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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
17 March 2005 (17.03.2005)

PCT

(10) International Publication Number
WO 2005/024719 A1

(51) International Patent Classification⁷: G06N 5/02, G06F 15/18, G06N 5/04

(21) International Application Number: PCT/US2004/028624

(22) International Filing Date: 2 September 2004 (02.09.2004)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/500,196 4 September 2003 (04.09.2003) US

(71) Applicant and
(72) Inventor: **HAMILTON, David, James** [US/US]; 13756 Bridlewood Drive, Gainesville, VA 20155 (US).

(74) Agent: **POMERANCE, Brenda**; Law Office of Brenda Pomerance, 260 West 52 Street Ste 27B, New York, NY 10019 (US).

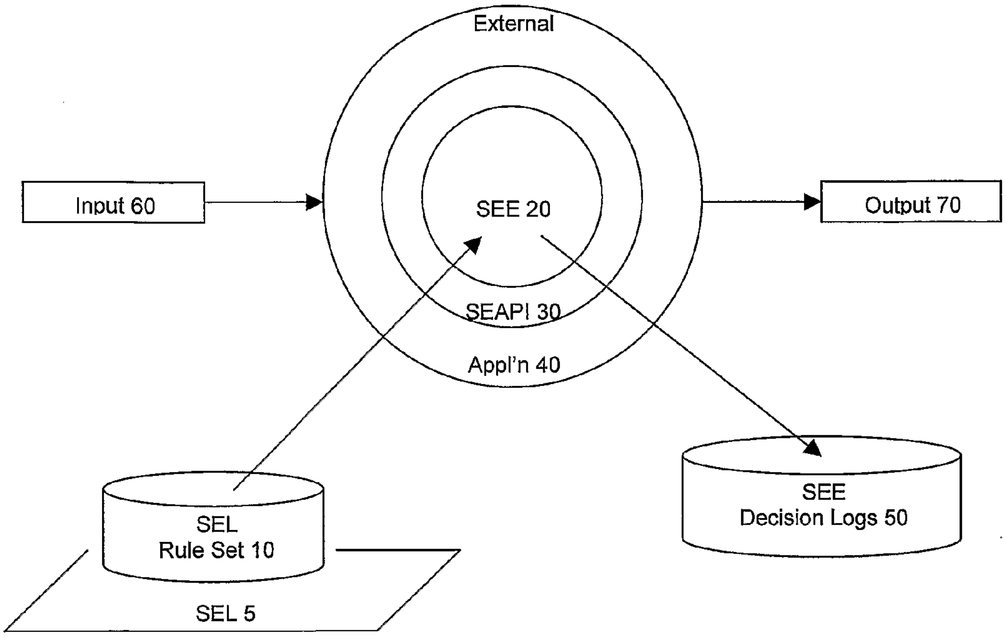
(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, 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, 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, IT, 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:
— *with international search report*

[Continued on next page]

(54) Title: EXPERT SYSTEM WITH SIMPLIFIED LANGUAGE



(57) **Abstract:** An inference engine (20) comprises a computer software program (20) for evaluating rules (10) written in accordance with a rule language syntax having a small number of BNF productions, a small number of terminal symbols, and being English-like. The rule language syntax enables a rule to be called from another rule, and the computer software program supports a predefined API (30) for communication with an external program (40).

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

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.

EXPERT SYSTEM WITH SIMPLIFIED LANGUAGE

BACKGROUND OF THE INVENTION

The present invention relates to expert systems, and more particularly, is directed to a language, inference engine and application programming interface having simplified functionality.

An expert system comprises a set of rules, usually representing industry specific knowledge, and a rules engine for executing the rules.

A rule engine may be viewed as a sophisticated if/then statement interpreter. The if/then statements that are interpreted are called rules. The "if" portions of rules contain conditions such as "shoppingCart.totalAmount > $100". The "then" portions of rules contain actions such as "recommendDiscount(5%)". The inputs to a rule engine are a ruleset and some data objects. The outputs from a rule engine are determined by the inputs and may include the originally inputted data objects with possible modifications, new data objects and side effects such as "sendMail('Thank you for shopping')."

Known expert systems include: OPS5 from Carnegie Mellon University; ART and ART-IM from Inference Corporation; CLIPS (C Language Integrated Production System) from NASA; Eclipse from The Haley Enterprise, Inc.; Blaze Advisor from Fair Isaac Corporation; JESS (Java Expert System Shell) from Sandia National Laboratories; AI::ExpertSystem::Simple from Peter Hickman, XML-based rule language, Tcl/Tk expshell.

Most Expert System languages are robust. The term robust is used here to mean that the language can support the implementation of large, real-world rule sets. These same languages, however, become unwieldy with size and, in some cases, must be augmented with rule set development tools. An example of rule set development tools can be found in the “Blaze Advisor” product from Fair Isaac. These tools comprise a set of software programs that enable a human user to view and update rules, and that automatically create charts showing the processing steps in a user-defined sequence, such as processing a loan application. Since a practical application typically contains thousands of rules, tools are needed to help manage the rule sets.

Backus Naur Form (BNF) is a formal notation to describe the syntax of a programming language. M. Marcotty & H. Ledgard, *The World of Programming*

Languages, Springer-Verlag, Berlin 1986., pages 41 and following. See also: http://cui.unige.ch/db-research/Enseignement/ analyseinfo/AboutBNF.html.

Each statement in BNF form is referred to as a "production" and the number of productions in a language is an indication of the complexity of the language. FIG. 1 shows CLIPS expressed in BNF. Since the JESS language is very similar to the language defined by the CLIPS expert system shell, which in turn is a highly specialized form of LISP, the characteristics of JESS are similar to the characteristics of CLIPS. FIG. 2 shows Blaze Advisor expressed in BNF. CLIPS has 74 BNF productions. Blaze Advisor has 57 BNF productions.

A terminal symbol is an element in a BNF statement that is not defined in terms of other elements. The number of terminal symbols in a language is another indication of the complexity of the language. CLIPS has 37 terminal symbols. Blaze Advisor has 120 terminal symbols.

The syntax of a language refers to the rules governing the formation of statements in the language, which are described using BNF. While programmers can adapt to a syntax optimized for a computer, such as the LISP-like syntax of CLIPS or the proprietary syntax of Blaze Advisor, most humans are used to thinking in English-like syntax and find the computer-optimized syntaxes to be burdensome.

Accordingly, CLIPS, JESS and Blaze Advisor are seen to be complicated languages that are burdensome for non-programmers to use. As a practical matter, the complexity of the languages necessitates use of a rule development toolkit. In turn, this makes use of the product feasible only for those prepared to invest the time in learning to use the toolkit, and actually using the toolkit frequently enough to remember its nuances.

Thus, there is a need for an expert system that is less onerous to use.

SUMMARY OF THE INVENTION

In accordance with an aspect of this invention, there is provided an inference engine, comprising a computer software program for evaluating rules written in accordance with a rule language syntax having a small number of BNF productions, a small number of terminal symbols, and being English-like.

In a further aspect of this invention, the rule language syntax enables a rule to be called from another rule, and the computer software program supports a predefined API for communication with an external program.

It is not intended that the invention be summarized here in its entirety. Rather, further features, aspects and advantages of the invention are set forth in or are apparent from the following description and drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a BNF definition of the C Language Integrated Production System (CLIPS) language;

Fig. 2 is a BNF definition of Blaze Advisor language;

Fig. 3 is a diagram showing the conceptual architecture of the expert system components according to the present invention;

Fig. 4 is a diagram showing the physical configuration in which an expert system according to the present invention operates;

Fig. 5 is a diagram showing the software architecture for application server 120;

Figs. 6A and 6B are BNF definitions of respective versions of a Simple Expert Language;

Figs. 7A-7H are flowcharts showing different state machines for the Expert Engine;

Fig. 8 is a detailed description of the customer service scenario for example 1;

Fig. 9 is a flowchart for the application workflow of example 1;

Fig. 10 is program code showing the operation of the VRU workflow expressed in SEL 5;

Fig. 11 is program code showing an example of rules encoded in SEL 5;

Fig. 12 is program code showing the rules of Fig. 11 organized into hierarchical files, and how these files are included in another file;

Fig. 13 shows program code in SEL 5 for solving the golfer problem; Fig. 14 shows program code in JESS for solving the golfer problem; and

Figs. 15A-15B are attribute data diagrams.

DETAILED DESCRIPTION

The Expert System described herein includes three components: an Expert Language, an Expert Engine and an Expert Application Programming Interface (API).

The Expert Language enables business knowledge to be represented in a form that a computer can readily process, referred to as a rule set. The Expert Language is used by humans to create rule sets.

The Expert Engine performs pattern matching between data presented and a rule set to determine which rules should be executed, and executes these rules. The Expert Engine is built in accordance with the Expert Language and is used by external computer programs. The Expert Engine is an instance of an inference engine.

The Expert API enables external computer programs to use the Expert Engine. Attributes passed to the Expert Engine via the Expert API from external computer programs are referred to as business attributes.

An embodiment of the Expert System, referred to as the Simple Expert System, will now be discussed.

Fig. 3 shows the conceptual architecture of the Simple Expert System. Simple Expert Language (SEL) 5 serves as a platform for creating rule set 10. Rule set 10 includes business rules and/or workflow rules. A business rule represents industry specific knowledge. A workflow rule relates to the order of the process flow for performing work. Application program 40 communicates, via Simple Expert API 30, with Simple Expert Engine (SEE) 20 to cause SEE 20 to execute appropriate rules in rule set 10 in dependance on input 20. SEE 20 generates decision logs 50 that identify its step by step activity. SEE 20's execution of rule set 10 generates output 70.

Fig. 4 shows the physical architecture of the Simple Expert System. Database server 100 is a general purpose computer programmed to store and provide data. Server 100 stores input 20 and output 70. Server 100 also stores rule set 10 and decision log 50. In some cases, server 100 stores different instances of rule set 10, for different applications, and correspondingly different instances of decision log 50. Server 10 is coupled to application server 120.

Application server 120 is a general purpose computer programmed to execute software programs residing thereon. Server 120 executes application program 40 and SEE 20. In some cases, server 120 stores different instances of application program 40, for different applications, and each of these instances of program 40 interacts with SEE 20. Only one version of SEE 20 is needed for different instances of program 40, and is invoked by the instances of program 40 using different instances of rule set 10. Server 120 is coupled to database server 100 and to firewall 130.

Firewall 130 is a general purpose computer programmed to filter external network traffic from application server 120 and web server 140, so that only desired traffic is presented to servers 120 and 140. Firewall 130 is coupled to servers 120, 140 and external network 150.

Web server 140 is a general purpose computer programmed to interact with remote browsers using a request/response type of protocol. For some requests, web server 140 sends a message to program 40, waits for a response from program 40, and uses the response from program 40 to respond to the browser's request. Web server 140 is coupled to firewall 130.

External network 150 may be the Internet or may be an intranet. External network 150 is coupled to user computers 160. Each user computer 160 is a general purpose computer programmed and is operative to execute a browser program, which sends requests via network 150 to server 140, and receives responses therefrom.

Fig. 5 shows the software architecture from the viewpoint of application server 120. Application server 120 executes an operating system used by application 40 and SEE 20.

Simple Expert Language

Fig. 6A is a BNF definition of SEL 5. SEL 5 is seen to have 13 BNF productions and 25 terminal symbols.

Since SEL 5 is compact, having a limited number of BNF productions and terminal symbols, it is readily learned and used by a non-programmer.

SEL 5 has an English-like structure. Specifically, statements have an antecedent and a consequence. An antecedent is the term "if" followed by an expression. A consequence is the term "then" followed by an expression. An expression has two ids, each being a value or a variable name, and a Boolean-type operator.

Since SEL 5 is an English-like rule language, SEL 5 does not require the use of rule set development tools to be effective. SEL 5 uses Boolean operators which are familiar logic constructs even for non-programmers. Thus, the rules themselves are sufficiently comprehensible by a non-programmer that the non-programmer can directly read and update the rules.

Although it may be possible to use a subset of constructs of a Prior Art Expert System to form the basis of a "simple" rule set implementation, the rule set still will not be clear. An implementation that is not clear will necessitate the use of programmers to author and maintain the rule sets, making them more expensive to implement and maintain.

