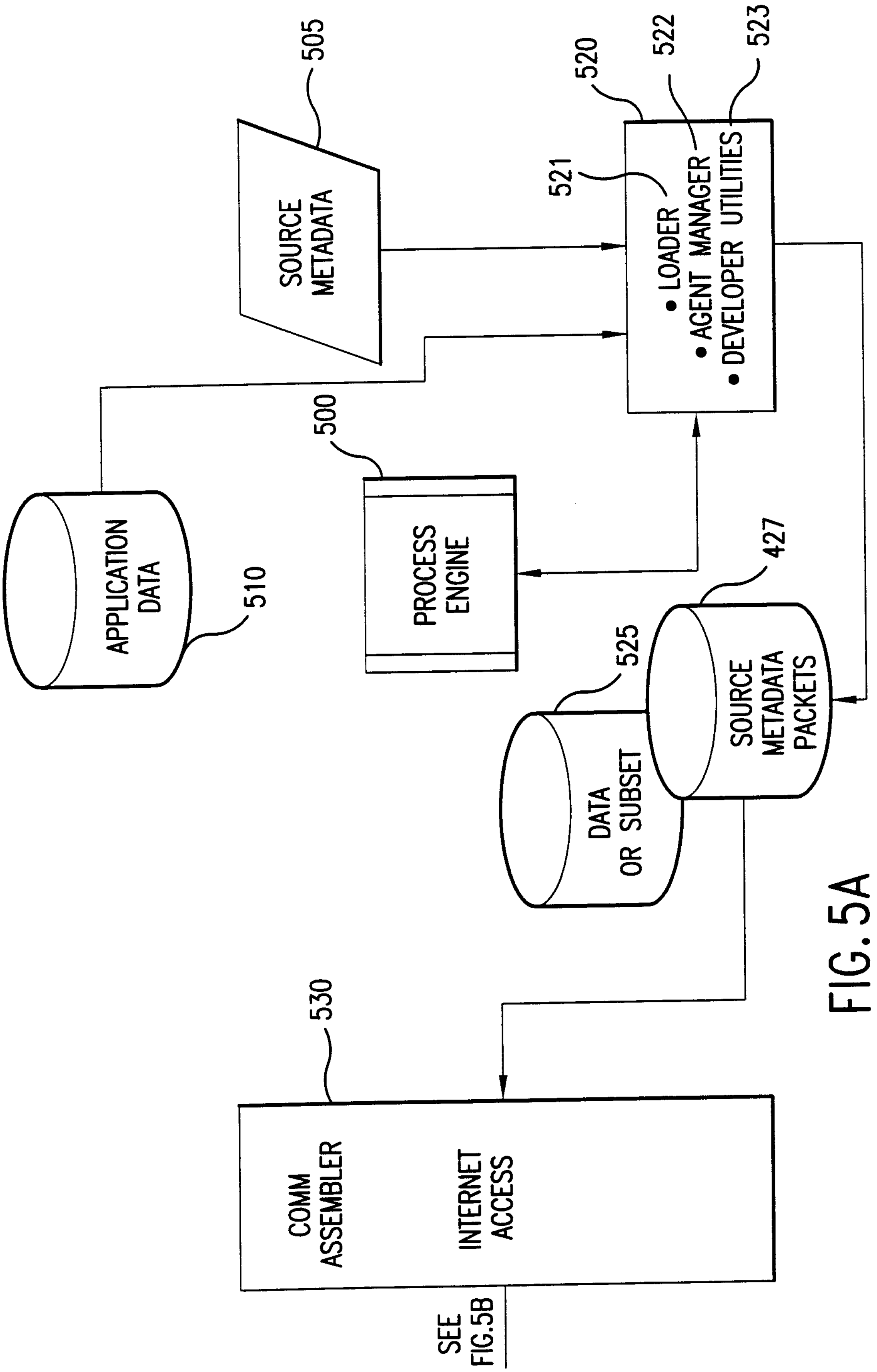

SOURCE METADATA
505
520
522
523
521
• LOADER
• AGENT MANAGER
• DEVELOPER UTILITIES
500
PROCESS ENGINE
APPLICATION DATA
510
427
525
SOURCE METADATA PACKETS
DATA OR SUBSET
530
COMM ASSEMBLER
INTERNET ACCESS
SEE FIG.5B

