A decision management system for managing and executing interpreted language code comprising:
- a parser for parsing controlled language code against both a language grammar and for building an execution model comprising parsed known concepts;
- a pattern matching engine for matching patterns in the controlled language code with a patterns for a new concept;
and
- a concept engine for creating a new concept from the matched pattern and controlled language code and for saving the new concept into a object model and into a language grammar so that current and future parsing errors will not occur when recognizing the new concept.

Figure 4

Extended Decision Management Process 400

401 DSL formed from Syntax & Vocabulary

402 Parser recieves statements

403 Parser parses statements

404 Errors?

405 Match discovery patterns

406 Re-analyze with discovery patterns

411 Infer type of vocabulary element

412 Add to vocabulary

413 Regenerate DSL

409 Create vocabulary element

410 Typed?

407 Pattern matched?

408 Error

Figure 4
Document frame 504

Definitions

- set ‘minimum income’ to 0.37 * the yearly income of ‘the borrower’;

If

- the yearly repayment of ‘the loan’ is at least ‘minimum income’

Then

- in ‘the loan report’, refuse the loan with the message “Too big Debt/Income ratio:” + the yearly repayment of ‘the loan’ / the yearly income of ‘the borrower’;

<table>
<thead>
<tr>
<th>Concept</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Underlined</td>
</tr>
<tr>
<td>Known Concepts</td>
<td>‘quotes’</td>
</tr>
<tr>
<td>New Concepts</td>
<td>Double underlined</td>
</tr>
</tbody>
</table>
ENTERPRISE DECISION MANAGEMENT

FIELD OF THE INVENTION

This invention relates to a method and apparatus for enterprise data management and in particular enterprise decision management.

BACKGROUND

The general environment of decision management comprises decision making tools and change detection tools for providing a rules management system that is easy to evolve, trace, audit, and test.

In decision management environments, users capture business statements using a controlled natural language (CNL). For instance, they capture business rules, business event processing rules or queries, using dedicated business-centric domain specific languages (DSL) that combine ease of use of natural language and the level of formalism required to capture statements that need to be unambiguously executed by a machine.

A controlled natural language is a subset of natural language obtained by restricting the grammar and the vocabulary used, in order to reduce ambiguity and complexity. A system can guide a user through type-ahead suggestions consistent with a CNL. Terms and phrases defined by the user to represent a business domain are part of a CNL. A set of terms and phrases manipulated in a CNL is known as a business vocabulary. Although a business vocabulary is usually defined by the business users who also use business-centric controlled languages, frequently a user would like to use some terminology that does not exist in this vocabulary. Further information about a business vocabulary can be found in the Business Vocabulary and Rules (SBVR) standard as defined by the Object Management Group (OMG), a not-for-profit computer industry standards consortium.

A typical example is the case where a user needs to set-up a new promotion rule for an e-commerce business and needs to refer to some concept property not yet exposed in the business vocabulary. The user may also want to refer to a specific concept that does not exist.
This new element of terminology could represent a completely new piece of data or a new computation of some existing elements. In both cases, the vocabulary needs to be updated to propose the new terminology and allow its usage in the controlled language.

In existing systems, business statement authoring tools are usually decoupled and provided in separate environments, sometimes addressing different kind of users. As a consequence, while capturing a business statement using a controlled language, the user has to go to the business vocabulary authoring environment to add the terminology element he would like to manipulate, before coming back to the statement authoring environment to continue editing his statement. This induces a significant context switch which has a negative impact on the user productivity and increases the risk of introducing errors.

Another issue is that the formalism to capture business vocabulary is usually different from the one used to capture business statements with a controlled language. Techniques and tools used to author a business vocabulary usually relies on dedicated form-based or graphical user interfaces (GUI) similar to the one used to capture object models in modern modeling software.

As a consequence, the user has to switch from a text-based authoring paradigm that characterize tools using controlled languages to a more graphical dedicated user interface, increasing the context switch effect described before.