SEL 5 facilitates representing knowledge as modules. Specifically, a rule id may be referenced in the antecedent or consequence of another rule. The rule id returns a value to the parent rule. Thus, antecedent clauses of rules may utilize attribute values set by consequences of different rules.

SEL 5 has a hierarchical structure. Specifically, SEL 5 has an include mechanism to facilitate the partitioning of rule sets across multiple files, shown as the keyword "file" in Fig. 6A. Accordingly, libraries of common rules sets can readily be implemented.

The rule-referencing ability of SEL 5, combined with its hierarchical structure, enables SEL 5 to handle large rule sets that are structured in a manner that a human can readily comprehend. Common sets of rules can be grouped together. Thus, do not need to be re-coded over and over again.

The hierarchical structure of SEL 5 combined with its compact size make SEL 5 easy to learn, and result in straightforward rule sets. In contrast, languages such as CLIPS, Jess, and Blaze Advisor have complex rule set, and are relatively difficult to learn, so that specialized knowledge engineers are required. SEL 5 can be effectively used by non-programmers, saving the expense of hiring knowledge engineers and avoiding information loss between the expert and the knowledge engineer.

SEL 5 supports internal attribute values. Internal attributes are attributes used in such a way as to not be set by the external application via the API. The syntax for internal attribute definition is identical to the syntax of input and output attribute definition . They are only internal attributes because of semantic interpretation, i.e. how they are used. Internal attributes are represented in byte code as attributes whose values are set by rules and not via the API. Internal attribute values are never passed in or passed back out of the expert engine, but are only used internally during inferencing. Groups of rules can be enabled or disabled by setting internal attribute values between inference engine states.

Internal attributes are used to establish hierarchies of rules to facilitate large rule sets. Large rule sets are grouped into two or more smaller rule sets of similar type. The order of evaluation is top down. A higher-level rule evaluation determines which lower level sets of rules are enabled. This is done through the use of internal attributes.

For example, in a customer service application, two types of rules could be used. One would be used to govern the customer interaction for a product under warranty, and another set used if the product is not under warranty. An internal attribute would be set by a higher-level rule that determined whether or not the product was under warranty. Based on the value of the internal attribute, one or the other lower-level sets of rules would be enabled.

The internal attribute mechanism of SEL 5 further improves its ability to support modular rule sets. Since SEL 5 supports modular knowledge, SEL 5 is robust.

Fig. 6B is a BNF definition of SEL 6. SEL 6 is seen to have 13 BNF productions and 25 terminal symbols. SEL 6 is similar to SEL 5, except that some of the terminal symbols and some of the keywords have different names.

Other embodiments of the Simple Expert Language will be apparent to those of ordinary skill in the art. A SEL according to the present invention has a small number of BNF productions, generally under about 20, and a small number of terminal symbols, generally under about 30.

Simple Expert Engine

SEE 20 performs the pattern matching of the rules to data as defined by the SEL instruction set. It is effectively the virtual machine that runs the bytecode of the SEL. All decisions made by SEE 20 are logged for audit and future reference purposes.

Fig. 7A is a flowchart showing the state machine for SEE 20. SEE 20 can be represented as a state machine since all of the rule antecedents are evaluated prior to performing any of the rule consequences. Order of rule evaluation is not important. In addition, SEE 20 performs both forward-chaining and backward-chaining. In forward-chaining, all of the attribute values are presented simultaneously to the engine prior to inferencing, so order of attribute evaluation is not important. In backward-chaining, however, where attribute values are being interactively requested, order is important. For backward-chaining, attribute order is specified via SEL 5. An instance of specifying the order of evaluating attributes is shown in Fig. 11, discussed below, specifically the numerical order in the "eval attr { ...}" statement. In Fig. 6A, the relevant keyword is "eval".

At step 205 of Fig. 7A, the internal attribute values are set by external program 40 in accordance with SEAPI 30. This step enables more modularization of rule sets. For example, a first set of rules can apply for one customer while a second set of rules can apply for another customer.

At step 210, SEE 20 disables all rules.

At step 215, SEE 20 enables only rules whose input attribute values have changed from the previous pass. For the first iteration, all rules having input attribute values depending on the input data are enabled.

At step 220, SEE 20 evaluates enabled rules and marks as fired or not-fired based on antecedent conditions. This is performed across all rules that should be evaluated, i.e., enabled. Consequences are only performed for those rules that have been marked as having fired. Thus it can be seen that the order of evaluation of these rules is not important during forward-chaining inferencing. In other words, evaluating an enabled rule is done to decide whether to look further at the rule, while marking a rule as fired or not-fired is done to determine whether or not its consequence should be performed.

At step 225, SEE 20 performs the consequences of rules marked as fired during the instant iteration of the flowchart of Fig. 7A.

At step 230, SEE 20 determines if any rules are marked as enabled. If so, processing returns to step 210. If no rules are marked as enabled, then processing proceeds to step 235.

At step 235, SEE 20 returns the results of its activity to external program 40 in accordance with SEAPI 30.

Other embodiments of the Simple Expert Engine will now be described.

Fig. 7B is a flowchart of the expert engine using the Rete algorithm, referred to as SEE 21. The Rete algorithm is described in various documents including the original paper by Charles L. Forgy, "Rete: A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem," Artificial Intelligence 19 (1982): 17-37. Order of rule evaluation is important in this embodiment in that the resultant pattern matching network will be different depending on the order of rule specification.

At step 250, the internal attribute values are set by external program 40 in accordance with SEAPI 30. This step enables more modularization of rule sets. For example, a first set of rules can apply for one customer while a second set of rules can apply for another customer.

At step 255, SEE 21 enables or disables rules based on attribute values.

At step 260, SEE 21 evaluates enabled rules and performs consequences based on antecedent conditions.

At step 265, SEE 21 enables or disables rules based on attribute values.

At step 270, SEE 21 determines if any rules are marked as enabled. If so, processing returns to step 250. If no rules are marked as enabled, then processing proceeds to step 275.

At step 275, SEE 21 returns the results of its activity to external program 40 in accordance with SEAPI 30.

Fig. 7C is a flowchart of the expert engine using the "rules finding facts" algorithm, referred to as SEE 22, as described in the book by Ernest Friedman-Hill, "Jess in Action: Rule-Based Systems in Java," Manning Publications Co. (2003): p135-136, is an inefficient embodiment of the Inference Engine. This algorithm simply evaluates every rule and performs the associated action iteratively until no further action is required. The inefficiency is in the fact that every rule is evaluated each pass which becomes combinatorially explosive in terms of computation time required to process large rule sets.

At step 290, the internal attribute values are set by external program 40 in accordance with SEAPI 30. This step enables more modularization of rule sets. For example, a first set of rules can apply for one customer while a second set of rules can apply for another customer.

At step 292, SEE 22 evaluates all rules and marks as fired or not-fired based on antecedent conditions.

At step 294, SEE 22 performs the consequence of any rule newly marked as fired during this iteration

At step 296, SEE 22 determines if any rules are newly marked as fired. If so, processing returns to step 290. If no rules are marked as fired, then processing proceeds to step 298.

At step 298, SEE 22 returns the results of its activity to external program 40 in accordance with SEAPI 30.

The differences between the SEE 20 (SEE), SEE 21 (Rete), and SEE 22 (rules finding facts) are as follows. SEE 22 evaluates all of the rules each pass and performs the associated actions. SEE 21 evaluates only enabled rules and performs the associated action for each rule before evaluating the next rule. SEE 20 evaluates only enabled rules but does not perform any associated actions until all of the rules have been evaluated.

SEE 20 and SEE 21 are more efficient than SEE 22 because they only evaluate enabled subsets of rules.

SEE 20 and SEE 22 are state-based and do not have a rule order-of-evaluation dependency. SEE 21 is not state-based and does have a rule order-of-evaluation dependency.

Figs. 7D-7F provide detail for a first embodiment of Fig. 7A, while Figs. 7G-7H provide detail for a second embodiment of Fig. 7A. Fig. 15A shows an attribute data diagram for the first embodiment. Fig. 15B shows an attribute data diagram for the second embodiment.

Fig. 7D provides detail for the first embodiment of Fig. 7A, step 215, which is concerned with enabling only those rules with input attribute value changes from a previous pass.

At step 330, the first rule is selected. At step 335, a list of relevant attribute objects in the rule's antecedent is acquired. At step 340, the first of these attributes is selected. At step 350, it is checked whether the previous pass attribute value is equal to the current pass attribute value. If not, then at step 345, the rule is enabled and processing continues at step 370. If so, then at step 360, it is checked whether there are any more attributes. If so, at step 355, the next attribute is selected and processing returns to step 350. If there are no more attributes for this rule, then at step 370, it is checked whether there are any more rules. If so, at step 365, the next rule is selected and processing returns to step 335. If there are no more rules, then processing is complete.

This first embodiment stores, for each attribute object, the previous and current value of the attribute object. It is believed that no other rules engine does this. Advantages of storing the previous and current values for an attribute object include: reduced search time, ease in locating changed attribute values, and the system's state is distributed into each object.

Fig. 7E provides detail for the first embodiment of Fig. 7A, step 220, which is concerned with evaluating enabled rules and marking them as fired on no-fired based on the rule's antecedent condition.

At step 375, the first rule is selected. At step 380, it is checked whether the rule was enabled during processing of Fig. 7D. If not, that is, there was no change, then at step 385, the rule is marked as not-fired and processing continues at step 400. If the rule was enabled, that is, there was a change in at least one attribute value, then at step 390. it is checked whether the rule's antecedent condition evaluates to true. If not, processing continues at step 385. If the rule's antecedent condition evaluates to true, then at step 395, the rule is marked as fired. At step 400, it is checked whether there are any more rules. If so, at step 405, the next rule is selected and processing continues at step 380. When there are no more rules, processing is complete.

Fig. 7F provides detail for the first embodiment of Fig. 7A, step 225, which is concerned with performing the consequence of each enabled rules marked as fired during this pass.

At step 410, the first rule is selected. At step 415, it is checked whether the rule was marked as fired during processing of Fig. 7E. If not, then processing continues at step 425. If the rule was marked as fired, then at step 420, the rule's consequence is performed. At step 425, it is checked whether there are any more rules. If so, then at step 430, the next rule is selected and processing continues at step 415. If there are no more rules, then processing is complete.

Fig. 7G provides detail for the second embodiment of Fig. 7A, steps 215, 220 and 225.

In the first embodiment, the steps of Fig. 7A are performed in the order of the first step for all rules, the next step for all rules and so on. In the second embodiment, the steps of Fig. 7A are performed in the order of, for the first rule, all steps, then for the second rule, all steps, and so on. Depending on the tradeoff between processing speed and memory cost, one or the other embodiments will be most optimal.

At step 435, the first rule is selected. At step 440, a list of relevant attribute objects in the rule antecedent is acquired. At step 445, the first attribute is selected. At step 455, it is determined whether the previous pass attribute value is equal to the current value. If not, then at step 450, the rule is enabled and processing continues at step 470. If the previous pass attribute value is equal to the current value, then at step 465, it is determined whether there are any more attributes. If so, at step 460, the next attribute is selected and processing returns to step 455. If there are no more attributes, processing continues at step 470.

At step 460, it is determined whether the rule is enabled. If not, processing continues at step 490. If the rule is enabled, then at step 480, it is determined whether the antecedent condition evaluates to true. If so, then at step 475, the rule's consequence is performed, and processing continues at step 490. If the antecedent condition does not evaluate to true, then processing continues at step 490.

At step 490, it is determined whether there are any more rules. If so, at step 485, the next rule is selected and processing continues at step 440. If not, then processing is complete.

Fig. 7H provides detail for the second embodiment of Fig. 7A, step 230, which is concerned with determining whether any rules are enabled.

At step 495, the first rule is selected. At step 500, it is determined whether the rule is enabled. If not, processing continues at step 515. If the rule is enabled, then at step 505, the future value is assigned to the current value. At step 515, it is determined whether there are any more rules. If so, then at step 510, the next rule is selected and processing continues at step 500. If not, then processing is complete.

Simple Expert API

SEAPI 30 conforms to a subset of the Java Rule Engine API standard currently being developed by the Java Community Process. www.jcp.org/jsr/detail/94.jsp

Example 1: Customer Service Interaction

An example will now be discussed in which a human has a problem with a product, calls a support telephone number for the product's supplier, and encounters a voice response unit (VRU) whose purpose is to ensure that only qualified product purchasers can talk to a technical support representative. The human interacts with the VRU. The VRU determines that the caller is a qualified product purchaser and forwards the human's call to a technical support representative (TSR). The TSR interacts with the caller to find a problem resolution.

