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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
17 November 2005 (17.11.2005)

PCT

(10) International Publication Number
WO 2005/109192 A2

(51) International Patent Classification[7]: G06F 9/44

(21) International Application Number:
PCT/US2005/014895

(22) International Filing Date: 29 April 2005 (29.04.2005)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
10/837,299 30 April 2004 (30.04.2004) US

(71) Applicant: **UNISYS CORPORATION** [US/US]; Unisys Way, MS/E8-114, Blue Bell, PA 19424-0001 (US).

(72) Inventors: **BAISLEY, Donald, Edward**; 24911 Hon Avenue, Laguna Hills, CA 92653 (US). **DIRCKZE, Ravi, Anthony, Joseph**; 146 Valley View Terrace, Mission Viejo, CA 92692 (US). **ZIEBELL, Jonathan, Virgil**; 21391 Eastglen Drive, Trabuco Canyon, CA 92679 (US). **COLE, Russel, Eliot**; 23962 Ibis Court, Laguna Niguel, CA 92677 (US).

(74) Agents: **STARR, Mark, T.** et al.; Unisys Corporation, Unisys Way, MS/E8-114, Blue Bell, PA 19424-0001 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, 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: GENERATING PROGRAMMATIC INTERFACES FROM NATURAL LANGUAGE EXPRESSIONS OF AUTHORIZATIONS FOR REQUEST OF INFORMATION

(57) Abstract: An embodiment of the present invention is a technique for processing an authorization rule. An object type is created for a return type of an operation in the rule authorizing a request for information. A current container is established. A propositional expression in the rule is processed to record the object type as a resulting context from the propositional expression in the current container.

WO 2005/109192 A2

GENERATING PROGRAMMATIC INTERFACES FROM NATURAL LANGUAGE EXPRESSIONS OF AUTHORIZATIONS FOR REQUEST OF INFORMATION

BACKGROUND

FIELD OF THE INVENTION

Embodiments of the invention are in the field of natural language processing, and relate more specifically to programmatic interfaces from natural language expressions.

DESCRIPTION OF RELATED ART

Natural language used by human to communicate tends to be contextual and imprecise. To automate natural language processing using computerized methods, certain rules are usually imposed to confine the natural language expressions to a well-defined format. There are several applications that can provide an environment where natural language expressions may be expressed in an unambiguous format. One such application is business rules.

When business services or transactions are automated in a computer system, business rules provide business requirements on the automated system. These requirements dictate what the system should and should not do, or who can provide information to the system and who can request information from the system. However, the process of translating business rules expressed in business language into software components using human software developers tends to be error-prone and time-consuming. Automatic generation of software systems from business rules will save time and avoid the problem of errors that would occur in manual translation.

SUMMARY OF THE INVENTION

An embodiment of the present invention is a technique to process an authorization rule. An object type is created for a return type of an operation in the rule authorizing a request for information. A current container is established. A propositional expression in the rule is processed to record the object type as a resulting context from the propositional expression in the current container.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention may best be understood by referring to the following description and accompanying drawings that are used to illustrate embodiments of the invention. In the drawings:

Figure 1 is a diagram illustrating a system in which one embodiment of the invention can be practiced.

Figure 2 is a flowchart illustrating a process to translate business rules into application programmatic interface (API) according to one embodiment of the invention.

Figure 3 is a flowchart illustrating a process to process a propositional expression according to one embodiment of the invention.

Figure 4 is a flowchart illustrating a process to establish a context for the propositional expression according to one embodiment of the invention.

Figure 5 is a flowchart illustrating a process to parse the propositional expression if the propositional interrogative is applied according to one embodiment of the invention.

Figure 6 is a flowchart illustrating a process to parse the propositional expression if the propositional expression includes an interrogative operator according to one embodiment of the invention.

Figure 7 is a flowchart illustrating a process to generate second object type for the term according to one embodiment of the invention.

Figure 8 is a diagram illustrating a computer system in which one embodiment of the invention can be practiced.

DESCRIPTION

An embodiment of the present invention is a technique to process an authorization rule. An object type is created for a return type of an operation in the rule authorizing a request for information. A current container is established. A propositional expression in the rule is processed to record the object type as a resulting context from the propositional expression in the current container.

In the following description, numerous specific details are set forth. However, it is understood that embodiments of the invention may be practiced without these specific details. In other instances, well-known circuits, structures, and techniques have not been shown in order not to obscure the understanding of this description.

An embodiment of the present invention considers the part of an authorization rule that expresses what information may be requested, apart from any consideration of who requests it or conditions under which it may be requested. The method of the present invention examines propositional expressions of the authorization rule with respect to use of propositional interrogative operators, interrogative operators and parametric operators.

Many relevant linguistic concepts are used in the following description. These concepts are developed using a linguistic terminology that includes a number of terms. These terms include "expression", "nominal expression", "term", "name", "numerical literal", "textual literal", "role expression", "sentence", "simple sentence", "complex sentence", "functional form", "sentential form", "parametric operator", "interrogative operator", "propositional interrogative", "identification scheme", "type", "category", "role", "supertype", and "subtype".

An expression is a symbol or combination of symbols that means something. The meaning can be anything, including a proposition, a rule, a number, etc.

A nominal expression is a category of expression that names a thing or things.

A term is a category of symbol, a symbol that denotes being of a type, i.e., a common noun. Examples: "car" denoting a category of vehicle, "bank account".

A name is a category of symbol and of nominal expression; a symbol that names an individual thing, i.e., a proper noun. Examples: "California" naming a state of the United States, "Unisys" naming the company Unisys.

A numerical literal is a category of name that denotes a number using numerals. For example, "123" meaning the number 123.

A textual literal is a category of symbol and of nominal expression; a symbol that represents words, punctuation, textual characters or a sequence of any of these by literal presentation, as in quotation marks. For example, "hello" representing the word "hello".

A role expression is a category of nominal expression. A nominal expression consists primarily of a term given in place of a placeholder in an expression based on a functional form, and consists secondarily of each operator (e.g., quantifier, pronominal operator, parametric operator, interrogative operator) and object modifier applied to the term together with any expression of instances specifically referenced by the term, or, if the denoted type's range is restricted using a nominal restrictive form, that nominal restrictive form along with the expression of each argument to the function delineated by that form. Examples: "a checking account" in the expression "a checking account has the overdraw limit ($1000.00)"; "the overdraw limit ($1000.00)" in the expression "a checking account has the overdraw limit ($1000.00)".

A sentence is a category of expression; an expression that denotes a proposition (possibly an open or interrogative proposition).

A simple sentence is a category of sentence; a sentence that is stated using a single sentential form – no logical connectives. It includes a nominal expression for each placeholder of the sentential form. Example: "Each person has a name".

A complex sentence is a category of sentence. It is a sentence that combines other sentences using a logical connective such as if, and, or, etc. Example: "Each American citizen has a name and a social security number".

A functional form is a category of symbol and of expression; a complex symbol that is a sequence of typed placeholders and words interspersed that delineates a function and serves as a form for invoking the function in expressions. Each typed placeholder

appears in the sequence as a term denoting the placeholder's type specially marked in some way (such as by underlining).

A sentential form is a category of functional form that delineates a propositional function. Example: "vendor charges price for product".

A parametric operator is a category of operator that when expressed with a term denotes a discourse referent determined by future discourse context, with singular quantification. Example: "a given" in "Each medical receptionist is authorized to provide what doctor sees a given patient".

An interrogative operator is a category of operator that when expressed with a term in a role expression denotes a discourse referent determined by future discourse context. The role expression is thereby a name for satisfiers in the encompassing sentence. Examples: the operator "what" in "What doctor sees what patient", the operators "which" and "what" in "Which doctor sees what patient". Note that "what" carries the meaning of "who", "when", "how", "where", "why", etc. when used as an operator on a term, e.g., in "what person", "what time" or "what date", "what method", "what location", "what purpose", etc.

A propositional interrogative is a category of operator that when expressed with a proposition denotes the truth-value of the proposition with regard to future discourse context. Example: the operator "whether" in "whether each doctor is licensed".

An identification scheme is a scheme by which a thing of some type can be identified by facts about the thing that relate the thing to signifiers or to other things identified by signifiers. The identifying scheme comprises of the set of terms that correspond to the signifiers. Example: an employee may be identified by employee number.