FIG. 5A

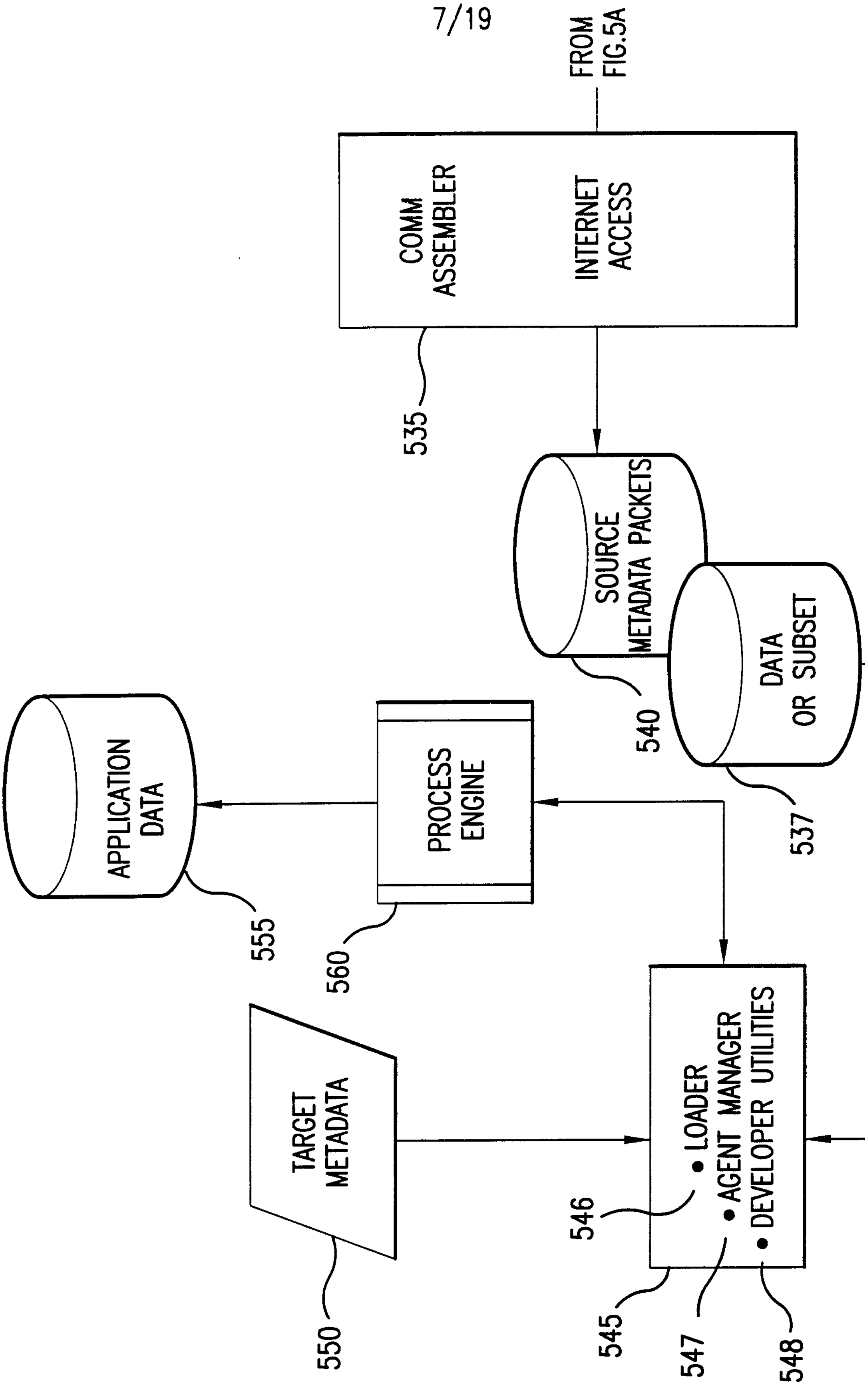

FROM FIG.5A
COMM ASSEMBLER
INTERNET ACCESS
535
SOURCE METADATA PACKETS
540
DATA OR SUBSET
537
APPLICATION DATA
555
PROCESS ENGINE
560
TARGET METADATA
550
• LOADER
• AGENT MANAGER
• DEVELOPER UTILITIES
546
545
547
548

FIG. 5B

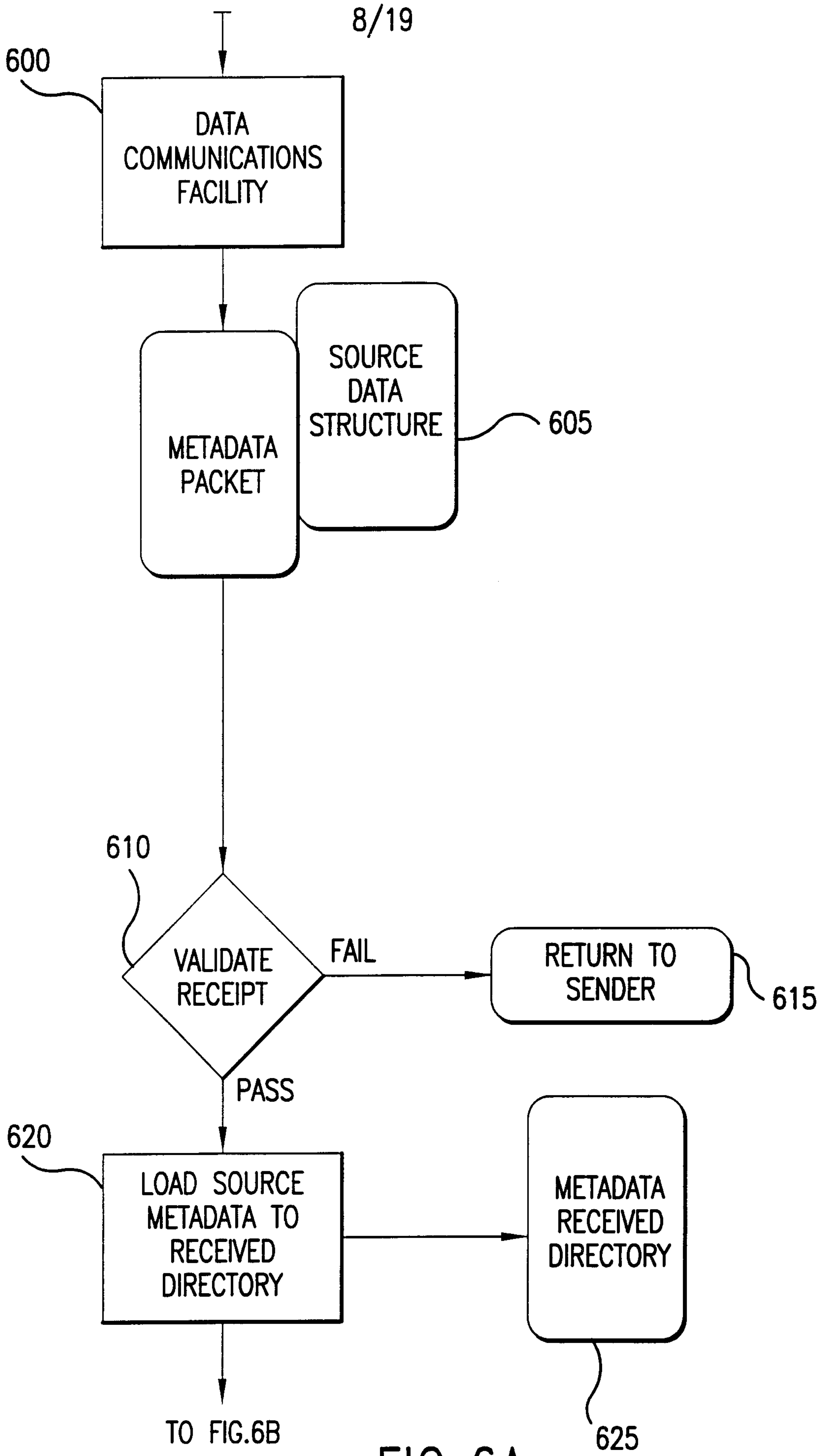

600
DATA COMMUNICATIONS FACILITY
METADATA PACKET
SOURCE DATA STRUCTURE
605
610
VALIDATE RECEIPT
FAIL
RETURN TO SENDER
615
PASS
620
LOAD SOURCE METADATA TO RECEIVED DIRECTORY
METADATA RECEIVED DIRECTORY
TO FIG.6B
625