Fig. 8 shows a step-by-step event log of the scenario for example 1.

To use business rules in an application requires an ordered processing mechanism called a workflow.

Fig. 9 is a flowchart for the application workflow of example 1 utilizing SEE 20 to perform both the evaluation of Business rules and Workflow rules within the Application's implementation. At step 305, external program 40 sets the business attributes to an initial state.

At step 310, program 40 runs SEE 20 with business rules. This step is done to compare the data presented with the business rules. At step 315, SEE 20 performs application actions based on business attribute values. This step amounts to performing appropriate ones of the consequences based on the data presented.

At step 320, program 40 runs SEE 20 with workflow rules. This step is done to figure out what to do next, such as when a computer is having a structured dialog with a user, and needs to figure out how to respond based on the user's input. At step 325, program 40 tests if the interaction is at an end. Typically, "end interaction" is a value associated with an attribute. If not, processing returns to step 305. If so, processing is complete.

Fig. 10 shows the operation of the VRU workflow expressed in SEL 5. Each of the attributes "state" and "next" can have one of the values: "vru", "human", "exit". The three workflow rules shown in Fig. 10 describe the next source of input, and when the VRU routine should exit.

Fig. 11 shows an example of rules encoded in SEL 5 for the VRU's operation.

Fig. 12 shows the rules of Fig. 11 organized into hierarchical files, specifically, the "language rules" file and the "fee" file, and also shows how these files are included in a "customer service" file. Fig. 12 demonstrates how SEL 5 supports representing knowledge in modular form.

Example 2: Math Class Problem

This example, from the book by Ernest Friedman-Hill, "Jess in Action: Rule-Based Systems in Java," Manning Publications Co. (2003): p4-8, contrasts te use of SEL with the Jess Rule Language to solve a non-intuitively obvious problem with a limited set of rules.

The problem is: in what order will four golfers tee off, and what color are each golfer's pants? The facts given are:

1. A foursome of golfers is standing at a tee, in a line from left to right. Each golfer wears different colored pants; one is wearing red pants. The golfer to Fred's immediate right is wearing blue pants.
2. Joe is second in line.
3. Bob is wearing plaid pants.
4. Tom isn't in position one or four, and he isn't wearing the hideous orange pants.

Fig. 13 shows program code in SEL 5 for solving the golfer problem. The code is seen to be English-like such that a non-programmer human can translate the statements into English, while being sufficiently definite that a computer can execute the code. Note that, at the conclusion of the program, values are assigned to attributes, and an external program is assumed to either use the values or display them to a human.

Fig. 14 shows program code in JESS for solving the golfer problem. The code is not English like, so a non-programmer cannot understand it. Note that the JESS program code is responsible for outputting the values, via "printout" statements. A programmer would consider this more powerful than SEL 5, as the output can be specified directly by JESS; however, the cost is that a non-programmer cannot directly modify the code.

Comparison of SEL 5 and Other Rule Languages

Table 1 compares SEL to CLIPS, JESS AND Blaze Advisor.

TABLE 1

EXPERT SYSTEM ATTRIBUTES	SEL	CLIPS	Jess	Blaze Advisor
Rule Language				
Implementation	SEL 5	CLIPS	Jess Rule Language	Structured Rule Language
Potential Embodiments	2+	1	1	1
Terminal Symbols	25	37	37+	120
BNF Productions	13	74	74+	57
Built-in Functions	0		176	32
Syntax Style	simple/clear	LISP-like	LISP-like	complex/proprietary
Inference Engine				
Implementation	**SEE**	Rete	Rete	proprietary
Potential Embodiments	3+	1	1	1
Performance	20x+	1x	20x	
API				
Implementation	standard		proprietary/ standard	proprietary
Potential Embodiments	2+		2	1

The 2+ Rule Language potential embodiments for SEL include SEL 5 and other languages having different syntax, but with the structured English style and other features of SEL 5, as shown in Figs. 6A and 6B.

The 3+ Inference Engine potential embodiments are "SEE," "Rete," "rules finding facts," as shown in Figs. 7A-7C, and any other inference engine implementation that can "run" the rules in a fashion consistent with the objective of the Simple Expert System.

The 2+ API potential embodiments are the "Java Rule Engine API" standard currently being discussed and another as-yet-undefined API standard.

Although illustrative embodiments of the present invention, and various modifications thereof, have been described in detail herein with reference to the accompanying drawings, it is to be understood that the invention is not limited to these precise embodiments and the described modifications, and that various changes and further modifications may be effected therein by one skilled in the art without departing from the scope or spirit of the invention as defined in the appended claims.

What is claimed is:

1. An inference engine, comprising:

a computer software program for evaluating rules written in accordance with a rule language syntax having a small number of BNF productions, a small number of terminal symbols, and being English-like.

2. The inference engine of claim 1, wherein the rule language syntax enables a rule to be called from another rule.

3. The inference engine of claim 1, wherein the computer software program supports a predefined API for communication with an external program.

FIG. 1 CLIPS

```
CLIPS Program
  <CLIPS-program> ::= <construct>*
  <construct>    ::= <deftemplate-construct> |
                 <deffacts-construct>  |
                 <defrule-construct>   |
                 <defmodule-construct>
Deftemplate Construct
  <deftemplate-construct> ::= (deftemplate <name>
[<comment>]
                        <slot-definition>*)
  <slot-definition>      ::= <single-slot-definition> |
                       <multislot-definition>
  <single-slot-definition> ::= (slot <slot-name>
                          <slot-attribute>*)
  <slot-name>           ::= <symbol>
  <multislot-definition>  ::= (multislot <slot-name>
                            <slot-attribute>*)
  <slot-attribute>       ::= <type-attribute>          |
                       <allowed-constant-attribute> |
                       <range-attribute>         |
                       <cardinality-attribute>    |
                       <default-attribute>
  <type-attribute>       ::= (type <type-specification>)
  <type-specification>    ::= <allowed-type>+ | ?VARIABLE
  <allowed-type>        ::= SYMBOL | STRING | LEXEME |
                       INTEGER | FLOAT | NUMBER
  <allowed-constant-attribute>
                    ::= (allowed-symbols <symbol-list>)  |
                       (allowed-strings <string-list>)  |
                       (allowed-lexemes <lexeme-list>   |
                       (allowed-integers <integer-list>) |
                       (allowed-floats <float-list>)    |
                       (allowed-numbers <number-list>)  |
                       (allowed-values <value-list>)
  <symbol-list>        ::= <symbol>+  | ?VARIABLE
  <string-list>        ::= <string>+  | ?VARIABLE
  <lexeme-list>        ::= <lexeme>+  | ?VARIABLE
  <integer-list>       ::= <integer>+  | ?VARIABLE
  <float-list>        ::= <float>+   | ?VARIABLE
  <number-list>        ::= <number>+   | ?VARIABLE
  <value-list>        ::= <constant>+ | ?VARIABLE
  <range-attribute>     ::= (range <range-specification>
                       <range-specification>)
  <range-specification>  ::= <number> | ?VARIABLE
  <cardinality-attribute> ::= (cardinality <cardinality-
specification>
                       <cardinality-specification>)
  <cardinality-specification> ::= <integer> | ?VARIABLE
  <default-attribute>    ::= (default <default-item>) |
                       (default-dynamic <expression>*)
  <default-item>       ::= ?DERIVE | ?NONE | <expression>*
Deffacts Construct
  (deffacts-construct>   ::= (deffacts <name> [<comment>]
                        <RHS-pattern>*)
Defrule Construct
  <defrule-construct>    ::= (defrule <name> [<comment>]
                        [<declaration>]
                        <conditional-element>*
                        =>
                        <expression>*)
  <declaration>        ::= (declare <rule-property>+)
  <rule-property>       ::= (salience <integer-expression>) |
                       (auto-focus <boolean-symbol>)
  <boolean-symbol>     ::= TRUE | FALSE
  <conditional-element>  ::= <pattern-CE>          |
                       <assigned-pattern-CE> |
                       <test-CE>          |
                       <not-CE>           |
                       <and-CE>           |
                       <or-CE>           |
                       <logical-CE>        |
```

```
                     <exists-CE>        |
                     <forall-CE>
  <pattern-CE>        ::= <ordered-pattern-CE> |
                     <template-pattern-CE>
  <assigned-pattern-CE>  ::= <single-field-variable> <-
<pattern-CE>
  <test-CE>          ::= (test <function-call>)
  <not-CE>           ::= (not <conditional-element>)
  <and-CE>           ::= (and <conditional-element>+)
  <or-CE>           ::= (or <conditional-element>+)
  <logical-CE>        ::= (logical <conditional-element>+)
  <exists-CE>        ::= (exists <conditional-element>+)
  <forall-CE>        ::= (forall <conditional-element>
                         <conditional-element>+)
  <ordered-pattern-CE>   ::= (<symbol> <constraint>+)
  <template-pattern-CE>  ::= (<deftemplate-name <LHS-
slot>*)
  <LHS-slot>         ::= <single-field-LHS-slot> |
                     <multifield-LHS-slot>
  <single-field-LHS-slot>  ::= (<slot-name> <constraint>)
  <multifield-LHS-slot>   ::= (<slot-name> <constraint>*)
  <constraint>        ::= ? | $? | <connected-constraint>
  <connected-constraint>  ::= <single-constraint> |
                     <single-constraint> & <connected-
constraint>|
                     <single-constraint> | <connected-constraint>
  <single-constraint>    ::= <term> | ~<term>
  <term>            ::= <constant>    |
                     <variable>     |
                     :<function-call> |
                     =<function-call>
Fact Specification
  <RHS-pattern>       ::= <ordered-RHS-pattern> |
                     <template-RHS-pattern>
  <ordered-RHS-pattern>   ::= (<symbol> <RHS-field>+)
  <template-RHs-pattern>  ::= (<deftemplate-name> <RHS-
slot>*)
  <RHS-slot>         ::= <single-field-RHS-slot> |
                     <multifield-RHS-slot>
  <single-field-RHS-slot>  ::= (<slot-name> <RHS-field>)
  <multifield-RHS-slot>   ::= (<slot-name> <RHS-field>*)
  <RHS-field>        ::= <variable> |
                     <constant> |
                     <function-call>
  <deftemplate-name>     ::= <symbol>

Varlables and Expressions
  <single-field-variable>  ::= ?<variable-symbol>
  <multifield-variable>   ::= $?<variable-symbol>
  <variable>         ::= <single-field-variable> |
                     <multifield-variable>
  <function-call>      ::= (<function-name> <expression>*) |
                      <special-function-call>
  <special-function-call>  ::= (assert <RHS-pattern>+)          |
                     (modify <expression> <RHS-slot>+)  |
                     (duplicate <expression> <RHS-slot>+) |
                     (bind <single-field-variable> <expression>)
  <expression>        ::= <constant> |
                     <variable> |
                     <function-call>
Data Types
  <symbol>       ::= A valid symbol
  <string>       ::= A valid string
  <float>       ::= A valid float
  <integer>      ::= A valid integer
  <number>       ::= <float> | <integer>
  <lexeme>       ::= <symbol> | <string>
  <constant>      ::= <number> | <lexeme>
  <comment>      ::= <string>
  <variable-symbol> ::= A <symbol> beginning with an
                 alphabetic character
  <function-name>  ::= <symbol>
  <name>         ::= <symbol>
```

FIG. 2 Blaze Advisor (page 1 of 4)

Reference: "Advisor Rules Syntax", Blaze Software, 1999, Pages 1-57 ("This document describes the syntax you use to write an Advisor rulebase. *Syntax* represents the laws of a language. In Advisor, the language you use is the Advisor *Structured Rule Language* (SRL).")

Rulebase
Object_Model [Rulesets]...[Event_Rules]...[Functions]...