A type is a classification of things (often by category or by role). A category is a role of a type in a categorization relation to a more general type. The category classifies a subset of the instances of the more general type based on some delimiting characteristic. Example: checking account is a category of account.

A role is a role of a type whose essential characteristic is that its instances play some part, or are put to some use, in some situation. The type classifies an instance based, not on a distinguishing characteristic of the instance itself (as with a category), but on some fact that involves the instance. Example: destination city is a role of a city.

A supertype is a role of a type used in relation to another type such that the other type is a category or role of the supertype, directly or indirectly. Each instance of the other type is an instance of the supertype. Examples: animal is a supertype of person (assuming person is a category of animal) and person is a supertype of driver (assuming driver is a role of a person).

A subtype is a role of a type used in relation to another type such that the subtype is a category or role of the other type, directly or indirectly. Each instance of the subtype is an instance of the other type. This is the inverse of supertype. Examples: person is a subtype of animal (assuming person is a category of animal) and driver is a subtype of person (assuming driver is a role of a person).

In one embodiment, the invention is implemented using object-oriented technique. The object-oriented technique is a method to represent a system using objects and associations between objects. The technique involves the use of "class", "association", "attribute", and "operation". Although these terms are commonly known, they are defined in the following for clarification.

A class is an abstract concept representing a real world thing of interest to the system, such as a person, a router in a network, etc. A class is a template that defines the behavior and attributes that particular type of object possesses. A class can be the base for other classes. The behavior of the object is the collective set of operations that the object can perform, which are defined in respective class. The state of the object is defined by the values of its attributes at any given time.

An association represents a relationship between objects.

An attribute represents some aspect of an object. For example, the color of an automobile, the date of birth of a person. Each attribute has a type that defines the range of values that the attribute can have.

An operation represents a behavior that an object possesses.

A business rule that authorizes requests for information by a business actor indicates what information can be provided to the business actor. Such indications build on three linguistic concepts: (1) propositional interrogatives, (2) interrogative operators, and (3) parametric operators. These concepts are easily exemplified in English or other languages. The following are some examples in English.

Propositional interrogative: "A bank manager may request whether at least one account is suspended." In this example, the word "whether" is a propositional interrogative for the proposition "an account is suspended". This rule allows a bank manager to request the answer to the yes/no question "whether at least one account is suspended".

Interrogative operator: "A bank manager may request what account is suspended." In this example, the word "what" is an interrogative operator acting on the term "account" in the scope of the proposition "account is suspended". This rule allows a bank manager to request a list of accounts which are suspended.

Parametric operator: "A bank manager may request whether a given account is suspended.", "A bank manager may request which loan officer is assigned to a given loan." A parametric operator is used to indicate where a requestor indicates what thing is being asked about. In the two examples above, the word "given" is a parametric operator acting on the terms "account" and "loan", respectively. The first rule allows a bank manager to request the answer to a yes/no question about any particular account that the bank manager specifies – whether that account is suspended. The second allows a bank manager ask a question about a loan that the bank manager specifies – what loan officer is assigned to it.

The third concept can be used in combination with either of the first two, but the first two concepts are distinct in usage and are not mixed except when using conjunction (explained below). For example, a propositional interrogative (like "whether") is not used in an authorization rule on a proposition containing an interrogative operator (like "what"). However, multiple interrogative operators can occur in a single proposition. For example: "A bank manager may request which loan officer is assigned to what loan." In the above example, "which" and "what" are both interrogative operators indicating that a bank manager can request a list of assignments of loan officers to loans.

Similarly, parametric operators can be used multiple times in a single proposition: "A bank manager may request whether a given loan officer is assigned to a given loan." In this example, "a given" is a parametric operator indicating that both a loan officer and a loan are identified by a bank manager when requesting information.

It is possible for a rule to be conditioned on other information. For example, the rule above could be conditioned: "A bank manager may request whether a given loan

officer is assigned to a given loan if the loan is with a bank managed by the bank manager."

The condition does not affect what information can be requested other than to limit the circumstances in which the request is allowed. Such conditions do not change affect the interface of a software component that dispenses the information, but only the behavior of the component — whether or not it will accept a request for information. For this reason, conditions placed on authorization rules can be ignored in the method of the present invention.

Note that the three linguistic concepts above are represented by many different words in English (e.g. "which" for "what") and by many different words in other languages. The present invention does not consider specific words, but the three specific linguistic concepts categorized above.

A single authorization rule can use conjunction in order to authorize multiple kinds of information. Example: "A bank manager may request what credit limit is on a given account and whether the account is suspended." The rule above uses the word "and" to indicate conjunction, which causes authorization to be granted, in this case, for two kinds of facts. Note that conjunction can occur using other words or forms. It is a general linguistic concept found in all languages.

One embodiment of the invention translates request interrogatives to object-oriented (OO) application programming interfaces (APIs). Each business rule that authorizes a request for information results in a single operation in an OO API. Each operation takes one or more parameters as arguments and returns a single object as a result. The type of the returned object, the number of parameters and the type of each parameter is determined in the operations described below.

Figure 1 is a diagram illustrating a system 100 in which one embodiment of the invention can be practiced. The system 100 includes a business rule generator 110, a translator 120, and a business processing system 130. The system 100 may be implemented by software or hardware, or a combination of hardware and software.

The business rule generator 110 generates business rules to be used in the business processing system 130. These business rules are in a linguistic form having a

predefined syntax or format. In one embodiment, these business rules may represent authorizations to request or to provide information.

The translator 120 translates the business rules generated from the business rule generator 110 into application programmatic interfaces (APIs) that can be incorporated into the automated processing system 130. The APIs represent software components that can be readily integrated into the framework of the business processing system 130.

The business processing system 130 processes the business rules encoded as the APIs provided by the translator 120. The business processing system 130 may represent any system that process commercial or business transactions such as product ordering, vacation planning, etc.

Elements of one embodiment of the invention may be implemented by hardware, firmware, software or any combination thereof. When implemented in software or firmware, the elements of an embodiment of the present invention are essentially the code segments to perform the necessary tasks. The software/firmware may include the actual code to carry out the operations described in one embodiment of the invention, or code that emulates or simulates the operations. The program or code segments can be stored in a processor or machine accessible medium or transmitted by a computer data signal embodied in a carrier wave, or a signal modulated by a carrier, over a transmission medium. The "processor readable or accessible medium" or "machine readable or accessible medium" may include any medium that can store, transmit, or transfer information. Examples of the processor readable or machine accessible medium include an electronic circuit, a semiconductor memory device, a read only memory (ROM), a flash memory, an erasable ROM (EROM), a floppy diskette, a compact disk (CD) ROM, an optical disk, a hard disk, a fiber optic medium, a radio frequency (RF) link, etc. The computer data signal may include any signal that can propagate over a transmission medium such as electronic network channels, optical fibers, air, electromagnetic, RF links, etc. The code segments may be downloaded via computer networks such as the Internet, Intranet, etc. The machine accessible medium may be embodied in an article of manufacture. The machine accessible medium may include data that, when accessed by a machine, cause the machine to perform the operations described above. The machine accessible medium may also include program code embedded therein. The program code may include machine readable

code to perform the operations described above. The term "data" here refers to any type of information that is encoded for machine-readable purposes. Therefore, it may include program, code, data, file, etc.

One embodiment of the invention may be described as a process which is usually depicted as a flowchart, a flow diagram, a structure diagram, or a block diagram. Although a flowchart may describe the operations as a sequential process, many of the operations can be performed in parallel or concurrently. A loop or iterations in a flowchart may be described by a single iteration. It is understood that a loop index or loop indices or counter or counters are maintained to update the associated counters or pointers. In addition, the order of the operations may be re-arranged. A process is terminated when its operations are completed. A process may correspond to a method, a program, a procedure, etc.

An embodiment of the invention performs the translation of a business rule, e.g., an authorization rule, into object-oriented application programmatic interfaces (OO APIs). The technique may be described by the following pseudo code.

Block-1: Process the authorization rule

1. Create an object type for the return type of the operation
2. Create an operation for the authorization whose return type is the object type created in step-1
3. Establish a variable, current-container
4. For each propositional expression in the rule describing requested information
   - 4.1 Set the current-container to be the object type for the return type
   - 4.2 If a propositional interrogative is applied to the propositional expression then
     - 4.2.1 process the expression as stated in block-2
   - 4.3 otherwise, if the propositional expression includes an interrogative operator
     - 4.3.1 process the expression as stated in block-3
   - 4.4 record that the object type in the current-container is the resulting context from the propositional expression

Block-2: Parse a propositional expression to which a propositional interrogative is applied.