FIG. 6A

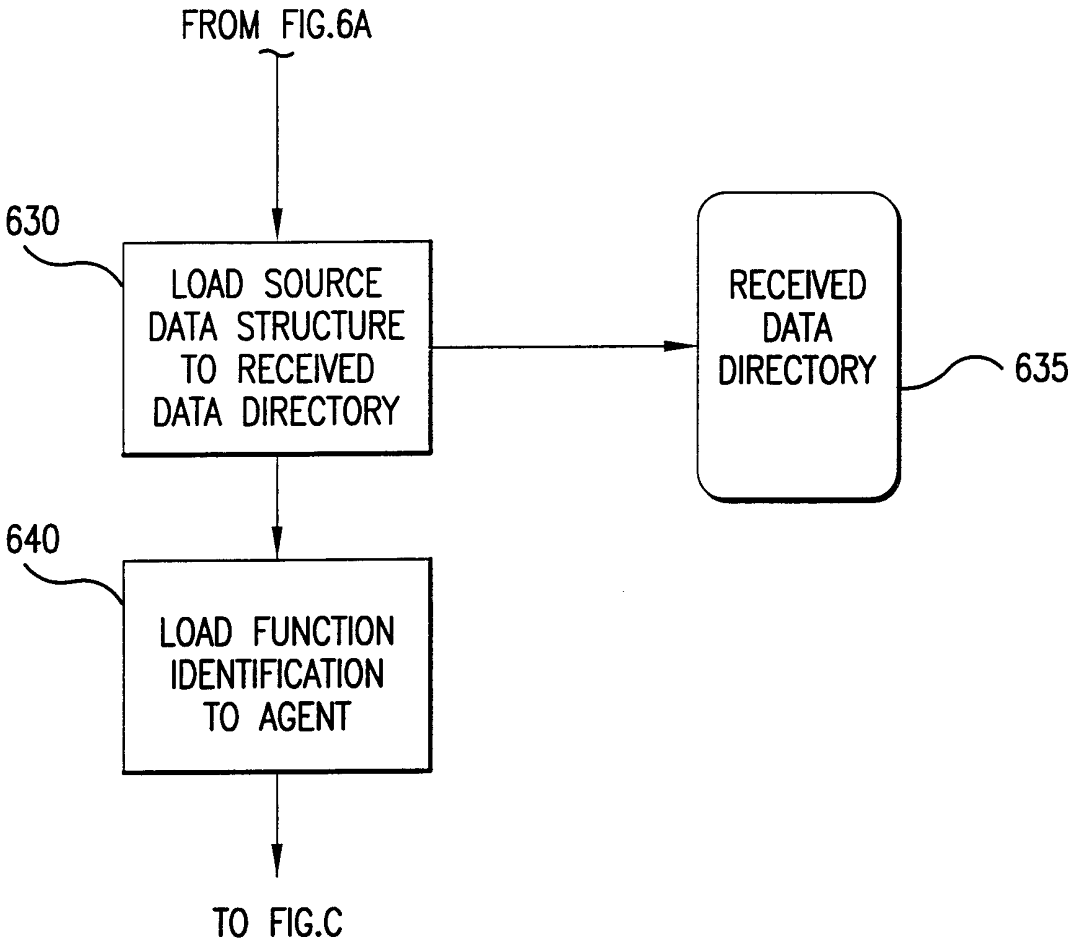

FROM FIG.6A
630
LOAD SOURCE DATA STRUCTURE TO RECEIVED DATA DIRECTORY
RECEIVED DATA DIRECTORY
635
640
LOAD FUNCTION IDENTIFICATION TO AGENT
TO FIG.C

FIG. 6B

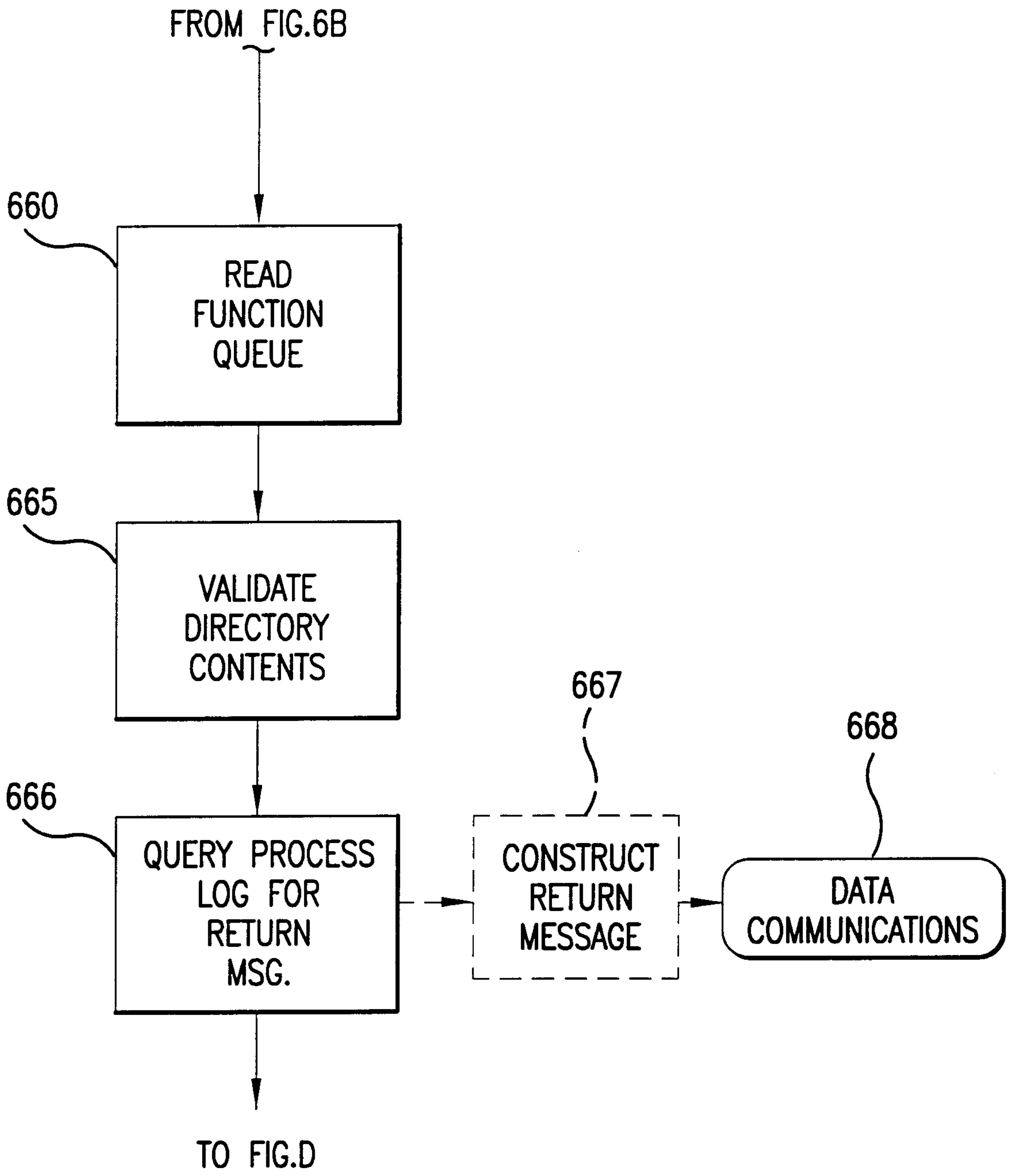

FROM FIG.6B
660
READ FUNCTION QUEUE
665
VALIDATE DIRECTORY CONTENTS
666
QUERY PROCESS LOG FOR RETURN MSG.
667
CONSTRUCT RETURN MESSAGE
668
DATA COMMUNICATIONS
TO FIG.D