Class_Definition
a[n] className is a[n] parent_className [with Property_Declaration_List]
[Initialize_Statement]

Object_Declaration
objectName is a[n] className [with Property_Declaration_List]
[Initialize_Statement]
objectName is an association from className | interfaceName | Primitive_Type to className | interfaceName | Primitive_Type
[with Property_Declaration_List]
[initially { it[key] = value,
it[key] = value,
...
}]
objectName is a[n] array of className | interfaceName | Primitive_Type [with Property_Declaration_List]
[initially { it.append(value) | it.insert(value),
it.append(value) | it.insert(value),
...
}]
objectName is a[n] fixed array of className | interfaceName | Primitive_Type [with Property_Declaration_List]
[initially { it[key] = value,
it[key] = value,
...
}]

Property_Declaration_List
{ Property_Declaration [, Property_Declaration] ... }
Property_Declaration
a[n] propertyName : a[n] Primitive_Type
a[n] propertyName : a[n] enumerationName
or
a[n] enumerationName
a[n] propertyName : some className | interfaceName
or
some className | interfaceName
a[n] propertyName : some association from className | interfaceName |
Primitive_Type to className | interfaceName | Primitive_Type
a[n] propertyName : some [fixed] array of className | interfaceName |
Primitive_Type

Initialize_Statement
initially { propertyName = Primary_Expression [, propertyName =
Primary_Expression] ... }
initially { Method_Call | Execute_Function | Create_Statement }
where Statement is one of the statements listed below.

Enumeration_Declaration
a[n] enumerationName is one of { itemName , itemName [, itemName] ... } .

Pattern_Declaration
patternName is any className | interfaceName [in collectionName]
[such that it. propertyName Comparison_Operator Primary_Expression
[and | or it. propertyName Comparison_Operator Primary_Expression]] .

Variable_Declaration
variableName is a[n] Primitive_Type
[initially true | false | unavailable | null | value] .
variableName is a[n] enumerationName
[initially enumerationItemName] .
variableName is some className | interfaceName
[initially objectExpression] .
variableName is some association from className | interfaceName |
Primitive_Type to className | interfaceName | Primitive_Type
[initially objectExpression] .
variableName is some [fixed] array of className | interfaceName |
Primitive_Type [initially objectExpression] .

Primitive_Type
boolean | integer | real | string | timestamp | date | time | duration | money

Ruleset_Definition
ruleset rulesetName [for { Parameter_Declarations }]
[returning a[n] Primitive_Type| className| interfaceName]
is { Ruleset_Body_Definition return [Primary_Expression] }

Ruleset_Body_Definition
[Object_Declaration] [Pattern_Declaration]
[Variable_Declaration]
Rule_Definition [Rule_Definition] ...
[Property_Event_Rule | Object_Event_Rule |
External_Event_Rule]

Rule_Definition
[rule ruleName [Rule_Priority] is] Rule_Body_Definition .

Rule_Priority
at priority n | at immediate priority [n]

Rule_Body_Definition
if Boolean_Expression then Statement_Block
[else Statement_Block]

Property_Event_Rule
[event rule ruleName is]
whenever the propertyName of className is changed | needed
do { Statement_Block }[.]
or
[event rule ruleName is]
whenever a[n] className.propertyName is changed | needed
do { Statement_Block }[.]

Object_Event_Rule
[event rule ruleName is]
whenever a[n] className is created | initialized | deleted
do { Statement_Block }[.]

External_Event_Rule
[event rule ruleName is]

FIG. 2 Blaze Advisor (page 2 of 4)

whenever a[n] className occurs
do { Statement_Block }[.]

Is_Changed_Property_Operators
old propertyName
new propertyName

Raw_Operator
raw Expression

Expressions
Primary_Expression
Boolean_Expression | Comparison_Expression |
Numeric_Expression |
Quantified_Expression | Literal_Value

Boolean_Expression
true | false
Comparison_Expression
not Boolean_Expression
Boolean_Expression and Boolean_Expression
Boolean_Expression or Boolean_Expression

Comparison_Expression
Primary_Expression Comparison_Operator Primary_Expression
Primary_Expression Comparison_Operator Literal_Value
where the two primary expressions are of compatible datatypes.

Numeric_Expression
Property_Value | value
Numeric_Expression Numeric_Operator Numeric_Expression
where both Property_Value and *value* evaluate to numeric values.

Quantified_Expression
at least n className [such that Boolean_Expression] satisfy
Boolean_Expression
at most n className [such that Boolean_Expression] satisfy
Boolean_Expression
every className [such that Boolean_Expression] satisfies
Boolean_Expression
exactly n className [such that Boolean_Expression] satisfies
Boolean_Expression
or
at least n elementTypeName in collectionName [such that
Boolean_Expression]
satisfy Boolean_Expression
at most n elementTypeName in collectionName [such that
Boolean_Expression]
satisfy Boolean_Expression
every elementTypeName in collectionName [such that
Boolean_Expression]
satisfies Boolean_Expression
exactly n elementTypeName in collectionName [such that
Boolean_Expression]
satisfies Boolean_Expression

Literal_Value
true | false | unavailable | unknown | available | known | null |
value

Property_Value
variableName | objectExpression. propertyName
or
the propertyName of objectExpression | variableName
or
the propertyName [of the objectPropertyName] of
objectExpression |
variableName
Default_Property_Value
Indexed_Property_Value

Default_Property_Value
objectExpression.$value
or
objectExpression.$object

Indexed_Property_Value
objectExpression. propertyName[index]

Statement_Block
Statement
or
{ Statement [, Statement]... }

Statement
Apply_Ruleset | Assignment_Statement |
Compound_Assignment_Statement |
Create_Statement | Delete_Statement | Execute_Function |
Method_Call |
Case_Selection | If_Then_Else | For_Each | While_Do |
Until_Do

Apply_Ruleset
apply rulesetName [with { Parameter_Bindings }]
Statements
or
rulesetName(Argument_List)

Assignment_Statement
Property_Value = Primary_Expression | true | false | unavailable
|
null | value

Built_In
calendar()
currencies()
events()
executeAgent()
exit()
print("Literal_Value")
print(Property_Value)
print(Property_Value "Literal_Value")
print([Property_Value] "" Property_Value ...)
print(Property_Value as a string)
promptBoolean(" Your prompt text here.")
promptInteger(" Your prompt text here.")
promptReal(" Your prompt text here.")
promptString(" Your prompt text here.")
promptEnumerationItem(enumName, " Your prompt text
here.") as an enumName
promptObject(className, " Your prompt text here.") as a
className
ignore(patternName)
objectMapper()

Compound_Assignment_Statement
Property_Value += Primary_Expression | value
Property_Value -= Primary_Expression | value
Property_Value *= Primary_Expression | value
Property_Value /= Primary_Expression | value

Create_Statement
a[n] className [Initialize_Statement]

Delete_Statement
delete objectExpression
delete objectExpression. propertyName
delete the propertyName of objectExpression

Execute_Function
execute functionName [with { Parameter_Bindings }]
or

FIG. 2 Blaze Advisor (page 3 of 4)

functionName(Argument_List)

Method_Call
className. methodName (Argument_List)
objectExpression. methodName (Argument_List)

Functions
Function_Definition
function functionName [for { Parameter_Declarations }]
[returning a[n] Primitive_Type| className | interfaceName |
collectionName]
is { Statement_Block [return Primary_Expression] }

Parameter_Declarations
parameterName : a[n] Primitive_Type | className |
interfaceName |
some association from className to className |
some [fixed] array of className | Primitive_Type
[, parameterName : a[n] Primitive_Type | className |
interfaceName |
some association from className to className |
some [fixed] array of className | Primitive_Type]
...

Parameter_Bindings
parameterName = Primary_Expression [, parameterName =
Primary_Expression]...

Argument_List
[Primary_Expression [, Primary_Expression]...]

Case_Selection
select Primary_Expression
case Primary_Expression : Statement_Block
case Primary_Expression : Statement_Block
...
otherwise : Statement_Block

For_Each
for each className | interfaceName [such that
Boolean_Expression]
do Statement_Block
for each elementTypeName in collectionName [such that
Boolean_Expression]
do Statement_Block

If_Then_Else
if Boolean_Expression then Statement_Block [else
Statement_Block]

While_Do
Until_Do
while Boolean_Expression do Statement_Block
until Boolean_Expression do Statement_Block

Try_Catch_Finally_Statement
try Statement_Block
catch a[n] exceptionclassName with Statement_Block
[finally Statement_Block]

Throw_Statement
throw exceptionExpression

Operators
Assignment_Operator
=

Boolean_Operator
not | or | and

Comparison_Operator
is | <> | < | > | <= | >=

Compound_Assignment_Operator
+= | -= | *= | /=

Numeric_Operator
+ | - | * | / | div | mod

Keywords
a, an
array
association from
Xxx to Xxx
fixed array
initially
is a, is an
is any
is one of
is some
it, he, she
some
such that
with
apply
at immediate
priority
at least...
satisfy...
at most...
satisfy...
at priority
does not apply
every... satisfy...
exactly...
satisfy....
for
if
is
is changed
is needed
return
returning
rule
ruleset
the ... of ...
then
do
event rule
is changed
is created
is deleted
is initialized
is needed
new
occurs
old
raw
whenever
case
do
for each
in
otherwise
select
until
while
execute
function
return
returning
catch
finally

FIG. 2 Blaze Advisor (page 4 of 4)

```
throw
try
and
as
delete
div
mod
not
or
is
=
<>
<
>
<=
>=
""
+
-
*
/
+=
-=
*=
/=
```

Builtin Methods

```
calendar
ceil
currencies
debugString
events
executeAgent
exit
floor
garbageCollect
ignore .
objectMapper
Datatypes
print
promptBoolean
promptObject
promptEnumerationItem
promptInteger
promptReal
promptString
round
stop
truncate
boolean
date
duration
integer
money
object
real
string
time
timestamp
```

FIG. 3

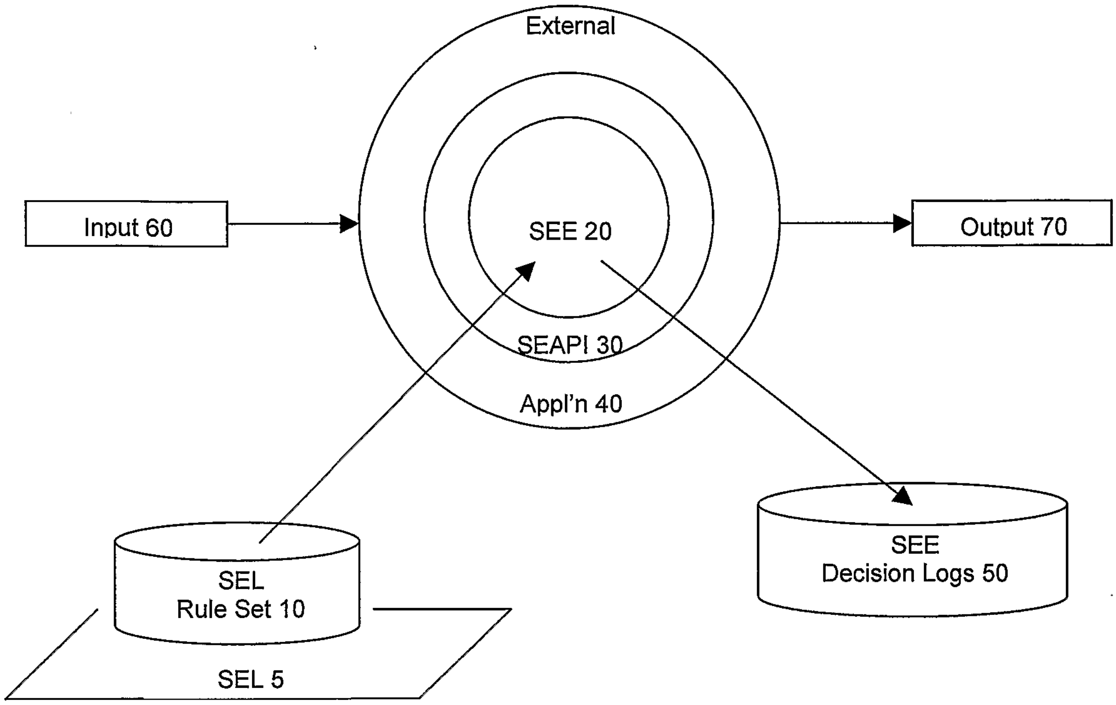

External
Input 60
SEE 20
Output 70
SEAPI 30
Appl'n 40
SEE
Decision Logs 50
SEL
Rule Set 10
SEL 5