1. Establish a context for the propositional expression following the steps in block-5
2. Process each role expression in the propositional expression

2.1 For each role expression that is not a pronominal reference and that includes no definite article or expression of specifically referenced instances, reading in logical order

2.1.1 If the role expression has a parametric operator,

2.1.1.1 Generate an object type for the term in the role expression as stated in block-4

2.1.1.2 Add a parameter whose type is the object type generated for the term (in step 2.1.1.1 above) to the operation being created.

3. Process the truthfulness of the propositional interrogative

3.1 Add a Boolean attribute to the object type designated by the current-container to indicate whether the proposition is true or false

Block-3: Parse a propositional expression containing a term to which an interrogative operator is applied.

1. Establish a context for the propositional expression following the steps in block-5
2. Process each role expression in the propositional expression

2.1 For each role expression that is not a pronominal reference, reading in the logical order

2.1.1 If the role expression has a parametric operator,

2.1.1.1 Generate an object type for the term in the role expression as stated in block-4

2.1.1.2 Add a parameter whose type is the object type generated for the term (in step 2.1.1.1 above) to the operation being created.

2.1.2 otherwise, if the role expression contains an interrogative operator

2.1.2.1 Generate an object type for the term in the role expression as stated in block-4

2.1.2.2 Add an attribute to the object type designated by the current-container whose attribute type is a collection of objects of the object type generated for the term (in step 2.1.2.1).

2.1.2.3 Set the current-container to the object-type generated for the term (in step 2.1.2.1)

Block-4: Generating an object type for a term

1. Create an object type
2. For each term in the identification scheme of the term

2.1 If the term has a numerical or lexical type

2.1.1 Add an attribute of the numerical or lexical type to the generated object type

2.2 Otherwise,

2.2.1 Generate an object type for the term as stated in block-4

2.2.2 Add an attribute whose attribute type is the object type generated in the previous step (step 2.2.1) to the object type created in step 1

Block-5: Establish a context for a propositional expression

1. Find each pronominal reference in the propositional expression whose referent is a term in a previously processed propositional expression, but exclude each referent whose referent is the subject of a parametric operator.
2. If a pronominal reference is found (in step 1 above), then

2.1 Select the last processed propositional expression having a referent term from step 1.

2.2 If any referent term in that last processed propositional expression is not the subject of an interrogative operator

2.2.1 Set the current-container to be the object type that is the resulting context from that propositional expression

2.3 Otherwise, the last referent term found in step 1 is the subject of an interrogative operator

2.3.1 Set the current-container to be the object type generated for that last referent term.

Figure 2 is a flowchart illustrating a process 200 to translate business rules into application programmatic interface (API) according to one embodiment of the invention. The process 200 implements the translator 120 shown in Figure 1.

Upon START, the process 200 creates an object type for the return type of an operation in a rule authorizing a request for information (Block 210). Next, the process 200 establishes a variable called a current container (Block 220). Then, the process 200 processes a proposition expression in the rule to record that the object type in the current container is the resulting context from the proposition expression (Block 230). The process 200 is then terminated.

Figure 3 is a flowchart illustrating a process 230 to process a propositional expression according to one embodiment of the invention.

Upon START, the process 230 determines if the rule contain an unprocessed propositional expression describing the requested information (Block 310). If not, the process 230 is terminated. Otherwise, the process 230 proceeds to set the current container to be the object type for the return type (Block 320).

Next the process 230 establishes a context for the propositional expression (Block 330). Then, the process 230 parses the propositional expression (Block 340 including two parts 340A and 340B) and is then terminated.

Figure 4 is a flowchart illustrating a process 330 to establish a context for the propositional expression according to one embodiment of the invention.

Upon START, the process 330 finds each pronominal reference in the propositional expression (PE) having a referent term to be a term in a previously processed propositional expression, and not to be a subject of a parametric operator (Block 410). Next, the process 330 determines if such a pronominal reference was found (Block 420). If not, the process 330 is terminated. Otherwise, the process 330 selects the last processed propositional expression having such a referent term (Block 430). Then, the process 330 determines if any referent term in that last processed PE is not the subject of an interrogative operator (Block 440). If so, the process 330 sets the current container to be the object type that is the resulting context from the PE (Block 450) and is then terminated. Otherwise, the process 330 sets the current container to be the object type generated from the last referent term (Block 460) and is then terminated.

Block 340 of the process 230 (Figure 3) for parsing the propositional expression comprises two processes, 340A and 340B.

Figure 5 is a flowchart illustrating a process 340A to parse the propositional expression if the propositional interrogative is applied according to one embodiment of the invention.

Upon START, the process 340A determines if a propositional interrogative is applied to the propositional expression (Block 510). If not, the process 340A is terminated. Otherwise, the process 340A reads in logical order a role expression in the PE (Block 520). Next, the process 340A determines if the unprocessed role expression is not a pronominal reference and including no definite article or expression of a specifically referenced instance (Block 530). If not, the process 340A proceeds to Block 570. Otherwise, the process 340A determines if the role expression have a parametric operator (Block 540). If not, the process 340A proceeds to Block 570. Otherwise, the process 340A generates a second object type for the term in the role expression (Block 550).

Next, the process 340A adds a parameter having the second object type to the operation being created (Block 560). Then, the process 340A adds a Boolean attribute to the object type designated by the current container to indicate a truth value (i.e., true or false) of the propositional expression (Block 570) and is then terminated.

Figure 6 is a flowchart illustrating a process 340B to parse the propositional expression if the propositional expression includes an interrogative operator according to one embodiment of the invention.

Upon START, the process 340B determines if the PE includes an interrogative operator (Block 610). If not, the process 340B is terminated. Otherwise, the process 340B reads in logical order a role expression in the PE (Block 620). Next, the process 340B determines if the unprocessed role expression is not a pronominal reference (Block 630). If not, the process 340B is terminated. Otherwise, the process 340B generates a second object type for the term in the role expression (Block 640).

Then, the process 340B determines if the role expression contain a parametric operator (Block 650). If so, the process 340B adds a parameter having the second object type to the operation (Block 660) and is then terminated. Otherwise, the process 340B determines if the role expression contain an interrogative operator (Block 670). If not, the process 340B is terminated. Otherwise, the process 340B adds an attribute to the object type designated by the current container having an attribute type to be a collection of objects of the second object type (Block 680). Next, the process 340B sets the current container to the second object type (Block 690) and is then terminated.

Figure 7 is a flowchart illustrating a process 550/640 to generate second object type for the term according to one embodiment of the invention.

Upon START, the process 550/640 creates a second object type (Block 710). Next, the process 550/640 determines if the identification scheme of the term contain an unprocessed term (Block 720). If not, the process 550/640 is terminated. Otherwise, the process 550/640 determines if the term has a numerical or lexical type (Block 730). If so, the process 550/640 adds an attribute of the numerical or lexical type to the second object type (Block 740) and returns to Block 720. Otherwise, the process 550/640 generates a temporary object type for the term (Block 750).

Next, the process 550/640 adds an attribute having an attribute type same as the temporary object type to the second object type (Block 760) and returns to Block 720.

The technique described above may be illustrated further by some examples. The examples in this section are based on the following vocabulary.

Vocabulary
agent
name
name *is a role of* text

customer
customer has name
identification criteria: customer is identified by name
meal preference
meal preference *is a role of* Text
customer has meal preference

departure date
departure date *is a role of* date

cruise
cruise has name
identification criteria: cruise is identified by name

customer has a reservation for cruise on departure date

location
city name
city name *is a role of* text
country name
country name *is a role of* text
location has city name
location has country name
identification criteria: location is identified by city name, country name

duration
duration *is a role of* integer

visit
visit is to location
visit is for duration
identification criteria: visit is identified by location, duration

cruise makes visit

Authorization rule 1: *Each* agent is authorized to request *whether a given* customer has a reservation for *a given* cruise on *a given* departure date.

Authorization rule 2: *Each* agent is authorized to request *what* customer has a reservation for *what* cruise on *a given* departure date *and whether the* customer has *a given* meal preference.

Authorization rule 3: *Each* agent is authorized to request *whether a given* customer has a reservation for *a given* cruise on *a given* departure date *and the* customer has *what* meal preference.