FIG. 6C

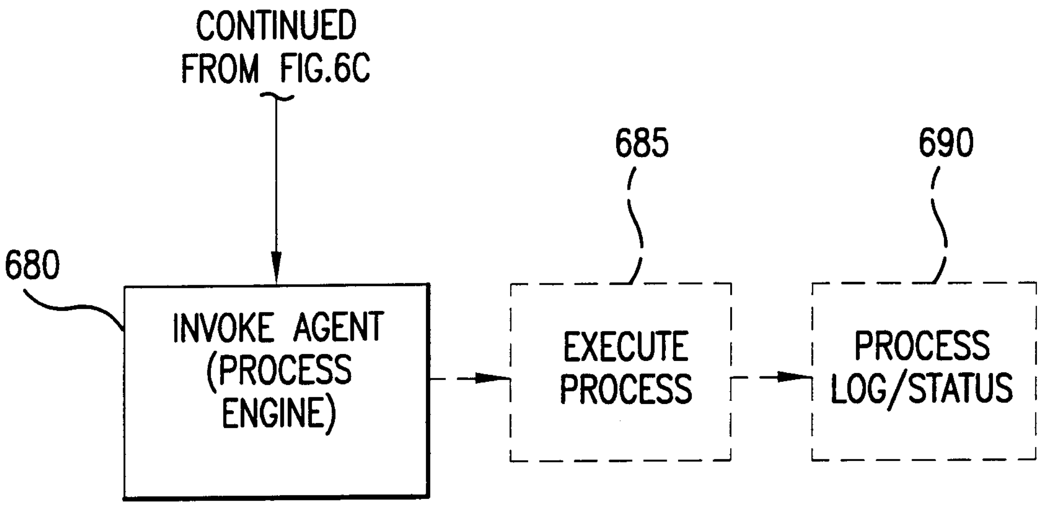

CONTINUED FROM FIG.6C
680
INVOKE AGENT (PROCESS ENGINE)
685
EXECUTE PROCESS
690
PROCESS LOG/STATUS

FIG. 6D

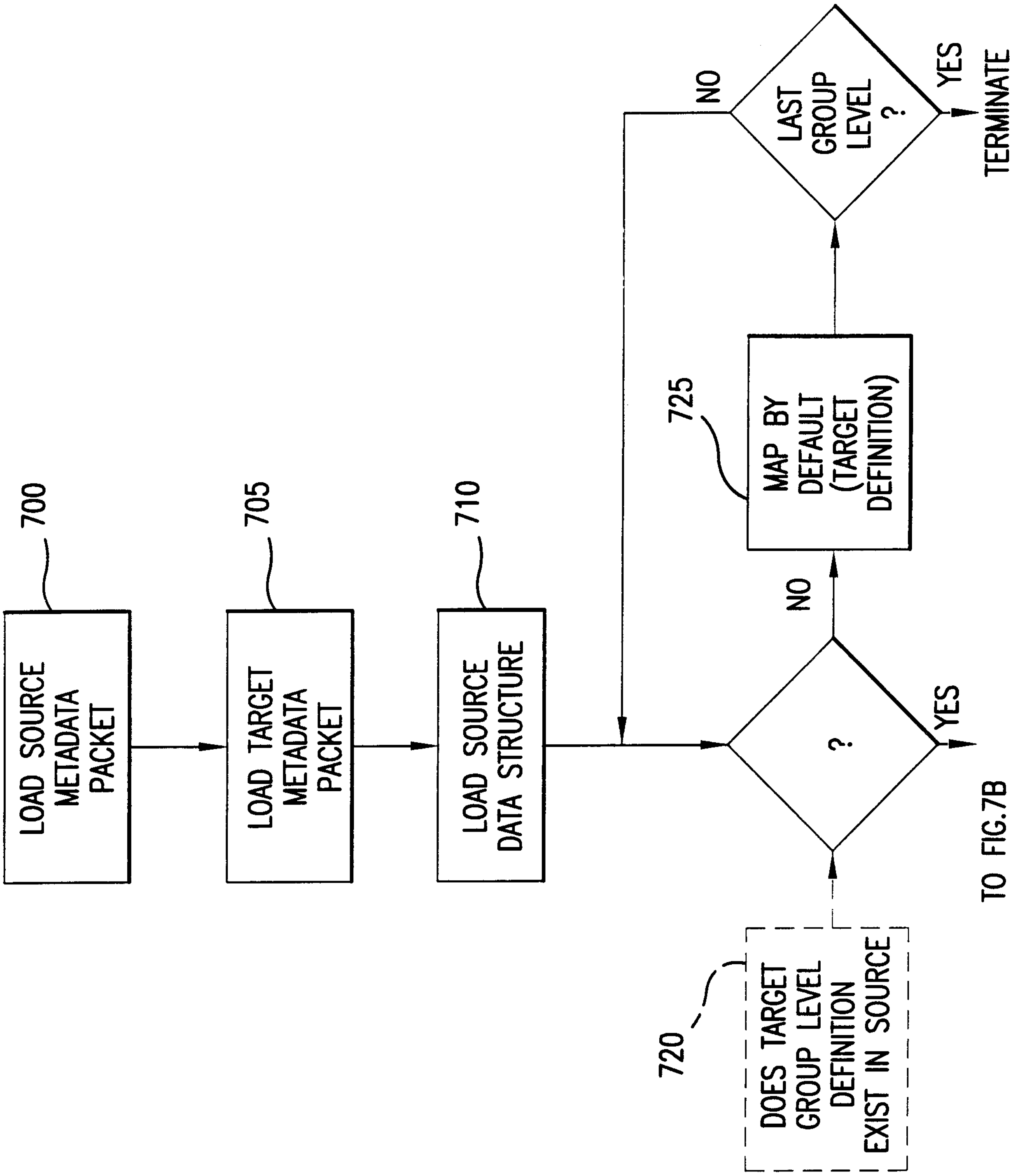

700
LOAD SOURCE METADATA PACKET
705
LOAD TARGET METADATA PACKET
710
LOAD SOURCE DATA STRUCTURE
720
DOES TARGET GROUP LEVEL DEFINITION EXIST IN SOURCE
?
YES
TO FIG.7B
NO
725
MAP BY DEFAULT (TARGET DEFINITION)
LAST GROUP LEVEL ?
NO
YES
TERMINATE