FIG. 4

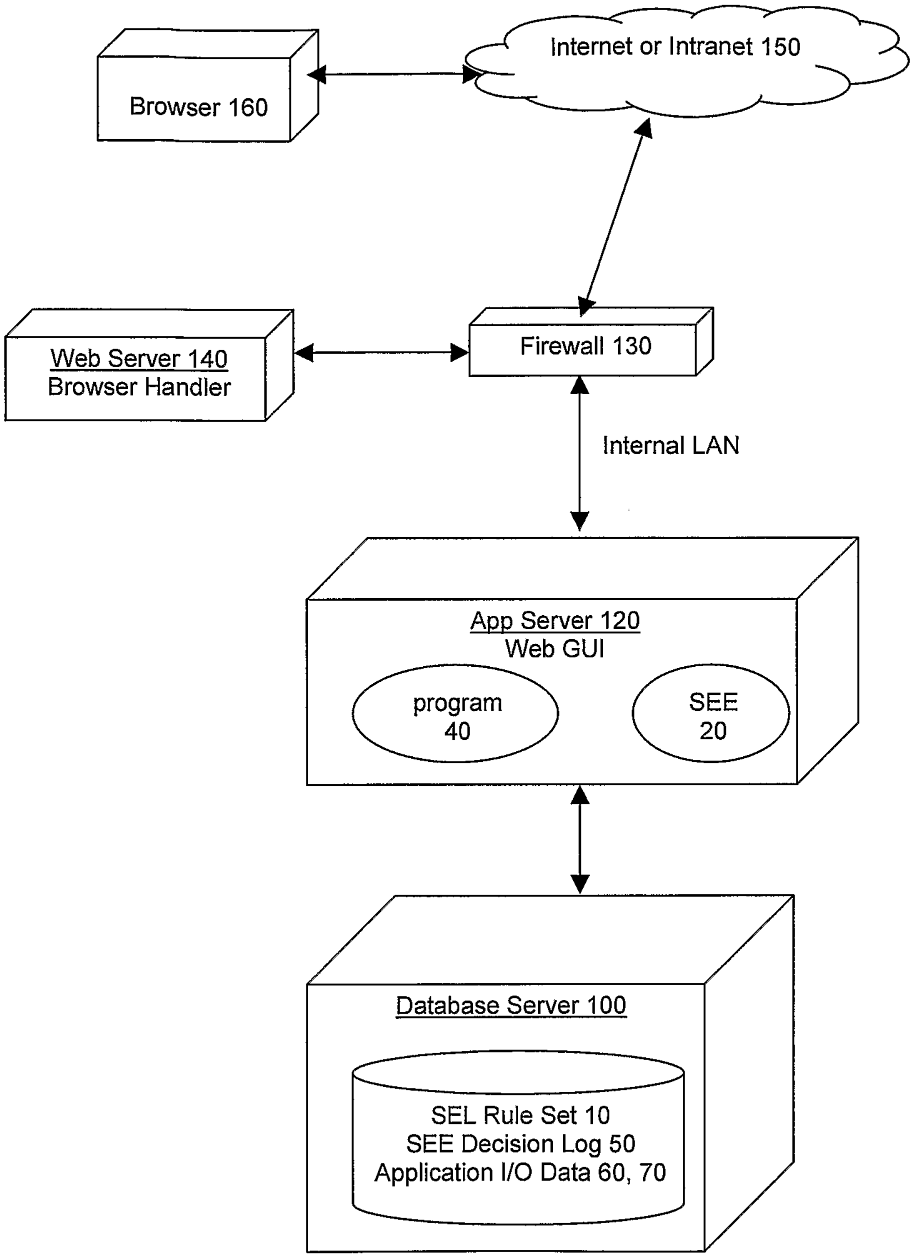

Browser 160
Internet or Intranet 150
Web Server 140
Browser Handler
Firewall 130
Internal LAN
App Server 120
Web GUI
program 40
SEE 20
Database Server 100
SEL Rule Set 10
SEE Decision Log 50
Application I/O Data 60, 70

FIG. 5

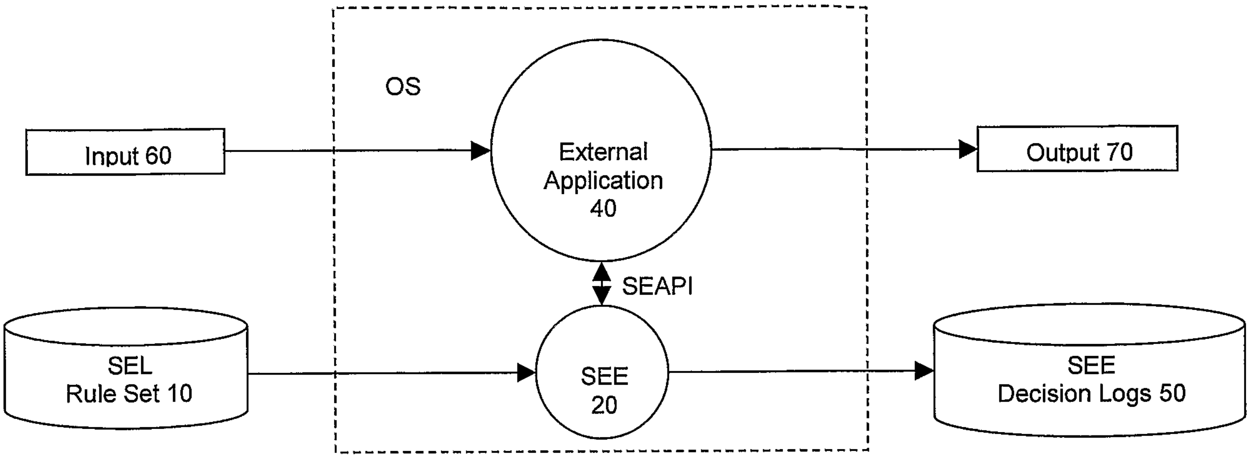

OS
Input 60
External
Application
40
Output 70
SEAPI
SEL
Rule Set 10
SEE
20
SEE
Decision Logs 50

FIG. 6A

```
<SEL_rule_set> ::=  <statement>*

<statement>   ::=   (<key_word> <id> <block>) |
                    <comment>

<block>      ::=    “{“ <phrase>+ “}” |
                    <list>

<phrase>     ::=    (<id> <list>) |
                    (<antecedent> <consequence>)

<antecedent>  ::=   if <expression>+

<consequence> ::=   then <expression>+

<expression>  ::=   (<id> <operator> <id>)

<operator>    ::=   and  |
                    or   |
                    “=”  |
                    “<>” |
                    “>”  |
                    “<”  |
                    “>=” |
                    “<=”

<list>       ::=    “(“ <id>+ “)”

<comment>     ::=   (“//” <character>* eoln) |
                     “/*” <character>* “*/”

<key_word>    ::=   attr  |
                    eval |
                    rule |
                    file  |
                    fired

<id>         ::=    <a-z | A-Z | 0-9 | “_”>+

<character>   ::=   <any_ascii_character_except_eoln>
```

FIG. 6B

```
<SEL_rule_set> ::=  <statement>*

<statement>    ::=  (<key_word> <id> <block>) |
                    <comment>

<block>        ::=  “[“ <phrase>+ “]” |
                    <list>

<phrase>       ::=  (<id> <list>) |
                    (<antecedent> <consequence>)

<antecedent>   ::=  if <expression>+

<consequence>  ::=  then <expression>+

<expression>   ::=  (<id> <operator> <id>)

<operator>     ::=  and |
                    or  |
                    eq  |
                    ne  |
                    gt  |
                    lt  |
                    ge  |
                    le

<list>         ::=  “(“ <id>+ “)”

<comment>      ::=  (“//” <character>* eoln) |
                    “/*” <character>* “*/”

<key_word>     ::=  variable      |
                    order_of_eval |
                    business_rule |
                    persistance   |
                    fired

<id>           ::=  <a-z | A-Z | 0-9 | “_”>+

<character>    ::=  <any_ascii_character_except_eoln>
```

FIG. 7A

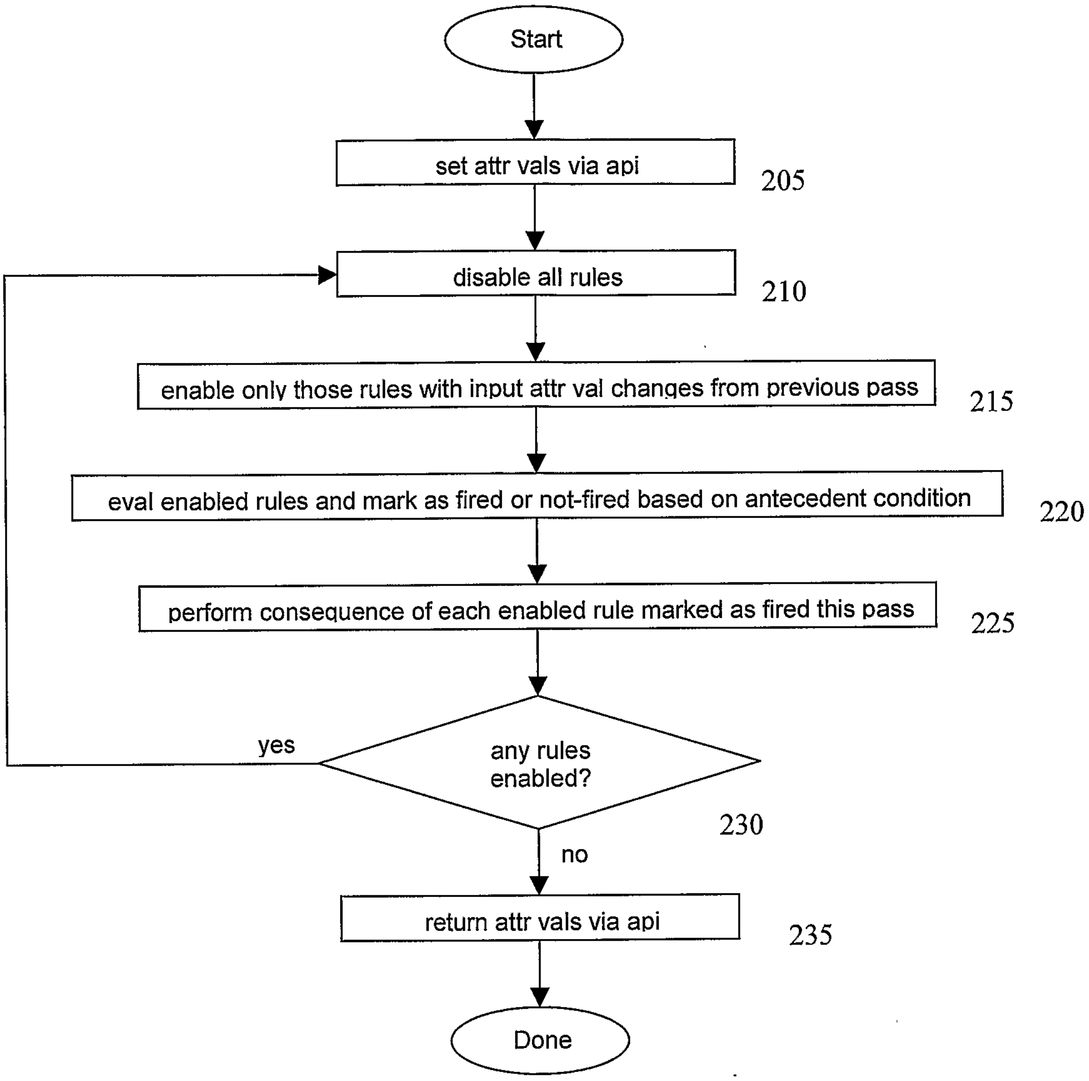

Start
set attr vals via api
205
disable all rules
210
enable only those rules with input attr val changes from previous pass
215
eval enabled rules and mark as fired or not-fired based on antecedent condition
220
perform consequence of each enabled rule marked as fired this pass
225
any rules enabled?
yes
no
230
return attr vals via api
235
Done