BRIEF SUMMARY OF THE INVENTION

In a first aspect of the invention there is provided a system for managing and executing interpreted language code comprising: a parser for parsing controlled language code against both a language grammar and for building an execution model comprising parsed known concepts; a pattern matching engine for matching patterns in the controlled language code with a patterns for a new concept; and a concept engine for creating a new concept from the matched pattern and controlled language code and for saving the new concept into a object model and into a language grammar so that current and future parsing errors will not occur when recognizing the new concept.

The concepts are not limited to the specific type of business rules concepts but any type of
concept including relationships and attributes of concepts.

The embodiments propose creation of a new business vocabulary element while a user edits a business statement using a controlled language and to support the user editing using the new business vocabulary element.

The embodiments propose a text editor that captures a business statement compliant with a given controlled language and that augments the business vocabulary on which the controlled language is based, when the user input text fragments that do not match the grammar but that match specific patterns described with the grammar.

For instance, a user creates the following business rule using a business action language:

If the number of clients where the income of each client is more than $100,000, is at least 5 then notify the supervisor.

The underlined terms “client” and “income” were foreign to, or not part of, the business vocabulary when the statement was written and the embodiments identify the foreign terms and augment the business vocabulary with the foreign terms. Using patterns of where new concepts might be expected to be found in statements, the embodiments make the assumption that a “client” is a concept and an “income” is a property for the “client” concept. The income property is also recognized as being of currency type dollar.

Instead of having a business rule in error, the rule would be correctly interpreted by the system and the increment to the existing vocabulary would be clearly identified. The embodiments changes prior art execution flow process that would have resulted in a error message whereby the embodiments handle some patterns of unknown text and allow a execution flow to continue. The embodiments operate at the machine level of the computer on program code statements regardless of what the program code statement are for. The embodiments comprises a computer that operates in a new way to handle unknown concepts in program code whereas a prior art computer would not so handle. The embodiments increase the reliability of the computer in handling program code.

In the preferred embodiment a discovery pattern mechanism is the pattern matching engine
and the concept engine and a decision execution server is the execution engine.

Advantageously an error during parsing of the language grammar triggers a search to match the error with a new concept.

Still more advantageous the interpreted language code comprises rules and actions for making decisions and the system is a decision management system.

Preferably the system further comprises matching patterns in the controlled language against patterns for types of new concepts.

More preferably the pattern matching engine is for matching patterns in the controlled language against patterns for new properties of concepts; and the concept engine is for creating new properties of a concept from the matched pattern and the controlled language code and saving the property of a concept with a corresponding action into the object model; and wherein execution engine is for executing the concepts and actions in the object model.

Still more preferably the pattern matching engine is for matching patterns in the controlled language against patterns for types of new properties.

Yet more preferably the system further comprises an execution engine for executing the concepts of the object model.

In a second aspect of the invention is provided a method for managing and executing interpreted language code comprising: parsing controlled language code against a language grammar to build an execution model comprising known concepts; matching patterns in the controlled language against patterns for new concepts; creating a new concept from the matched pattern and the controlled language code; and saving the new concept into a object model.

In a third aspect of the invention there is provided a computer program product for managing and executing interpreted language code, the computer program product comprising a computer-readable storage medium having computer-readable program code embodied therewith and the computer-readable program code configured to perform the method of any
one of claims 8 to 14.

The computer program product comprises a series of computer-readable instructions either fixed on a tangible medium, such as a computer readable medium, for example, optical disk, magnetic disk, solid-state drive or transmittable to a computer system, using a modem or other interface device, over either a tangible medium, including but not limited to optical or analogue communications lines, or intangibly using wireless techniques, including but not limited to microwave, infrared or other transmission techniques. The series of computer readable instructions embodies all or part of the functionality previously described herein.