FIG. 7A

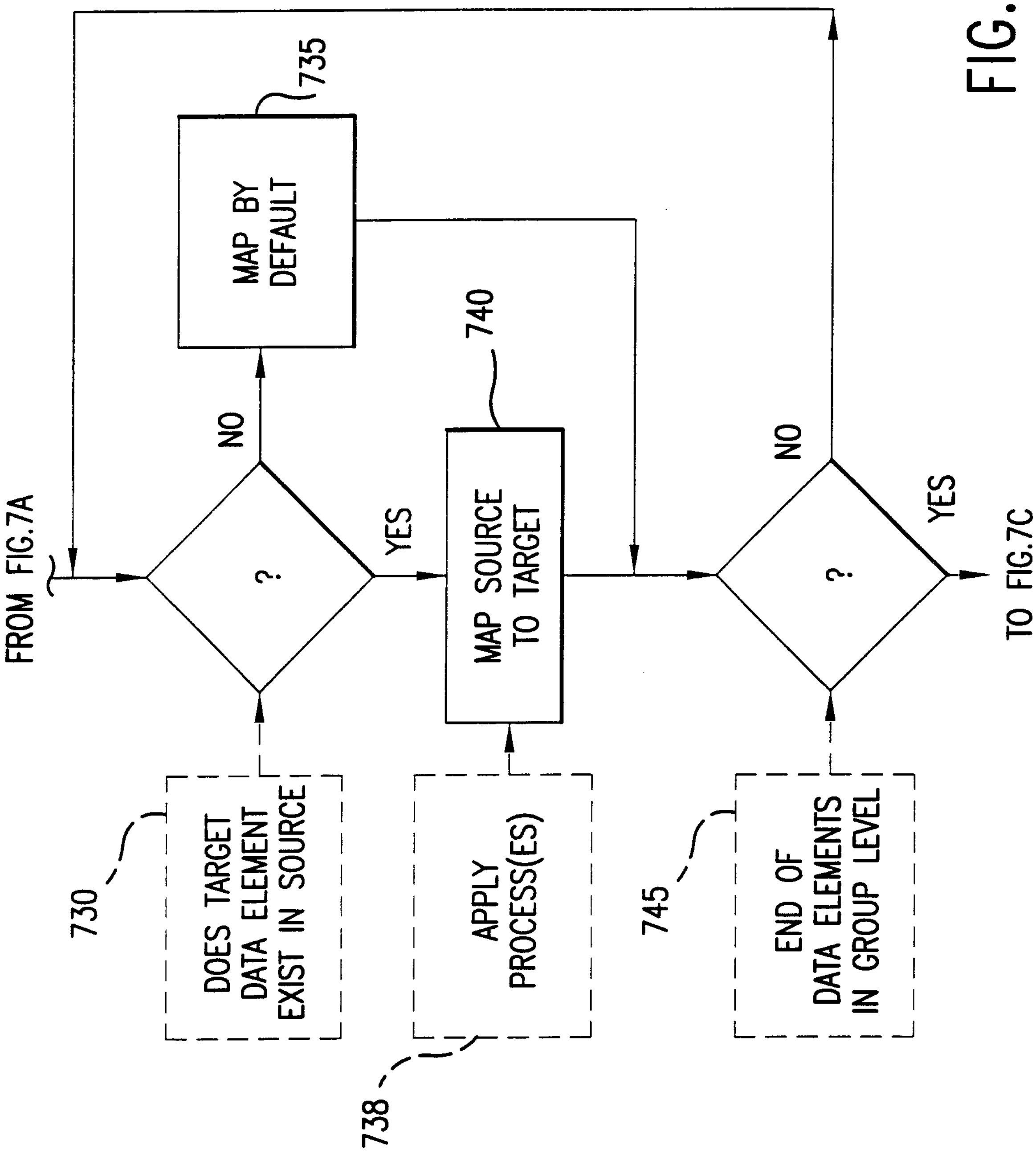

FROM FIG.7A
?
YES
NO
MAP BY DEFAULT
735
740
MAP SOURCE TO TARGET
?
NO
YES
TO FIG.7C
730
DOES TARGET DATA ELEMENT EXIST IN SOURCE
APPLY PROCESS(ES)
738
745
END OF DATA ELEMENTS IN GROUP LEVEL