FIG. 7B

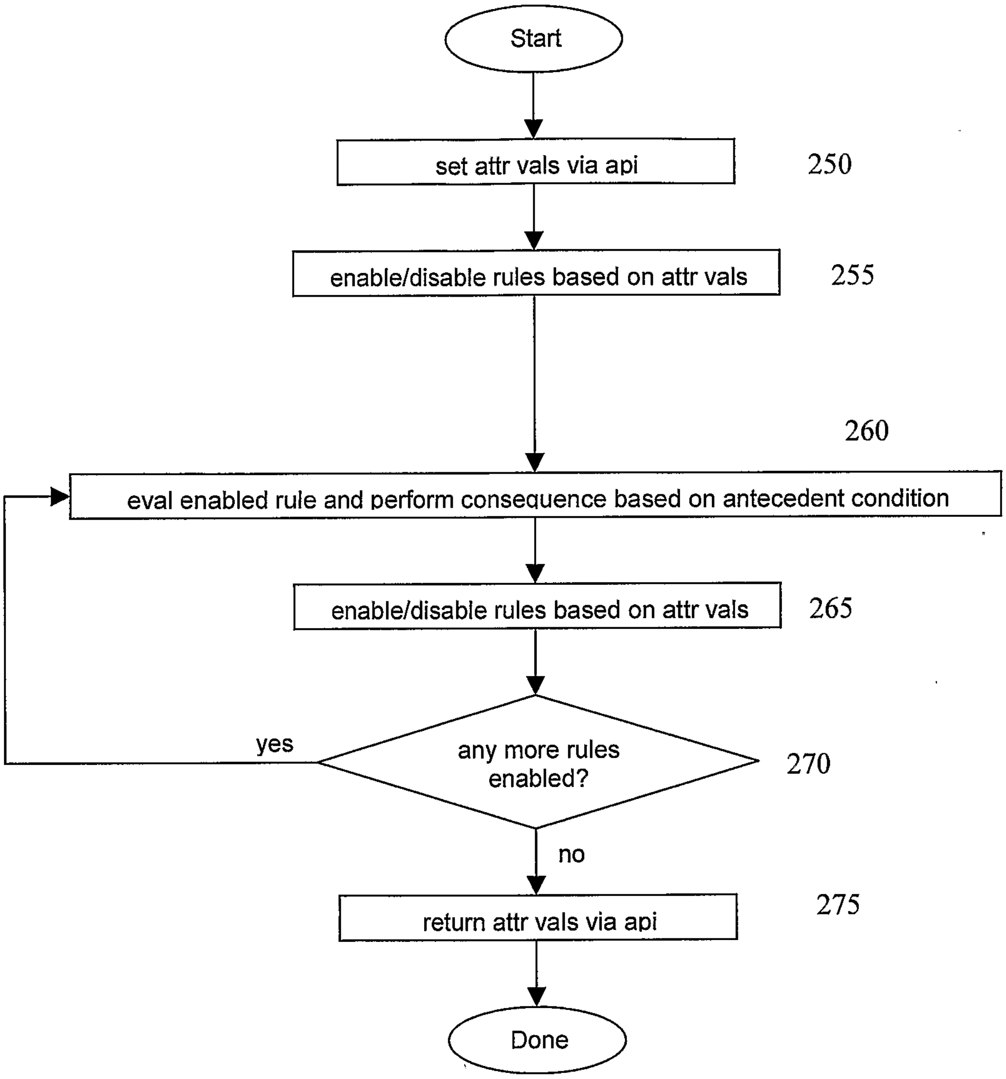

Start
set attr vals via api
250
enable/disable rules based on attr vals
255
260
eval enabled rule and perform consequence based on antecedent condition
enable/disable rules based on attr vals
265
yes
any more rules enabled?
270
no
return attr vals via api
275
Done

FIG. 7C

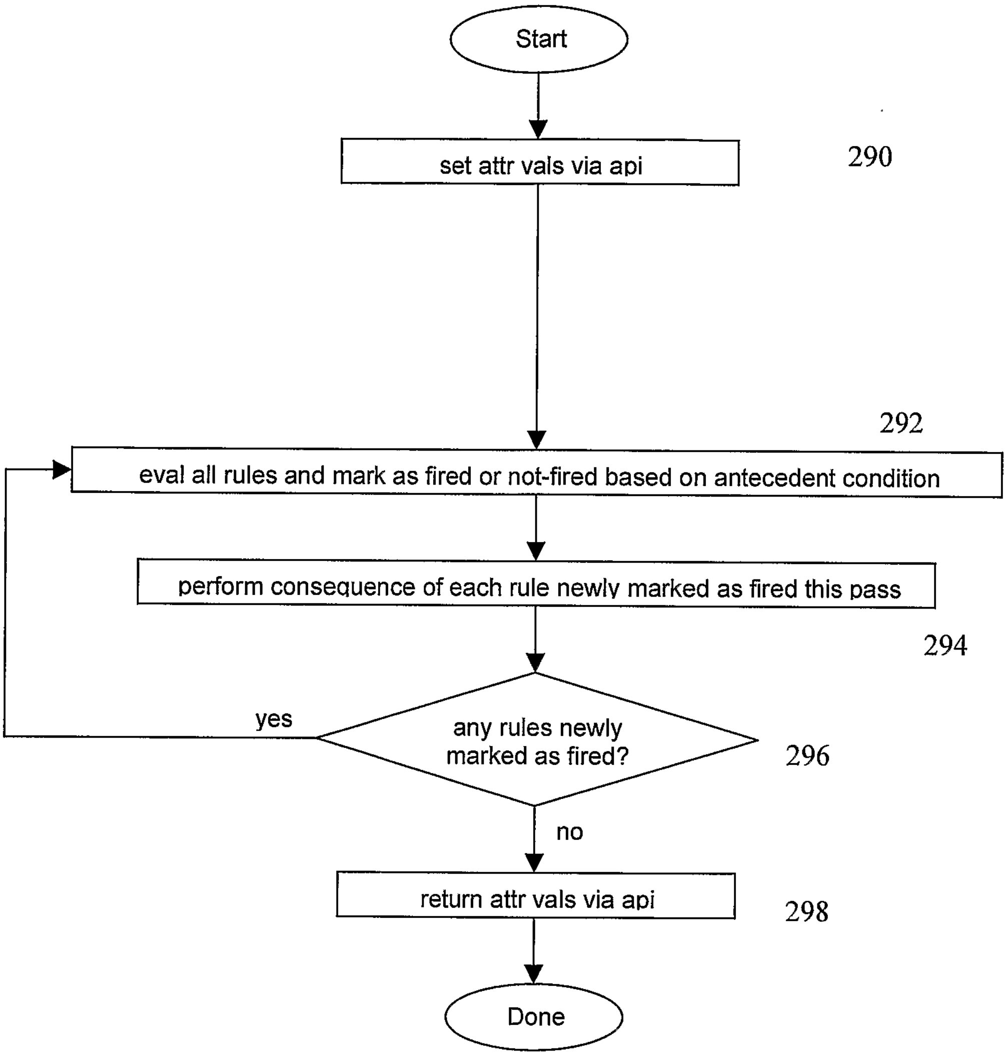

Start
set attr vals via api
290
292
eval all rules and mark as fired or not-fired based on antecedent condition
perform consequence of each rule newly marked as fired this pass
294
yes
any rules newly marked as fired?
296
no
return attr vals via api
298
Done

FIG. 7D

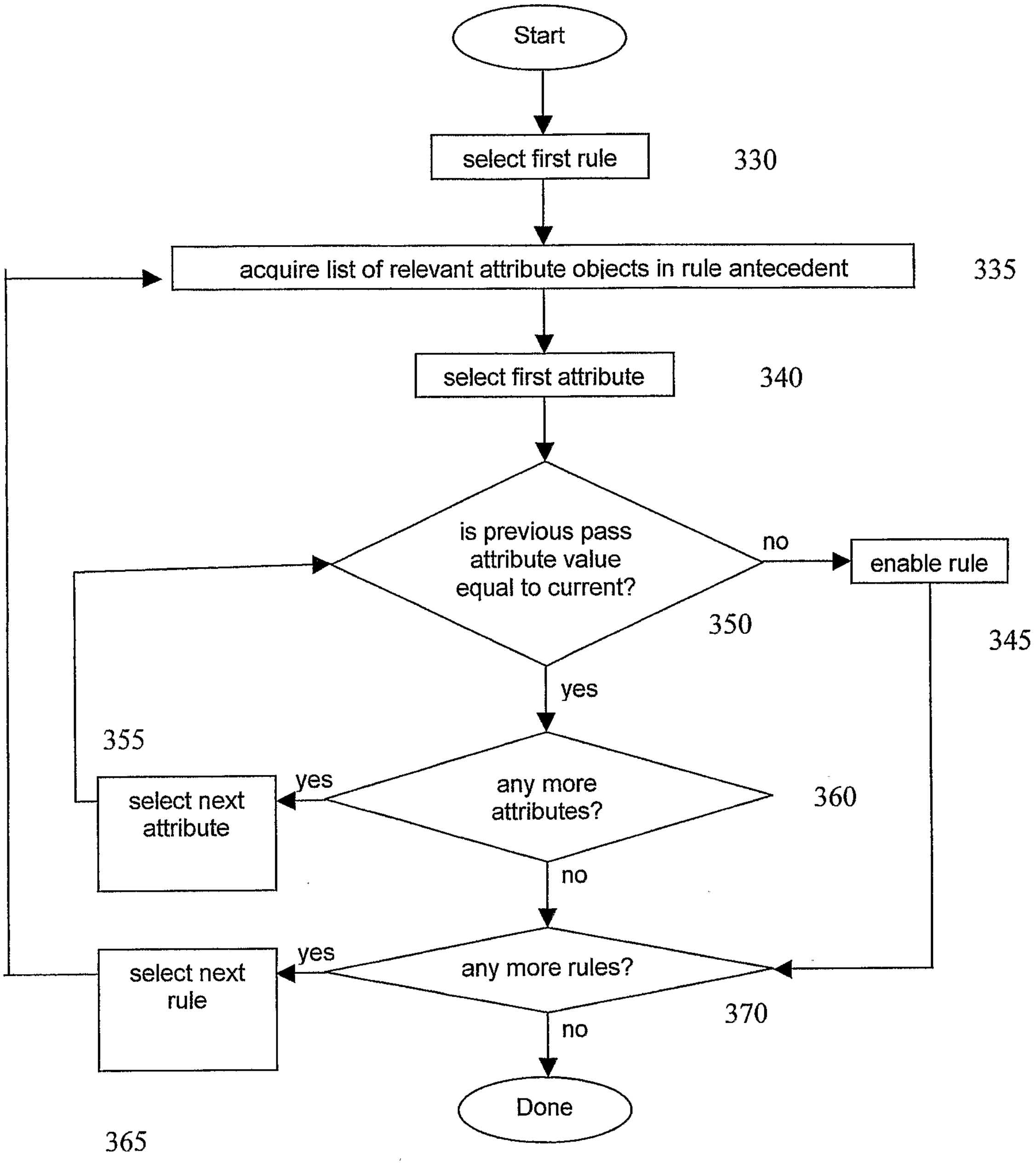

Start
select first rule
330
acquire list of relevant attribute objects in rule antecedent
335
select first attribute
340
is previous pass attribute value equal to current?
no
enable rule
350
345
yes
355
select next attribute
yes
any more attributes?
360
no
select next rule
yes
any more rules?
370
no
Done
365