Authorization rule 4: *Each* agent is authorized to request *what* customer has a reservation for *a given* cruise on *a given* departure date *and the* cruise makes *what* visit.

Interface for Authorization rule 1: *Each* agent is authorized to request *whether a given* customer has a reservation for *a given* cruise on *a given* departure date. The execution of the algorithm will result in the following OO interface:

```
Return-Object-Type GeneratedOperation(Customer-Type : customer, Cruise-Type
cruise, Departure-Date-Type departure-date)
Class: Return-Object-Type {
        Boolean IsPropositionTrue
}

Class: Customer-Type {
        Text : name
}

Class Cruise-Type {
        Text: name
}

Class Departure-Date-Type {
        Date: date
}
```

Interface for Authorization rule 2: *Each* agent is authorized to request *what* customer has a reservation for *what* cruise on *a given* departure date *and whether the* customer has *a given* meal preference. The execution of the algorithm will result in the following OO interface:

```
Return-Object-Type GeneratedOperation(Departure-Date-Type departure-date, Meal-
Preference-Type meal-preference)
Class: Return-Object-Type {
        Customer-Type-Collection: customer-type-collection
```

Note: Customer-Type-Collection is a collection of Customer-Type objects.

}

Class: Customer-Type {
Text : name
Cruise-Type-Collection : cruise-type-collection
Note: Cruise-Type-Collection is a collection of Cruise-Type objects.
Boolean : has-meal-preference
}

Class Cruise-Type {
Text: name
}

Class Meal-Preference-Type {
Text: meal-type
}

Interface for Authorization rule 3: *Each* agent is authorized to request *whether a given* customer has a reservation for *a given* cruise on *a given* departure date *and the* customer has *what* meal preference. The execution of the algorithm will result in the following OO interface:

Return-Object-Type GeneratedOperation(Customer-Type: customer, Cruise-Type cruise, Departure-Date-Type departure-date)

Class: Return-Object-Type {
Boolean IsPropositionTrue
Meal-Preference-Type-Collection: meal-preference—type-collection
Note: Meal-Preference-Type-Collection is a collection of Meal-Preference-Type objects.
}

Class: Customer-Type {
Text : name
}

Class Cruise-Type {
Text: name
}

Class Departure-Date-Type {
Date: date
}

Class Meal-Preference-Type {
Text: meal-type

}

Interface for Authorization rule 3: *Each* agent is authorized to request *what* customer has a reservation for *a given* cruise on *a given* departure date *and the* cruise makes *what* visit

Return-Object-Type GeneratedOperation(Cruise-Type cruise, Departure-Date-Type departure-date)

```
Class: Return-Object-Type {
        Customer-Type-Collection: customer-type-collection
    Note: Customer-Type-Collection is a collection of Customer-Type objects.
        Visit-Type-Collection: visit-type-collection
    Note: Visit-Type-Collection is a collection of Visit-Type objects.
}

Class Cruise-Type {
        Text: name
}

Class Departure-Date-Type {
        Date: date
}

Class: Customer-Type {
        Text : name
}

Class Visit-Type {
        Location-Type: location
        Integer: duration
}

Class Location-Type {
        Text : city-name
        Text : country-name
}
```

Figure 8 is a diagram illustrating a computer system 800 in which one embodiment of the invention can be practiced.

The computer system 800 includes a processor 812, a memory 814, and a mass storage device 816. The computer system 800 receives a stream of input representing a set of business rules or vocabulary, processes the business rules or the vocabulary in accordance to the method of the present invention, and outputs an object-oriented API.

The processor 812 represents a central processing unit of any type of architecture, such as embedded processors, mobile processors, micro-controllers, digital signal processors, superscalar computers, vector processors, single instruction multiple data (SIMD) computers, complex instruction set computers (CISC), reduced instruction set computers (RISC), very long instruction word (VLIW), or hybrid architecture.

The memory 814 stores system code and data. The memory 814 is typically implemented with dynamic random access memory (DRAM) or static random access memory (SRAM). The system memory may include program code or code segments implementing one embodiment of the invention. The memory 814 includes a translator module 815 of the present invention when loaded from mass storage 816. The translator module 815 implements all or part of the translator 120 shown in Figure 1. The translator module 815 may also simulate the translation functions described herein. The translator module 815 contains instructions that, when executed by the processor 812, cause the processor to perform the tasks or operations as described above.

The mass storage device 816 stores archive information such as code, programs, files, data, databases, applications, and operating systems. The mass storage device 816 may include compact disk (CD) ROM, a digital video/versatile disc (DVD), floppy drive, and hard drive, and any other magnetic or optic storage devices such as tape drive, tape library, redundant arrays of inexpensive disks (RAIDs), etc. The mass storage device 816 provides a mechanism to read machine-accessible media. The machine-accessible media may contain computer readable program code to perform tasks as described above.

While the invention has been described in terms of several embodiments, those of ordinary skill in the art will recognize that the invention is not limited to the embodiments described, but can be practiced with modification and alteration within the spirit and scope of the appended claims. The description is thus to be regarded as illustrative instead of limiting.

CLAIMS

What is claimed is:

1. A method comprising:

creating an object type for a return type of an operation in a rule authorizing a request for information;

establishing a current container; and

processing a propositional expression in the rule to record the object type to be a resulting context from the propositional expression in the current container.

2. The method of claim 1 wherein processing the propositional expression comprises:

setting the current container to be the object type;

establishing a context for the propositional expression; and

parsing the propositional expression.

3. The method of claim 2 wherein establishing the context for the propositional expression comprises:

finding a pronominal reference in the propositional expression having a referent term to be a term in a previously processed propositional expression and not to be a subject of a parametric operator; and

if the referent term is not subject of an interrogative operator,

setting the current container to be the object type of the context resulting from the propositional expression;

else setting the current container to be the object type generated for the referent term.

4. The method of claim 2 wherein parsing the propositional expression comprises:

reading in logical order a role expression in the propositional expression if a propositional interrogative is applied to the propositional expression, the role expression being not a pronominal reference and including no definite article or expression of a specifically referenced instance;

adding a Boolean attribute to the object type designated by the current container to indicate a truth value of the propositional expression; and

if the role expression has a parametric operator,

generating a second object type for a term in the role expression, and

adding a parameter having the second object type to the operation.

5. The method of claim 2 wherein parsing the propositional expression comprises:

reading in logical order a role expression in the propositional expression if the propositional expression includes an interrogative operator, the role expression being not a pronominal reference;

generating a second object type for a term in the role expression;

if the role expression has a parametric operator,

adding a parameter having the second object type to the operation;

else, if the role expression contains the interrogative operator,

adding an attribute to the object type designated by the current container having an attribute type to be a collection of objects of the second object type, and

setting the current container to the second object type.

6. The method of claim 5 wherein generating the second object type for the term comprises:

creating the second object type; and

if the term in an identification scheme of the term has a numerical or lexical type,

adding an attribute of the numerical or lexical type to the second object type;

else

generating a temporary object type for the term, and

adding an attribute having an attribute type to be same as the temporary object type to the second object type.

7. An article of manufacture comprising:

a machine-accessible medium including data that, when accessed by a machine, causes the machine to perform operations comprising:

creating an object type for a return type of an operation in a rule authorizing a request for information;

establishing a current container; and

processing a propositional expression in the rule to record the object type to be a resulting context from the propositional expression in the current container.

8. The article of manufacture of claim 7 wherein the data causing the machine to perform processing the propositional expression comprises data that, when accessed by the machine, causes the machine to perform operations comprising:

setting the current container to be the object type;

establishing a context for the propositional expression; and

parsing the propositional expression.

9. The article of manufacture of claim 8 wherein the data causing the machine to perform establishing the context for the propositional expression comprises data that, when accessed by the machine, causes the machine to perform operations comprising:

finding a pronominal reference in the propositional expression having a referent term to be a term in a previously processed propositional expression and not to be a subject of a parametric operator;

if the referent term is not subject of an interrogative operator,

setting the current container to be the object type of the context resulting from the propositional expression;

else setting the current container to be the object type generated for the referent term.

10. The article of manufacture of claim 8 wherein the data causing the machine to perform parsing the propositional expression comprises data that, when accessed by the machine, causes the machine to perform operations comprising:

reading in logical order a role expression in the propositional expression if a propositional interrogative is applied to the propositional expression, the role expression being not a pronominal reference and including no definite article or expression of a specifically referenced instance;

adding a Boolean attribute to the object type designated by the current container to indicate a truth value of the propositional expression; and

if the role expression has a parametric operator,
generating a second object type for a term in the role expression, and
adding a parameter having the second object type to the operation.