FIG. 7B

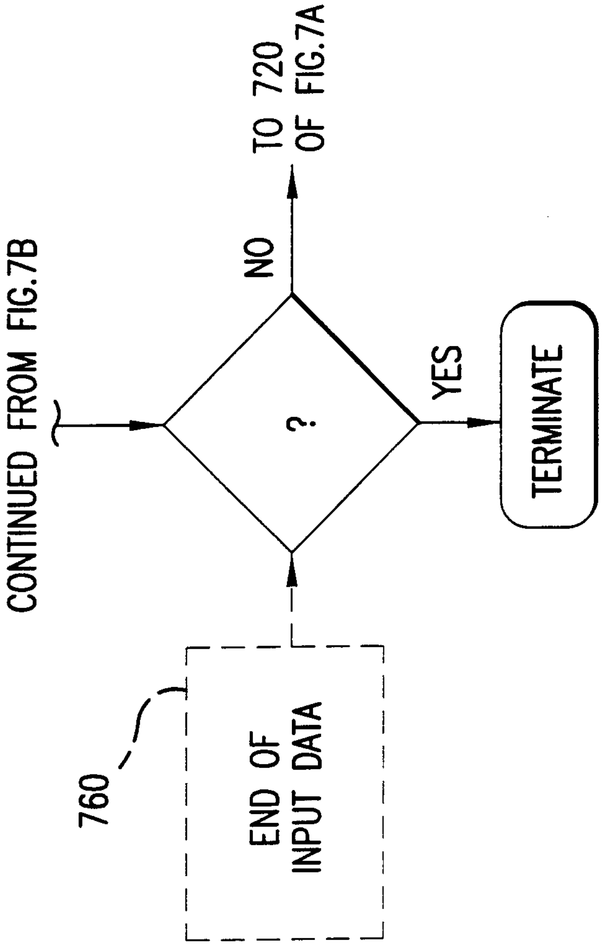

CONTINUED FROM FIG.7B
NO
TO 720
OF FIG.7A
?
YES
TERMINATE
760
END OF
INPUT DATA

FIG. 7C

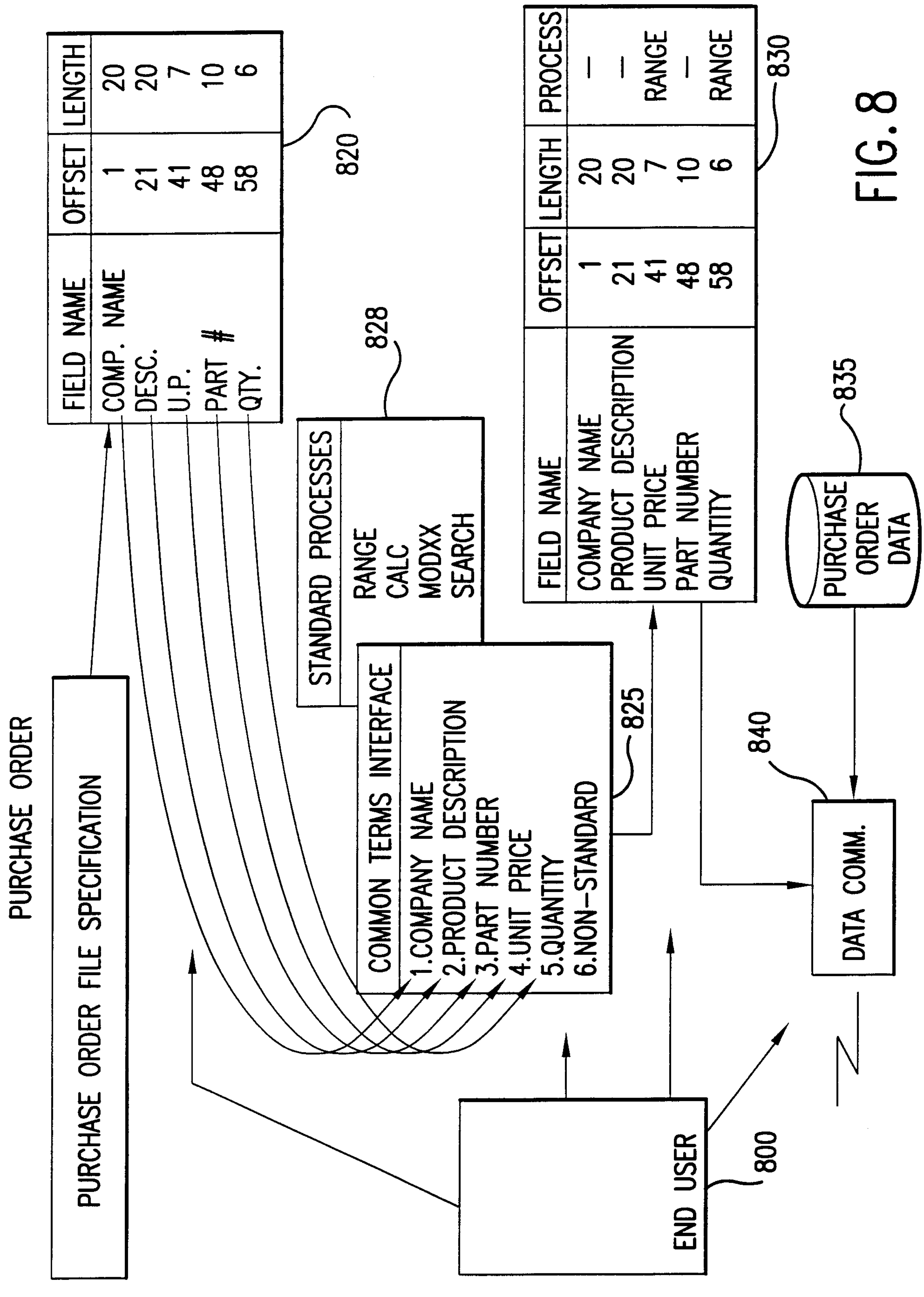

PURCHASE ORDER
PURCHASE ORDER FILE SPECIFICATION
FIELD NAME
COMP. NAME
DESC.
U.P.
PART #
QTY.
OFFSET
1
21
41
48
58
LENGTH
20
20
7
10
6
820
STANDARD PROCESSES
RANGE
CALC
MODXX
SEARCH
828
COMMON TERMS INTERFACE
1.COMPANY NAME
2.PRODUCT DESCRIPTION
3.PART NUMBER
4.UNIT PRICE
5.QUANTITY
6.NON-STANDARD
825
FIELD NAME
COMPANY NAME
PRODUCT DESCRIPTION
UNIT PRICE
PART NUMBER
QUANTITY
OFFSET
1
21
41
48
58
LENGTH
20
20
7
10
6
PROCESS
—
—
RANGE
—
RANGE
830
PURCHASE ORDER DATA
835
DATA COMM.
840
END USER
800

FIG. 8

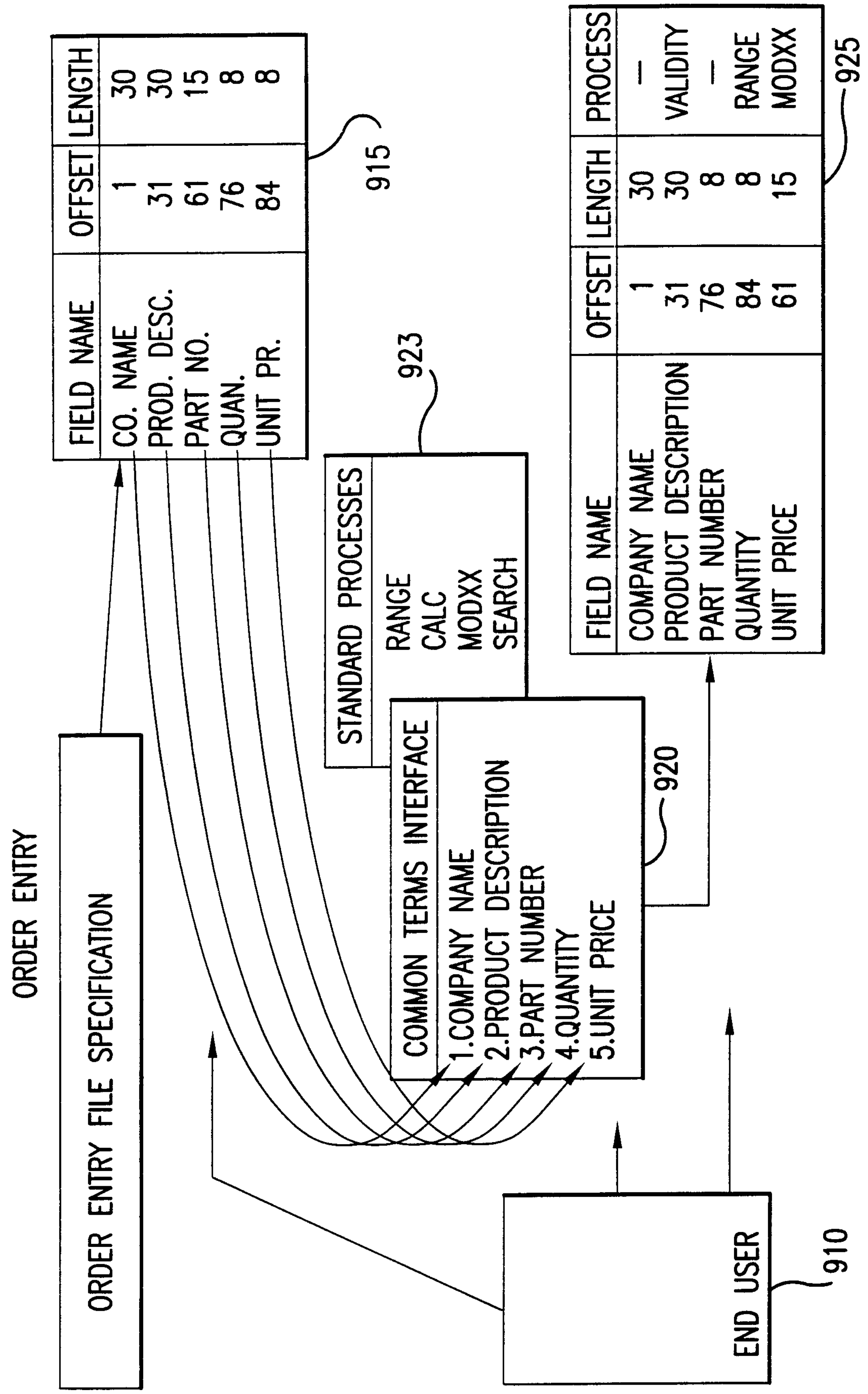

ORDER ENTRY
ORDER ENTRY FILE SPECIFICATION
FIELD NAME | OFFSET | LENGTH
CO. NAME | 1 | 30
PROD. DESC. | 31 | 30
PART NO. | 61 | 15
QUAN. | 76 | 8
UNIT PR. | 84 | 8
915
STANDARD PROCESSES
RANGE
CALC
MODXX
SEARCH
923
COMMON TERMS INTERFACE
1.COMPANY NAME
2.PRODUCT DESCRIPTION
3.PART NUMBER
4.QUANTITY
5.UNIT PRICE
920
FIELD NAME | OFFSET | LENGTH | PROCESS
COMPANY NAME | 1 | 30 | —
PRODUCT DESCRIPTION | 31 | 30 | VALIDITY
PART NUMBER | 76 | 8 | —
QUANTITY | 84 | 8 | RANGE
UNIT PRICE | 61 | 15 | MODXX
925
END USER
910