FIG. 7E

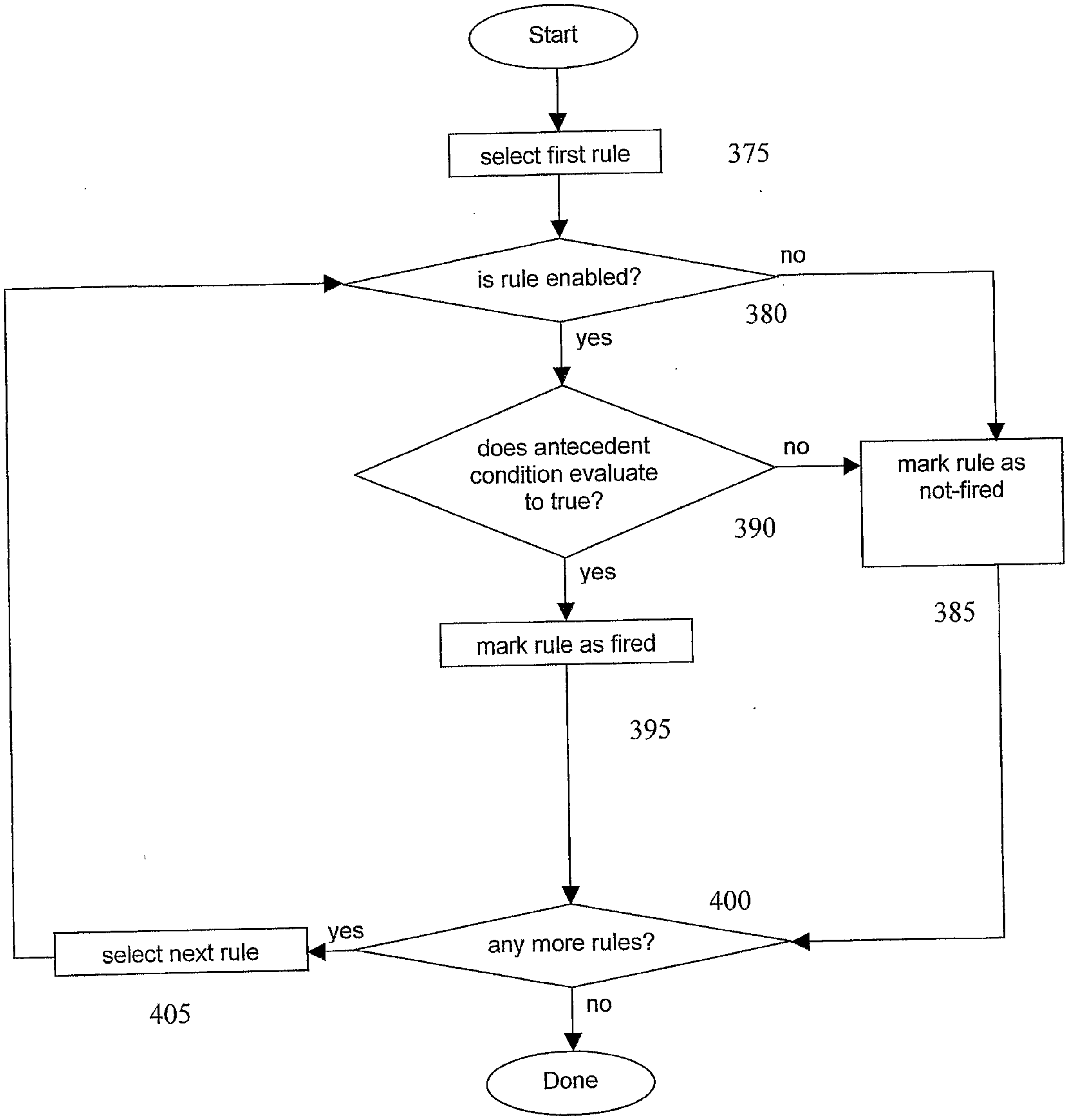

Start
select first rule
375
is rule enabled?
no
380
yes
does antecedent condition evaluate to true?
no
mark rule as not-fired
390
yes
385
mark rule as fired
395
400
yes
select next rule
any more rules?
405
no
Done

FIG. 7F

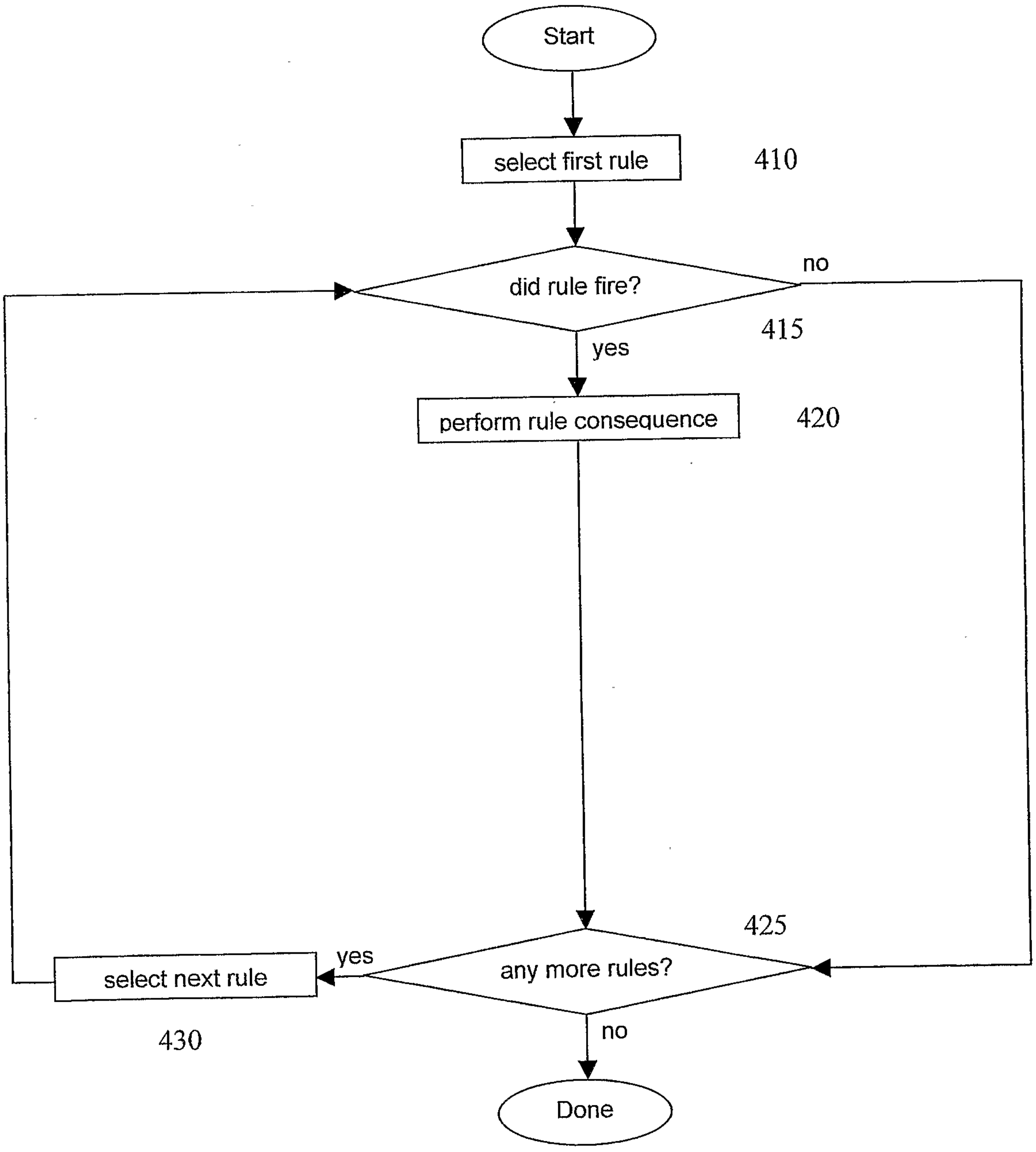

Start
select first rule
410
did rule fire?
no
415
yes
perform rule consequence
420
425
any more rules?
yes
select next rule
430
no
Done

FIG. 7G

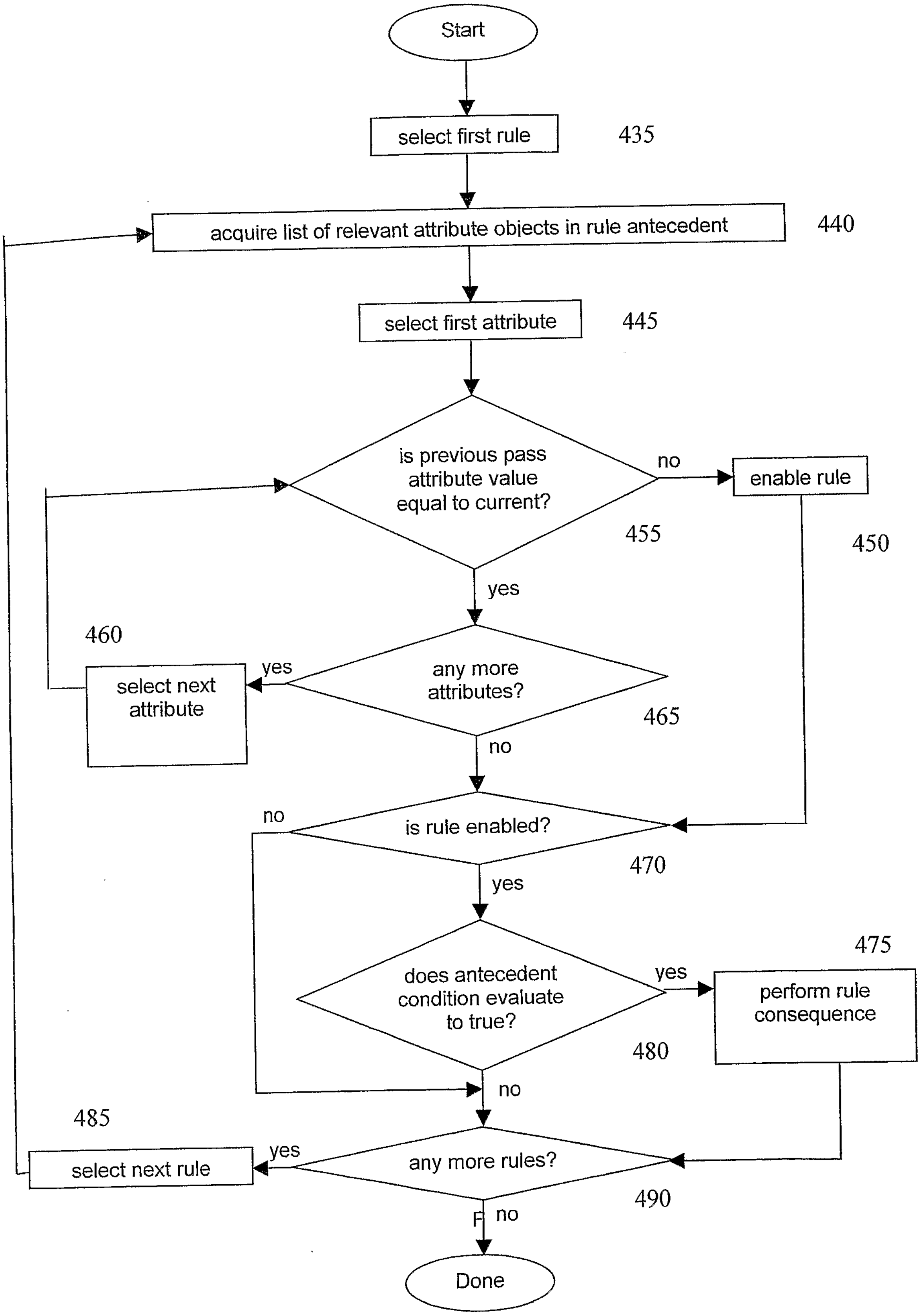

Start
select first rule
435
acquire list of relevant attribute objects in rule antecedent
440
select first attribute
445
is previous pass attribute value equal to current?
no
enable rule
455
450
yes
460
yes
select next attribute
any more attributes?
465
no
no
is rule enabled?
470
yes
475
does antecedent condition evaluate to true?
yes
perform rule consequence
480
no
485
yes
select next rule
any more rules?
490
F
no
Done

FIG. 7H

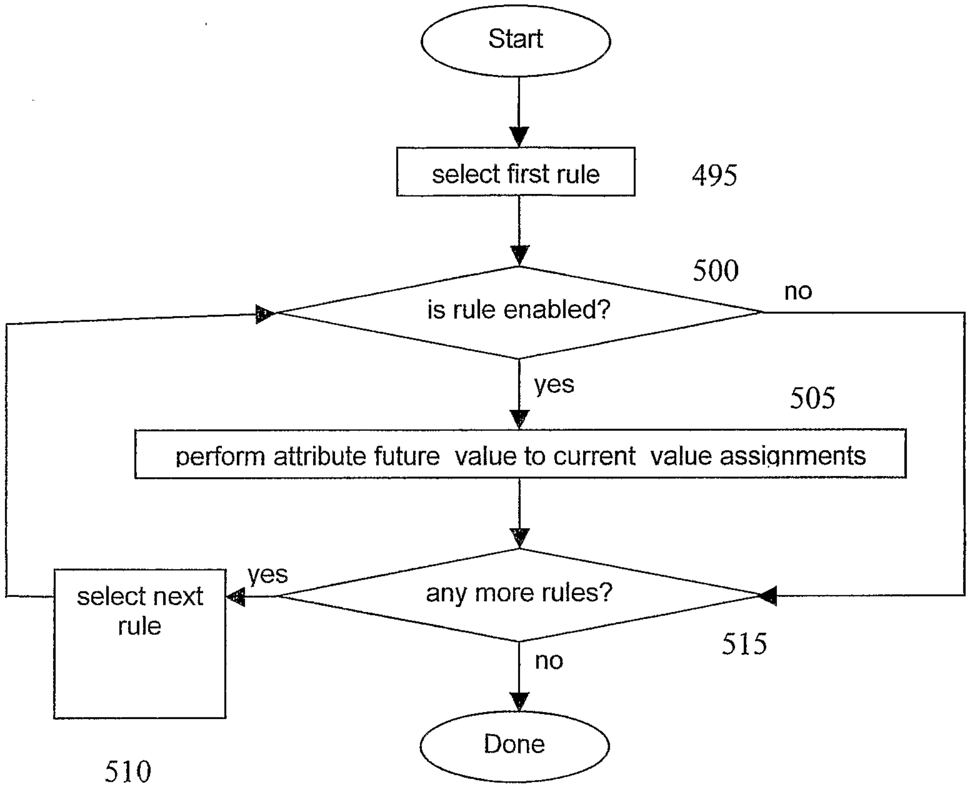

Start
select first rule
495
500
is rule enabled?
no
yes
505
perform attribute future value to current value assignments
yes
select next rule
any more rules?
515
no
Done
510

FIG. 8 Example 1

I received printer by mail.