11. The article of manufacture of claim 8 wherein the data causing the machine to perform parsing the propositional expression comprises data that, when accessed by the machine, causes the machine to perform operations comprising:
reading in logical order a role expression in the propositional expression if the propositional expression includes an interrogative operator, the role expression being not a pronominal reference;
generating a second object type for a term in the role expression;
if the role expression has a parametric operator
adding a parameter having the second object type to the operation;
else, if the role expression contains the interrogative operator
adding a Boolean attribute to the object type designated by the current container having an attribute type to be a collection of objects of the second object type, and
setting the current container to the second object type.

12. The article of manufacture of claim 11 wherein the data causing the machine to perform generating the second object type for the term comprises data that, when accessed by the machine, causes the machine to perform operations comprising:
creating the second object type; and
if the term in an identification scheme of the term has a numerical or lexical type
adding an attribute of the numerical or lexical type to the second object type;
else
generating a temporary object type for the term; and
adding an attribute having an attribute type to be same as the temporary object type to the second object type.

13. A system comprising:
a processor; and

a memory coupled to the processor, the memory containing instructions that, when executed by the processor, cause the processor to:

create an object type for a return type of an operation in a rule authorizing a request for information,

establish a current container, and

process a propositional expression in the rule to record the object type to be a resulting context from the propositional expression in the current container.

14. The system of claim 13 wherein the instructions causing the processor to process the propositional expression comprises instructions that, when executed by the processor, cause the processor to:

set the current container to be the object type;

establish a context for the propositional expression; and

parse the propositional expression.

15. The system of claim 14 wherein the instructions causing the processor to establish the context for the propositional expression comprises instructions that, when executed by the processor, cause the processor to:

find a pronominal reference in the propositional expression having a referent term to be a term in a previously processed propositional expression and not to be a subject of a parametric operator;

if the referent term is not subject of an interrogative operator

set the current container to be the object type of the context resulting from the propositional expression;

else set the current container to be the object type generated for the referent term.

16. The system of claim 14 wherein the instructions causing the processor to parse the propositional expression comprises instructions that, when executed by the processor, cause the processor to:

read in logical order a role expression in the propositional expression if a propositional interrogative is applied to the propositional expression, the role expression being not a pronominal reference and including no definite article or expression of a specifically referenced instance;

add a Boolean attribute to the object type designated by the current container to indicate a truth value of the propositional expression; and

if the role expression has a parametric operator,

generate a second object type for a term in the role expression, and

add a parameter having the second object type to the operation.

17. The system of claim 14 wherein the instructions causing the processor to parse the propositional expression comprises instructions that, when executed by the processor, cause the processor to:

read in logical order a role expression in the propositional expression if the propositional expression includes an interrogative operator, the role expression being not a pronominal reference;

generate a second object type for a term in the role expression;

if the role expression has a parametric operator

add a parameter having the second object type to the operation;

else, if the role expression contains the interrogative operator,

add a Boolean attribute to the object type designated by the current container having an attribute type to be a collection of objects of the second object type, and

set the current container to the second object type.

18. The system of claim 17 wherein the instructions causing the processor to generate the second object type for the term comprises instructions that, when executed by the processor, cause the processor to:

create the second object type; and

if the term in an identification scheme of the term has a numerical or lexical type,

add an attribute of the numerical or lexical type to the second object type;

else

generate a temporary object type for the term; and

add an attribute having an attribute type to be same as the temporary object type to the second object type..

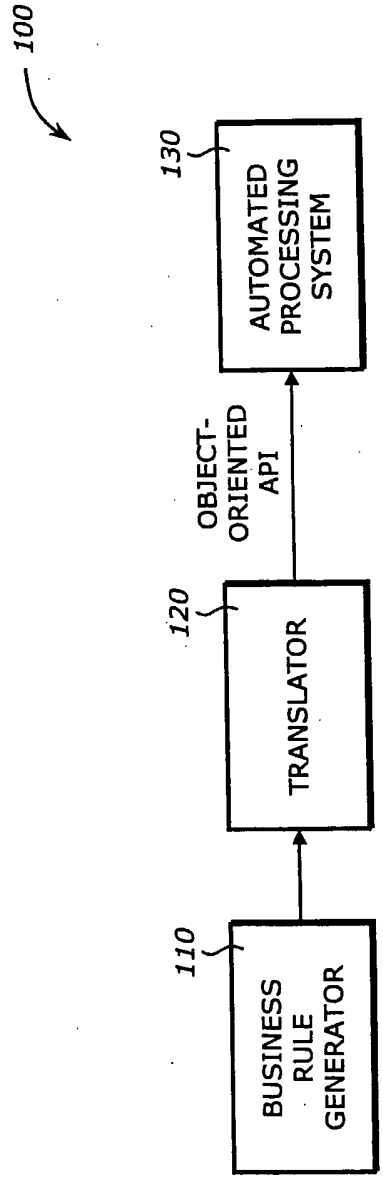
100
110
BUSINESS RULE GENERATOR
120
TRANSLATOR
OBJECT-ORIENTED API
130
AUTOMATED PROCESSING SYSTEM

FIG. 1

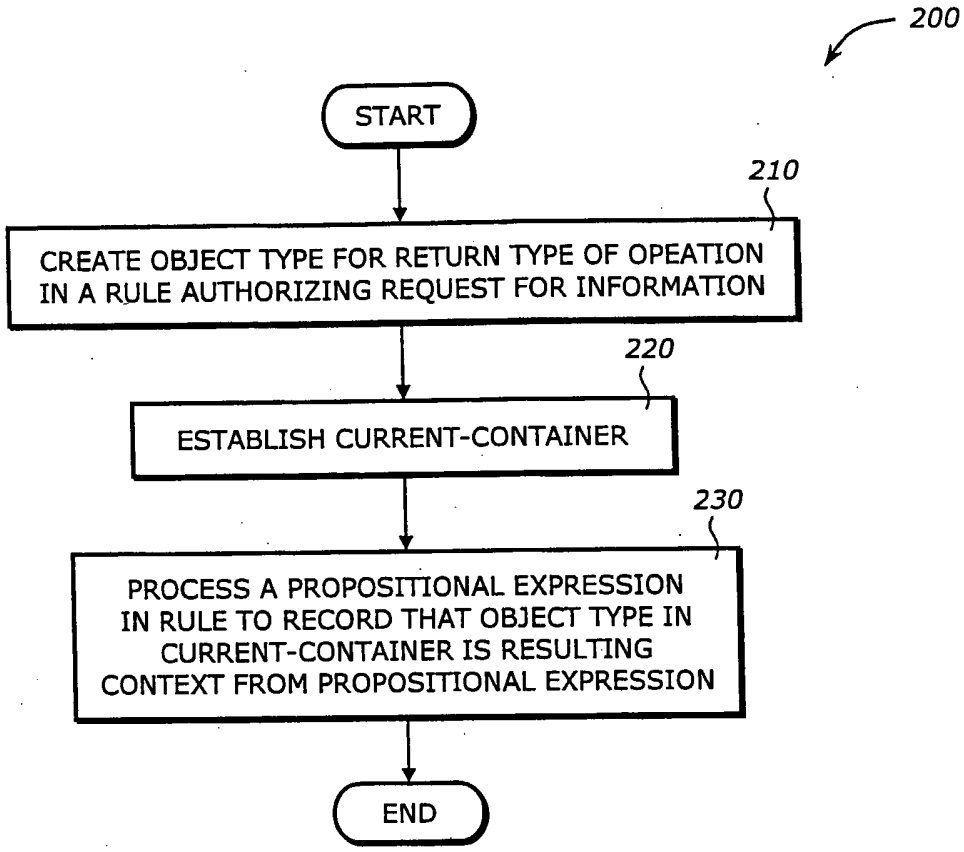
200
START
210
CREATE OBJECT TYPE FOR RETURN TYPE OF OPEATION IN A RULE AUTHORIZING REQUEST FOR INFORMATION
220
ESTABLISH CURRENT-CONTAINER
230
PROCESS A PROPOSITIONAL EXPRESSION IN RULE TO RECORD THAT OBJECT TYPE IN CURRENT-CONTAINER IS RESULTING CONTEXT FROM PROPOSITIONAL EXPRESSION
END