FIG. 9

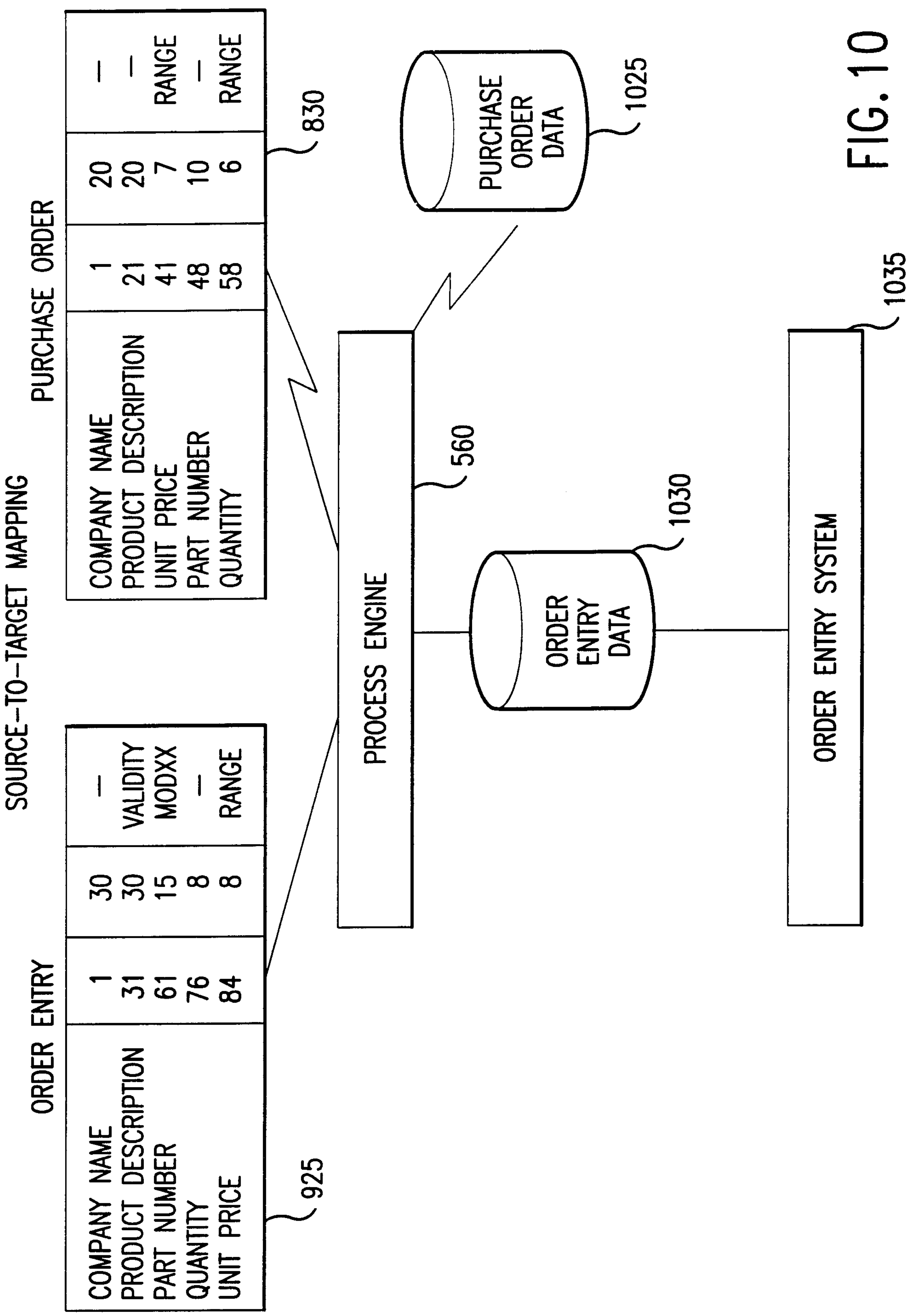

SOURCE-TO-TARGET MAPPING
ORDER ENTRY
COMPANY NAME 1 30 —
PRODUCT DESCRIPTION 31 30 VALIDITY
PART NUMBER 61 15 MODXX
QUANTITY 76 8 —
UNIT PRICE 84 8 RANGE
925
PURCHASE ORDER
COMPANY NAME 1 20 —
PRODUCT DESCRIPTION 21 20 —
UNIT PRICE 41 7 RANGE
PART NUMBER 48 10 —
QUANTITY 58 6 RANGE
830
PURCHASE ORDER DATA
1025
PROCESS ENGINE
560
ORDER ENTRY DATA
1030
ORDER ENTRY SYSTEM
1035