I unpacked printer.

I noticed loose plastic parts inside printer.

I printed first test page fine.

I printed second test page, which caused paper jam.

I diagnosed problem with printer as page eject levers broken off.

I decided that I needed a replacement printer.

I checked vendor warranty coverage

I found vendor CS phone number in printer doc.

I called vendor.

VRU answered.

VRU asked, if English speaking to stay online or hit 1 for Spanish.

I waited.

VRU asked for product name.

I answered with product name.

VRU asked if I owned product less than one year.

I answered yes.

VRU said to please wait.

I waited.

Human answered.

Human asked for phone number.

I answered with phone number.

Human asked for last name.

I answered with last name.

Human asked for first name.

I answered with first name.

Human asked if I wanted to receive telemarketing calls, spam, and/or junk mail.

I answered no.

Human asked for product name.

I answered with product name of printer.

Human asked for product serial number.

I answered with product serial number.

Human asked for problem description.

I answered that the page eject levers had broken off and I described my approach to diagnosing the problem.

Human agreed that printer is damaged and covered under warranty.

Human checked on resolution options.

Human stated four resolution option choices: return then receive paying cost of shipping, receive then return with credit card collateral for free, 2 day shipping for more money, one day shipping for even more money.

I chose second option to receive then return with credit card collateral for free.

Human asked for shipping address.

I answered with shipping address.

Human asked for credit card number.

I answered with credit card number.

Human requested credit card authorization.

Human stated that credit card authorization was successful.

Human gave me customer service order number.

I confirmed by reading back customer service order number.

Human gave me case ID.

I confirmed by reading back case ID.

Human asked if I had any other questions.

I answered no.

Human said thank you for calling CS.

I said thank you.

We ended call.

FIG. 9

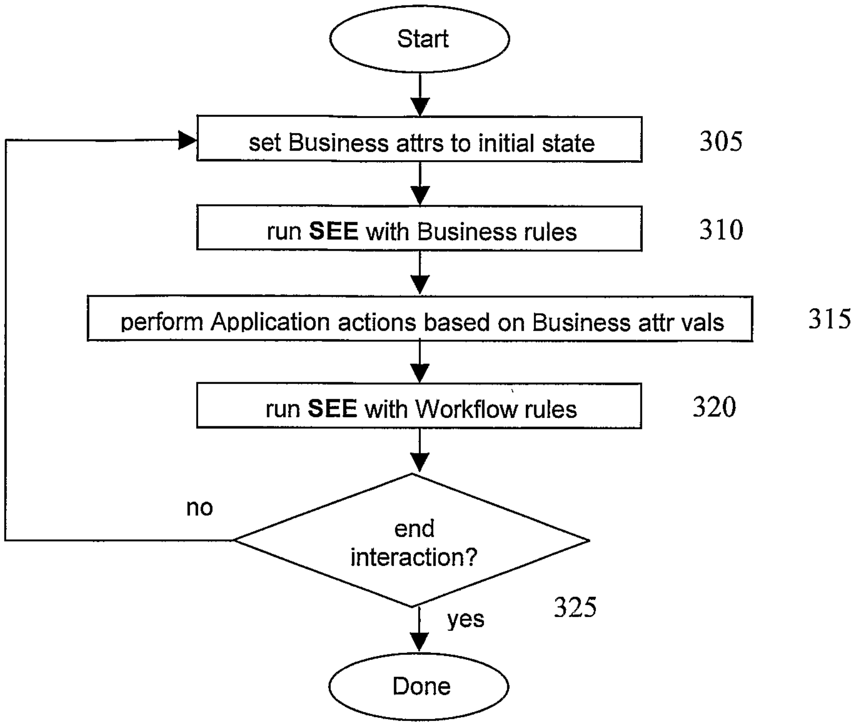

Start
set Business attrs to initial state
305
run **SEE** with Business rules
310
perform Application actions based on Business attr vals
315
run **SEE** with Workflow rules
320
no
end interaction?
325
yes
Done

FIG. 10

```
// Work Flow State Machine Example

attr state      (vru human exit)
attr next       (vru human exit)
attr action     (get_vru_resp involve_human end_interaction)

rule next_state_vru {
        if      state   = vru and
                action  = get_vru_resp
        then    next    = vru
}
rule next_state_human {
        if      state   = vru and
                action  = involve_human
        then    next    = human
}
rule next_state_exit {
        if      (state  = vru or
                 state  = human) and
                action  = end_interaction
        then    next    = exit
}
```

FIG. 11 Example 1

```
// SEL: Example 1 - Customer Service Interaction
/* Example 1 Subset:
I called vendor.
VRU answered.
VRU asked, if English speaking to stay online or
hit 1 for Spanish.
I waited.
VRU asked for product name.
I answered with product name.
VRU asked if I owned product less than one year.
I answered yes.
VRU said to please wait.
I waited.
Human answered. */

attr lang_spoken {
        ask (    If you speak Spanish, please
press 1.
                 If you speak English, please stay
        on the line.)
        val (1 null)
}
attr lang_use (english spanish)
attr prod_name {
        ask (What is the name of the product that
you purchased?)
        val (printer null)
}
attr human (yes no)
```

```
attr purch_under_year_ago {
        ask (Did you purchase the product under
one year ago?)
        val (yes no)
}
attr support_free (yes no)

eval attr {
        1 (lang_spoken)
        2 (prod_name)
        3 (purch_under_year_ago)
}

rule lang_use_set_english {
        if lang_spoken = null
        then lang_use  = english
}
rule lang_use_set_spanish {
        if      lang_spoken = 1
        then    lang_use   = spanish
}
rule human_involvement_determination {
        if      prod_name = printer
        then    human     = yes
}
rule support_fee_determination {
        if      purch_under_year_ago = yes
        then    support_free        = yes
}
```

FIG. 12

FILE: "LanguageRules.sel"

```
attr lang_spoken {
          ask (     If you speak Spanish, please
press 1.
                    If you speak English, please stay
          on the line.)
          val (1 null)
}
attr lang_use (english spanish)
rule lang_use_set_english {
          if lang_spoken = null
          then lang_use  = english
}
rule lang_use_set_spanish {
          if        lang_spoken = 1
          then      lang_use    = spanish
}
```

FILE: "Fee.sel"

```
attr purch_under_year_ago {
          ask (Did you purchase the product under
one year ago?)
          val (yes no)
}
attr support_free (yes no)
rule support_fee_determination {
          if        purch_under_year_ago = yes
          then      support_free         = yes
}
```

FILE: "CustomerServiceInteraction.sel"

```
//this command includes files "LanguageRules.sel"
and "Fee.sel" inline
eval file (LanguageRules Fee)

attr prod_name {
          ask (What is the name of the product that
you purchased?)
          val (printer null)
}
attr human (yes no)

eval attr {
          1 (lang_spoken)
          2 (prod_name)
          3 (purch_under_year_ago)
}

rule human_involvement_determination {
          if        prod_name             = printer
and
                    (support_fee_determination =
fired   or  //reference to rule id
                     suport_free          = yes)
//attribute set prior
          then      human = yes
}
```

FIG. 13 Example 2 SEL

```
attr name               (fred joe bob tom)
attr position           (1 2 3 4)
attr color             (red blue plaid orange)
attr position_plus_one_color (red blue plaid orange)
attr joe_position        (true false)
attr bob_color           (true false)
attr tom_position_color     (true false)
attr name_position_color     (true false)

rule fred_position_plus_one_color_correct {
        if      name                    = fred  and
                position_plus_one_color     = blue
        then    fred_position_plus_one_color = true
}
rule joe_position_correct {
        if      name      = joe  and
                position    = 2
        then    joe_position = true
}
rule bob_color_correct {
        if      name     = bob   and
                color     = plaid
        then    bob_color = true
}
rule tom_position_color_correct {
        if      name          =  tom    and
                (position        <> 1      and
         position         <> 4)    and
        color           <> orange
        then    tom_position_color =  true
}
rule name_position_color_correct {
        if      fred_position_plus_one_color = true and
        joe_position          = true and
                bob_color              = true and
        tom_position_color        = true
        then    name_position_color        = true
}
```

FIG. 14 Example 2 JESS

```
(deftemplate pants-color (slot of) (slot is))
(deftemplate position (slot of) (slot is))
(defrule find-solution
        ;rule 1 – Fred’s position plus one color
        (position (of Fred) (is ?p1))
        (pants-color (of Fred) (is ?c1))
        (position (of ?n&~Fred)
                (is ?p&: (eq ?p (+ ?p1 1))))
        (pants-color (of ?n&~Fred)
                (is blue&~?c1))
        ;rule 2 - Joe’s position
        (position (of Joe) (is ?p2&2&2~?p1))
        (pants-color (of Joe) (is ?c2&~?c1))
        ;rule 3 – Bob’s pants color
        (position (of Bob)
                (is ?p3&~?p1&~?p&~?p2))
        (pants-color (of Bob&~?n)
                (is plaid&?c3&~?c1&~?c2))
        ;rule 4 – Tom’s position and pants color
        (position (of Tom&~?n)
                (is ?p4&~1&~4&~?p1&~?p2&~?p3))
        (pants-color (of Tom)
                (is ?c4&~orange&~blue&~?c1&~?c2&~?c3))
        =>
        (printout t Fred “ “ ?p1 “ “ ?c1 crlf)
        (printout t Joe  “ “ ?p2 “ “ ?c2 crlf)
        (printout t Bob  “ “ ?p3 “ “ ?c3 crlf)
        (printout t Tom  “ “ ?p4 “ “ ?c4 crlf crlf))
```

FIG. 15A

```
object_type    attribute
     string_type    name
     object_type    previous_value
     object_type    current_value
```

FIG. 15B

```
object_type    attribute
     string_type    name
     object_type    previous_value
     object_type    current_value
     object_type    future_calue
```

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US04/28624

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) : G06N 5/02; G06F, 15/18; G06N, 5/04

US CL : 706/47, 14, 60

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 706/47, 14, 60

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

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

USPTO East, US-PGPUB, USPAT, EPO, JPO, DEWENT, IBM, IEEE

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US PUB 2002/0120917 A1 (Abrari et al) 29 August 2002 (29.08.2002), Abstract, Figures 2, 6, Paragraphs 0017, 0031, 0040, 0048, 0050.	1-3
X	US 5,485,544 A (Nonaka et al) 16 January 1996 (16.01.1996), Abstract, column 7, lines 11-14, column 7, lines 60-63, column 8, lines 49-54.	1-3

□ Further documents are listed in the continuation of Box C. □ See patent family annex.

*	Special categories of cited documents:
"A"	document defining the general state of the art which is not considered to be of particular relevance
"E"	earlier application or patent published on or after the international filing date
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O"	document referring to an oral disclosure, use, exhibition or other means
"P"	document published prior to the international filing date but later than the priority date claimed
"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&"	document member of the same patent family

Date of the actual completion of the international search

06 January 2005 (06.01.2005)

Date of mailing of the international search report

13 JAN 2005

Name and mailing address of the ISA/US

Mail Stop PCT, Attn: ISA/US
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450

Facsimile No. (703) 305-3230

Authorized officer for Michelle

Joseph P. Hirl

Telephone No. 571-272-3685