FIG. 2

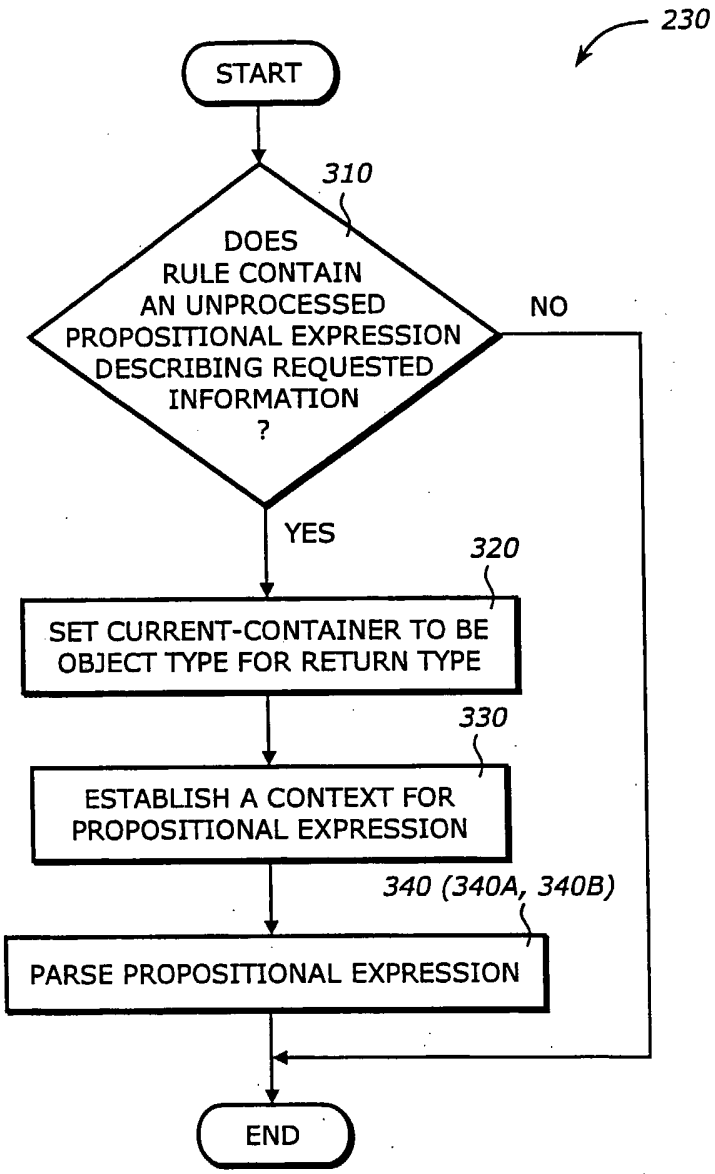
230
START
310
DOES RULE CONTAIN AN UNPROCESSED PROPOSITIONAL EXPRESSION DESCRIBING REQUESTED INFORMATION ?
NO
YES
320
SET CURRENT-CONTAINER TO BE OBJECT TYPE FOR RETURN TYPE
330
ESTABLISH A CONTEXT FOR PROPOSITIONAL EXPRESSION
340 (340A, 340B)
PARSE PROPOSITIONAL EXPRESSION
END

FIG. 3

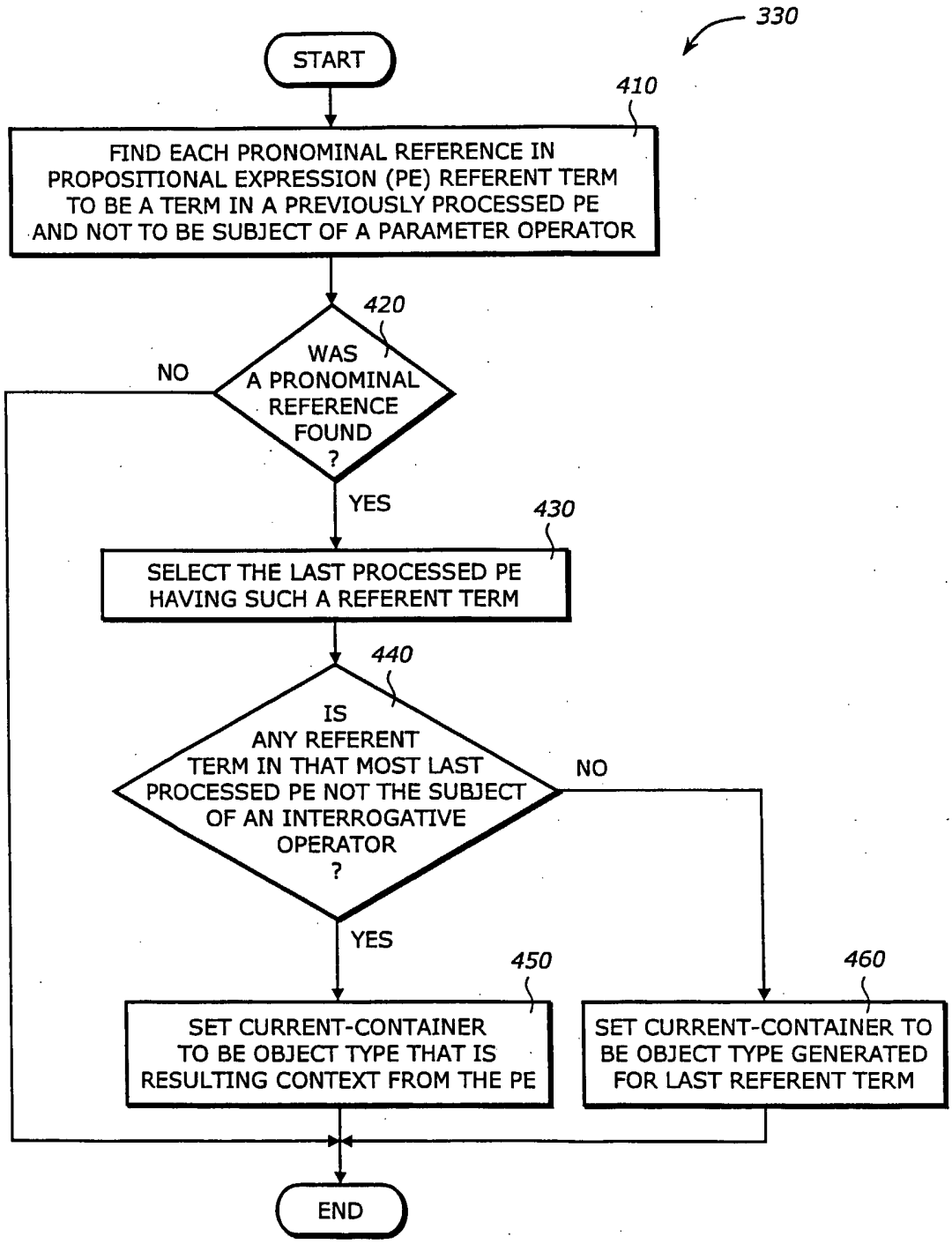
330
START
410
FIND EACH PRONOMINAL REFERENCE IN PROPOSITIONAL EXPRESSION (PE) REFERENT TERM TO BE A TERM IN A PREVIOUSLY PROCESSED PE AND NOT TO BE SUBJECT OF A PARAMETER OPERATOR
420
WAS A PRONOMINAL REFERENCE FOUND ?
NO
YES
430
SELECT THE LAST PROCESSED PE HAVING SUCH A REFERENT TERM
440
IS ANY REFERENT TERM IN THAT MOST LAST PROCESSED PE NOT THE SUBJECT OF AN INTERROGATIVE OPERATOR ?
NO
YES
450
SET CURRENT-CONTAINER TO BE OBJECT TYPE THAT IS RESULTING CONTEXT FROM THE PE
460
SET CURRENT-CONTAINER TO BE OBJECT TYPE GENERATED FOR LAST REFERENT TERM
END