FIG. 10

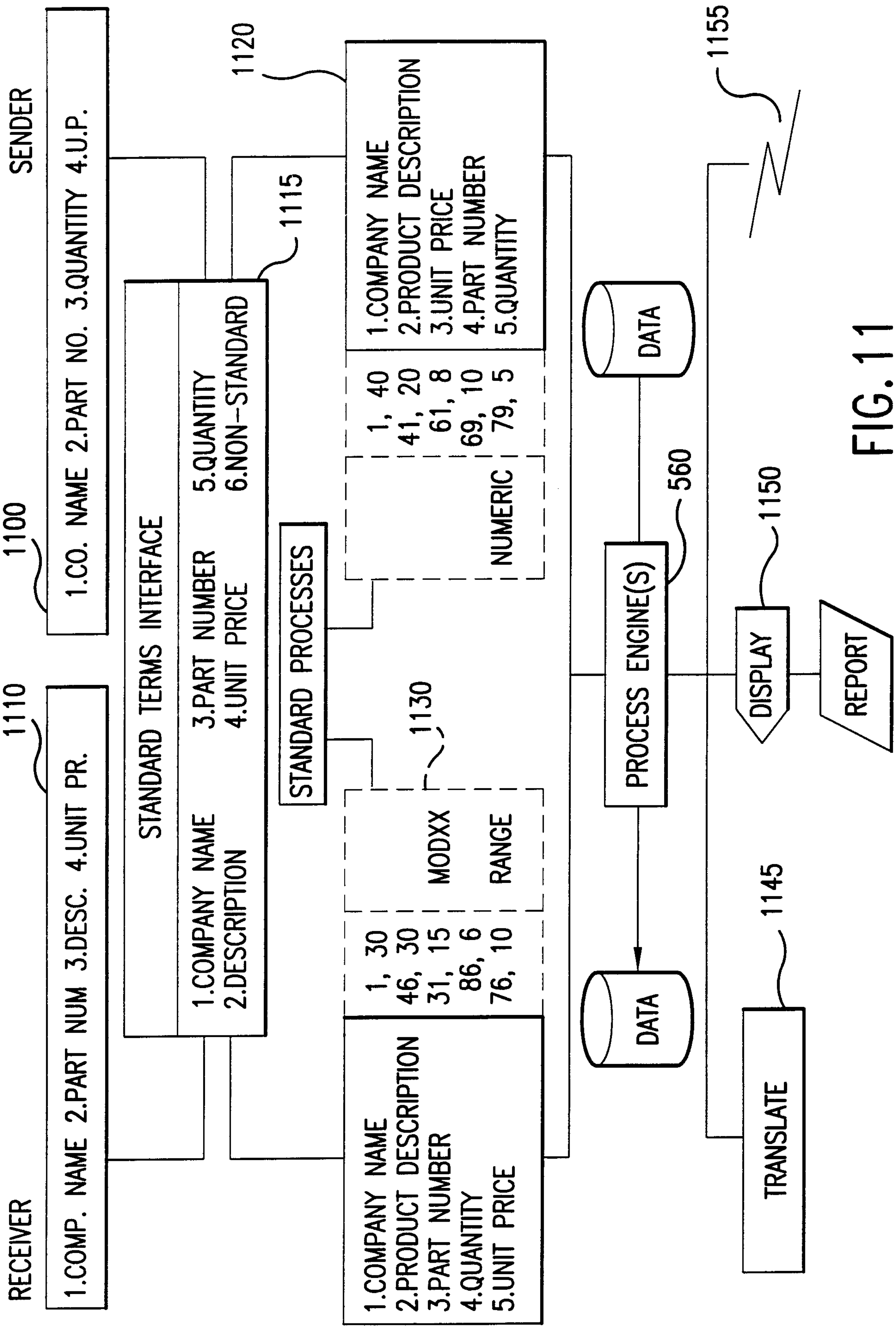

SENDER
1100
1.CO. NAME 2.PART NO. 3.QUANTITY 4.U.P.
RECEIVER
1110
1.COMP. NAME 2.PART NUM 3.DESC. 4.UNIT PR.
1115
STANDARD TERMS INTERFACE
1.COMPANY NAME
2.DESCRIPTION
3.PART NUMBER
4.UNIT PRICE
5.QUANTITY
6.NON-STANDARD
STANDARD PROCESSES
1120
1.COMPANY NAME
2.PRODUCT DESCRIPTION
3.UNIT PRICE
4.PART NUMBER
5.QUANTITY
1, 40
41, 20
61, 8
69, 10
79, 5
NUMERIC
1130
MODXX
RANGE
1, 30
46, 30
31, 15
86, 6
76, 10
1.COMPANY NAME
2.PRODUCT DESCRIPTION
3.PART NUMBER
4.QUANTITY
5.UNIT PRICE
DATA
560
PROCESS ENGINE(S)
DATA
1155
1150
DISPLAY
REPORT
1145
TRANSLATE

FIG. 11

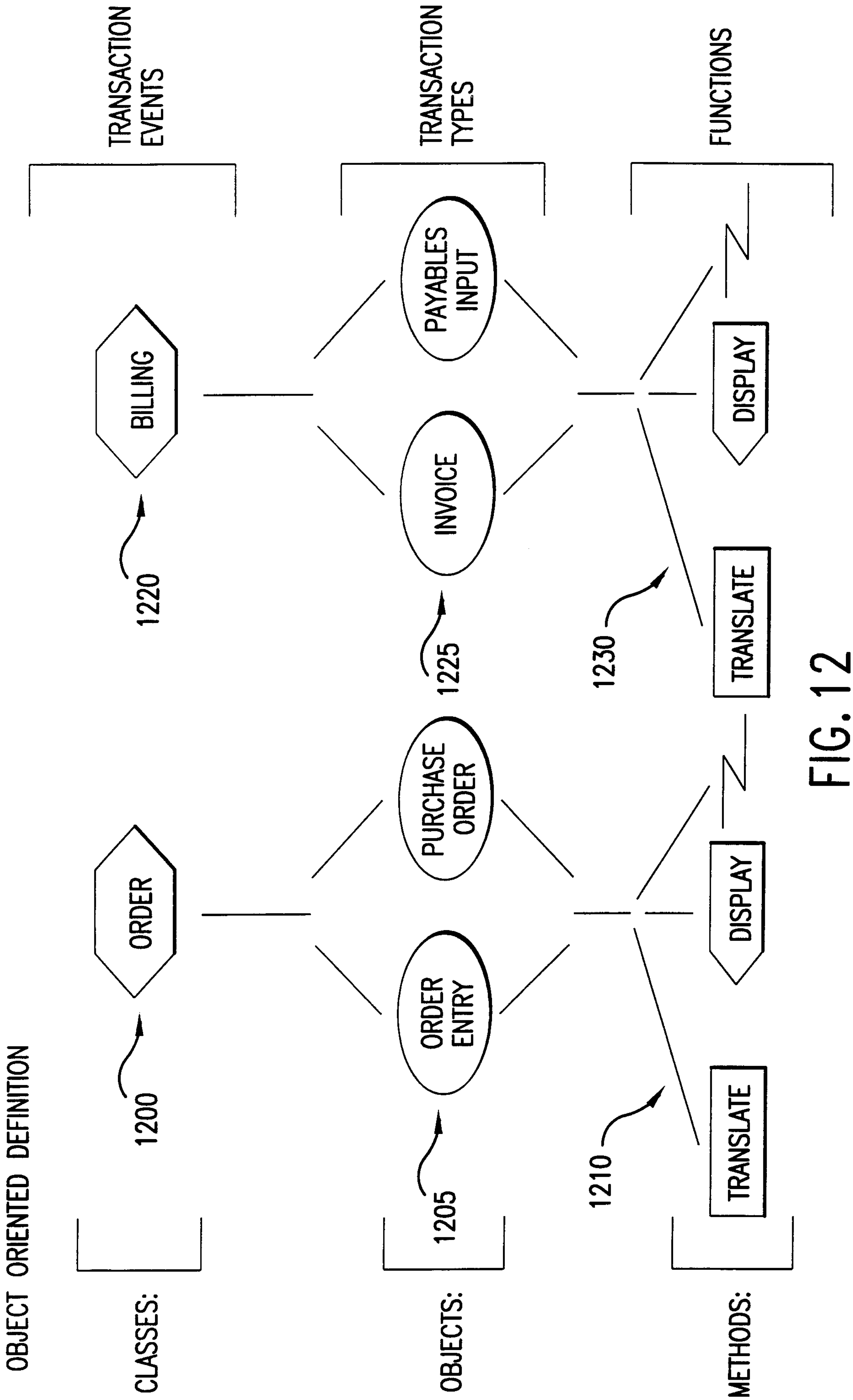

OBJECT ORIENTED DEFINITION
CLASSES:
OBJECTS:
METHODS:
TRANSACTION EVENTS
TRANSACTION TYPES
FUNCTIONS
ORDER
BILLING
ORDER ENTRY
PURCHASE ORDER
INVOICE
PAYABLES INPUT
TRANSLATE
DISPLAY
TRANSLATE
DISPLAY
1200
1205
1210
1220
1225
1230

FIG. 12