FIG. 4

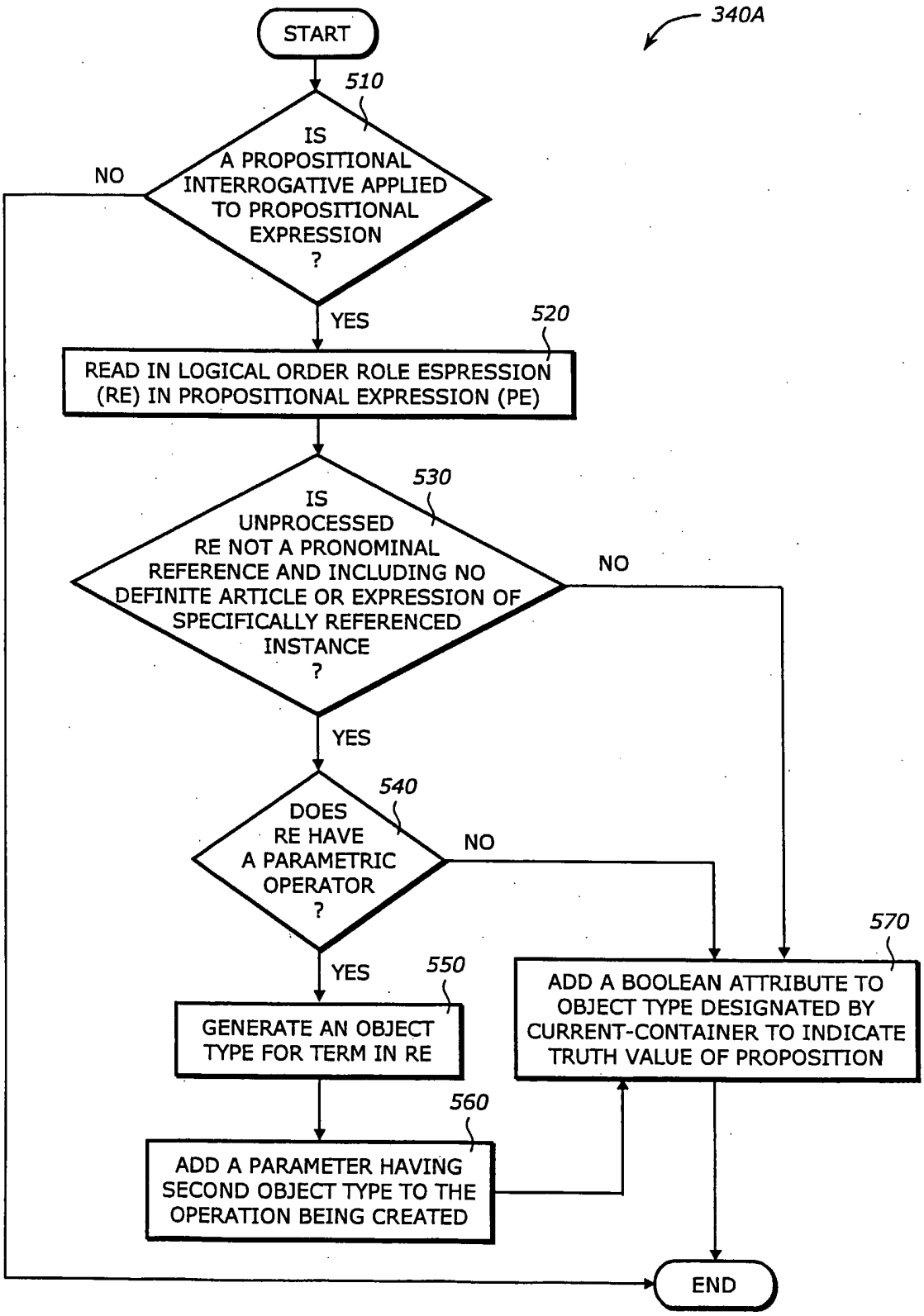
340A
START
510
IS A PROPOSITIONAL INTERROGATIVE APPLIED TO PROPOSITIONAL EXPRESSION ?
NO
YES
520
READ IN LOGICAL ORDER ROLE ESPRESSION (RE) IN PROPOSITIONAL EXPRESSION (PE)
530
IS UNPROCESSED RE NOT A PRONOMINAL REFERENCE AND INCLUDING NO DEFINITE ARTICLE OR EXPRESSION OF SPECIFICALLY REFERENCED INSTANCE ?
NO
YES
540
DOES RE HAVE A PARAMETRIC OPERATOR ?
NO
YES
550
GENERATE AN OBJECT TYPE FOR TERM IN RE
560
ADD A PARAMETER HAVING SECOND OBJECT TYPE TO THE OPERATION BEING CREATED
570
ADD A BOOLEAN ATTRIBUTE TO OBJECT TYPE DESIGNATED BY CURRENT-CONTAINER TO INDICATE TRUTH VALUE OF PROPOSITION
END

FIG. 5

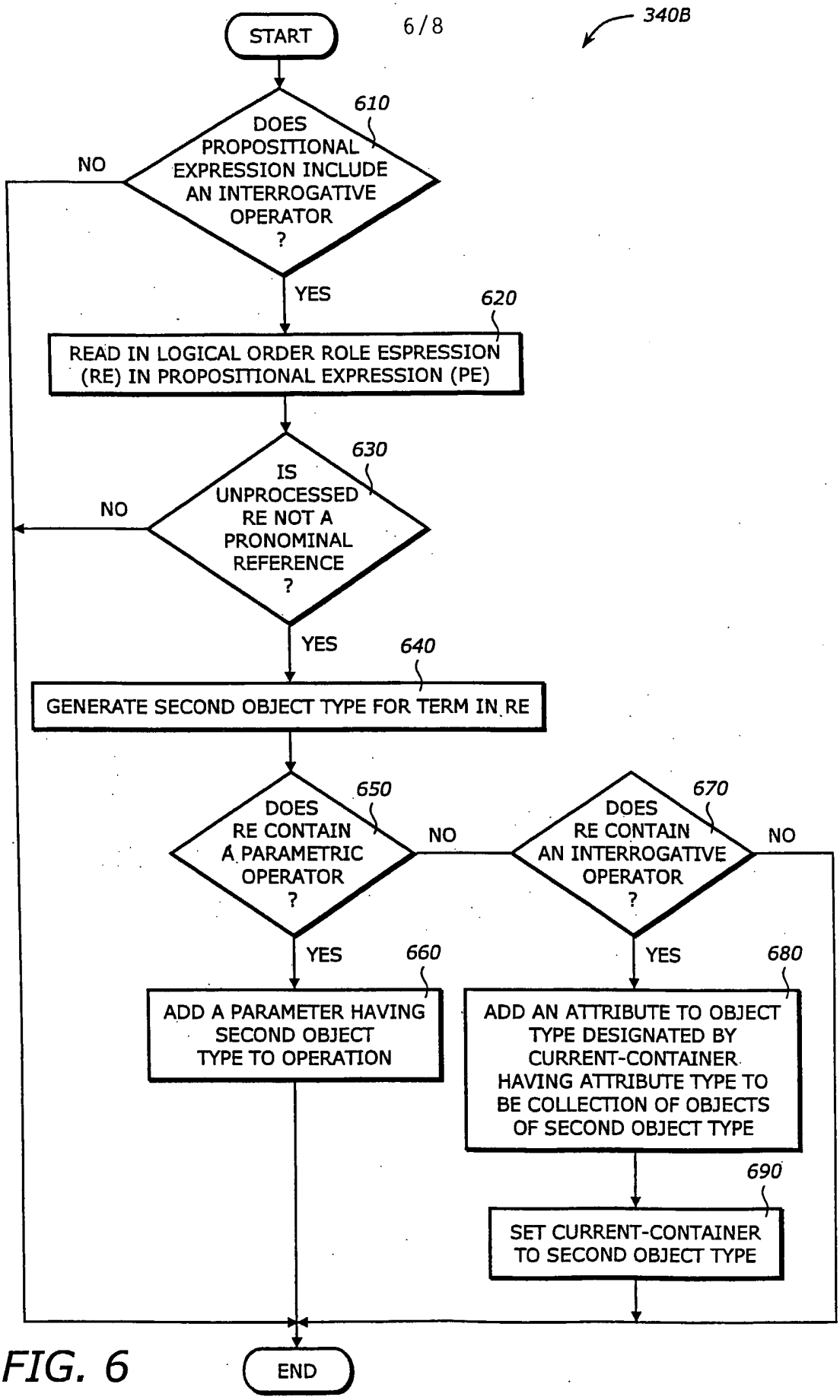
340B
START
610
DOES PROPOSITIONAL EXPRESSION INCLUDE AN INTERROGATIVE OPERATOR ?
NO
YES
620
READ IN LOGICAL ORDER ROLE ESPRESSION (RE) IN PROPOSITIONAL EXPRESSION (PE)
630
IS UNPROCESSED RE NOT A PRONOMINAL REFERENCE ?
NO
YES
640
GENERATE SECOND OBJECT TYPE FOR TERM IN RE
650
DOES RE CONTAIN A PARAMETRIC OPERATOR ?
NO
YES
660
ADD A PARAMETER HAVING SECOND OBJECT TYPE TO OPERATION
670
DOES RE CONTAIN AN INTERROGATIVE OPERATOR ?
NO
YES
680
ADD AN ATTRIBUTE TO OBJECT TYPE DESIGNATED BY CURRENT-CONTAINER HAVING ATTRIBUTE TYPE TO BE COLLECTION OF OBJECTS OF SECOND OBJECT TYPE
690
SET CURRENT-CONTAINER TO SECOND OBJECT TYPE
END

FIG. 6

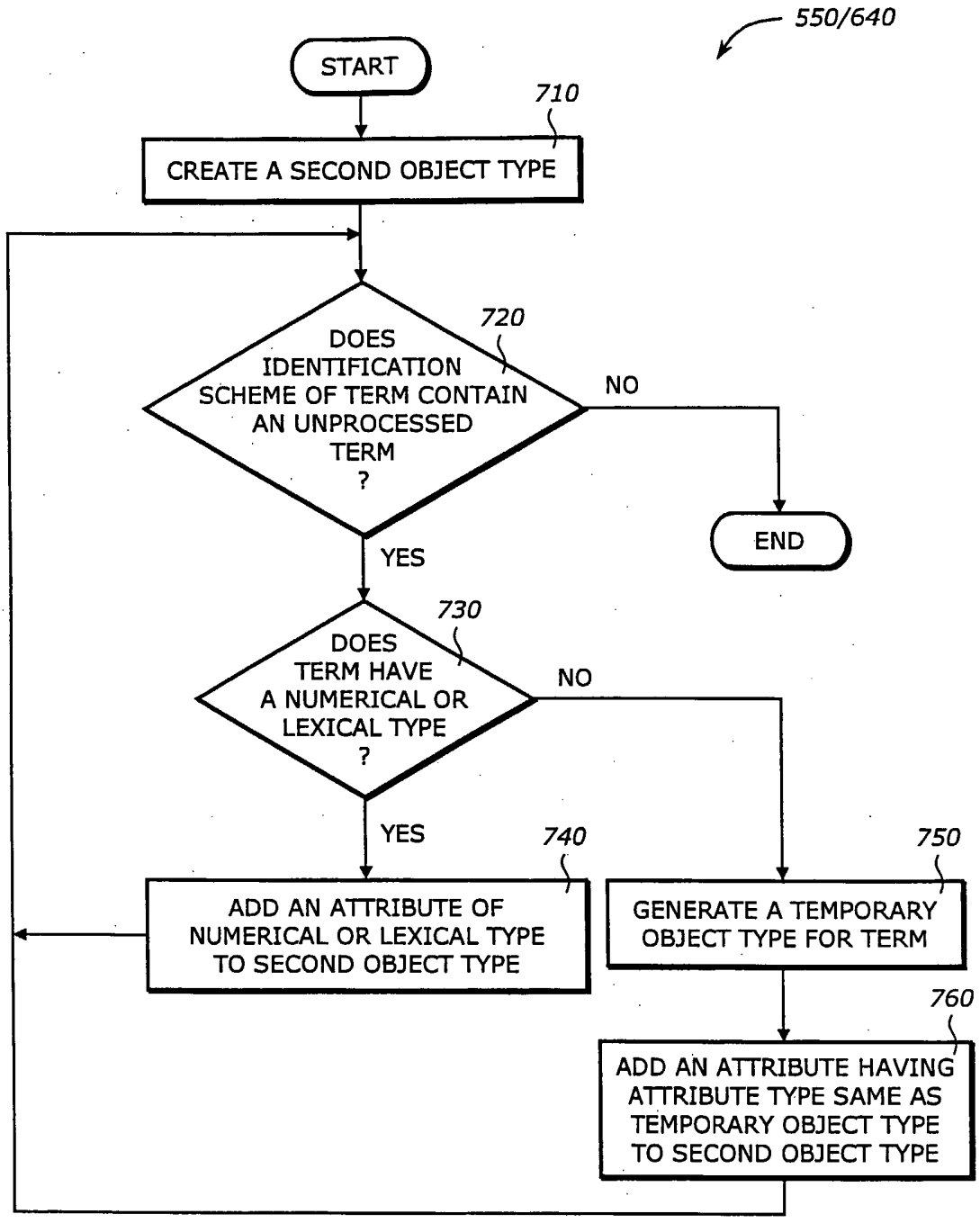
550/640
START
710
CREATE A SECOND OBJECT TYPE
720
DOES IDENTIFICATION SCHEME OF TERM CONTAIN AN UNPROCESSED TERM ?
NO
END
YES
730
DOES TERM HAVE A NUMERICAL OR LEXICAL TYPE ?
NO
YES
740
ADD AN ATTRIBUTE OF NUMERICAL OR LEXICAL TYPE TO SECOND OBJECT TYPE
750
GENERATE A TEMPORARY OBJECT TYPE FOR TERM
760
ADD AN ATTRIBUTE HAVING ATTRIBUTE TYPE SAME AS TEMPORARY OBJECT TYPE TO SECOND OBJECT TYPE

FIG. 7

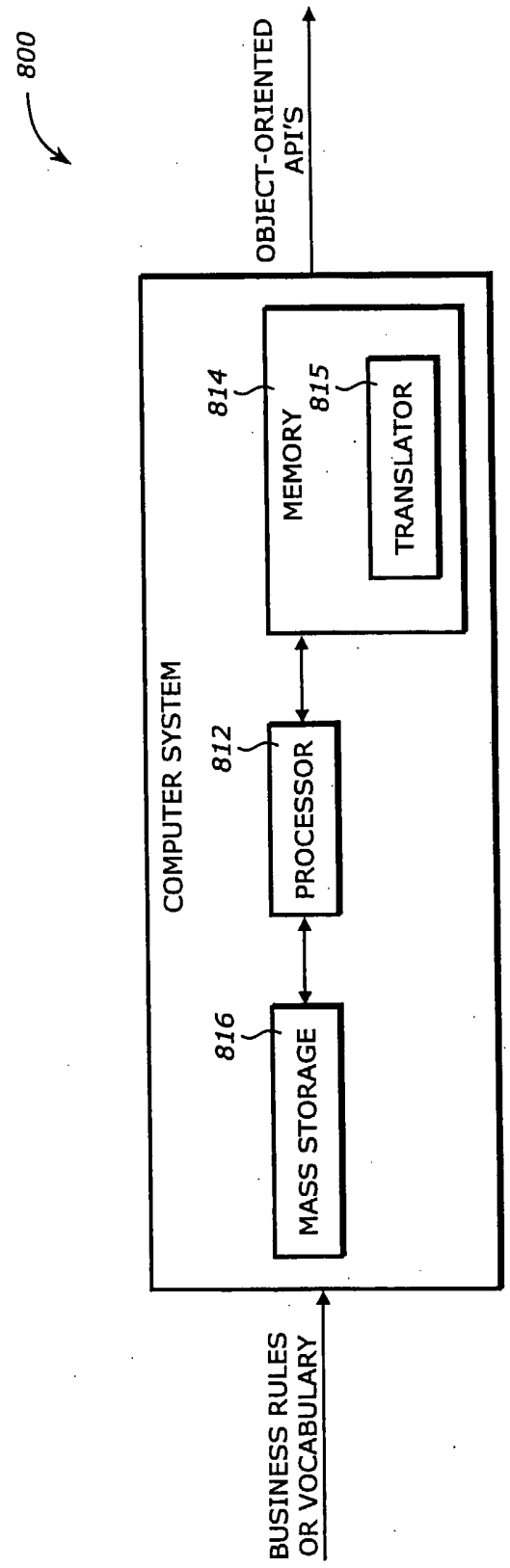
800
OBJECT-ORIENTED API'S
COMPUTER SYSTEM
814
MEMORY
815
TRANSLATOR
812
PROCESSOR
816
MASS STORAGE
BUSINESS RULES OR VOCABULARY

FIG. 8