Those skilled in the art will appreciate that such computer readable instructions can be written in a number of programming languages for use with many computer architectures or operating systems. Further, such instructions may be stored using any memory technology, present or future, including but not limited to, semiconductor, magnetic, or optical, or transmitted using any communications technology, present or future, including but not limited to optical, infrared, or microwave. It is contemplated that such a computer program product may be distributed as a removable medium with accompanying printed or electronic documentation, for example, shrink-wrapped software, pre-loaded with a computer system, for example, on a system ROM or fixed disk, or distributed from a server or electronic bulletin board over a network, for example, the Internet or World Wide Web.

In a fourth aspect of the invention there is provided a computer program stored on a computer readable medium and loadable into the internal memory of a digital computer, comprising software code portions, when said program is run on a computer, for performing the method of any of claims 8 to 14.

In a fifth aspect of the invention there is provided a data carrier aspect of the preferred embodiment that comprises functional computer data structures to, when loaded into a computer system and operated upon thereby, enable said computer system to perform all the steps of the method. A suitable data-carrier could be a solid-state memory, magnetic drive or optical disk. Channels for the transmission of data may likewise comprise storage media of all descriptions as well as signal-carrying media, such as wired or wireless signal-carrying media.
BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the present invention will now be described, by way of example only, with reference to the following drawings in which:

Figure 1 is decision management system in which a decision management server of a preferred embodiment is deployed;
Figure 2 is a component diagram of a computing system for executing a decision management server of a preferred embodiment;
Figure 3A is a component diagram of a prior art decision management server;
Figure 3B is a component diagram of a decision management server according to a preferred embodiment;
Figure 4 is a flow diagram of a method of the decision management server of a preferred embodiment;
Figure 5 is an example of the language code processed by a preferred embodiment; and
Figure 6 is a component diagram of a parallel computing system for executing a decision management system according to a parallel computing embodiment.

DETAILED DESCRIPTION OF THE EMBODIMENTS

Referring to Figure 1, a decision management system 2 includes: a decision execution server 3; a decision design server 4; and a decision management server 5.

Decision execution server 3 is for managing decisions and event detection separately from applications. Decision server 3 provides the runtime and development components to automate the response of highly variable decisions based on the specific context of a process, transaction, or interaction. A business network can be monitored to discover and take action on event-based data patterns, and then process this information against hundreds or even thousands of business rules in order to determine how to respond within front-end and back-end systems.

Decision design server 4 is for the rule development before making the rules available to the decision server and the execution server.
Decision management server 5 is for putting decision management in the hands of those who drive the business. Business users can manage decisions and events directly based on organizational knowledge and best practices, with limited dependence on the information technology department. The degree of dependence can range from a limited review by business users of the business logic implemented by developers, to complete control over the specification, creation, testing, and deployment of the business logic by business users. Business and information technology functions can work in collaboration, aligning the entire organization in the implementation of automated decisions and accelerating the maintenance life cycle as they evolve based on new external and internal requirements.

Referring to Figure 2, a deployment of a preferred embodiment in computer processing system 10 is described. Computer processing system 10 is operational with numerous other general purpose or special purpose computing system environments or configurations. Examples of well-known computing processing systems, environments, and/or configurations that may be suitable for use with computer processing system 10 include, but are not limited to, personal computer systems, server computer systems, thin clients, thick clients, hand-held or laptop devices, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronics, network PCs, minicomputer systems, mainframe computer systems, and distributed cloud computing environments that include any of the above systems or devices.

Computer processing system 10 may be described in the general context of computer system-executable instructions, such as program modules, being executed by a computer processor. Generally, program modules may include routines, programs, objects, components, logic, and data structures that perform particular tasks or implement particular abstract data types.

Computer processing system 10 may be embodied in distributed cloud computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed cloud computing environment, program modules may be located in both local and remote computer system storage media including memory storage devices.

Computer processing system 10 comprises: general-purpose computer server 12 and one or more input devices 14 and output devices 16 directly attached to the computer server 12. Computer processing system 10 is connected to a network 20. Computer processing system
10 communicates with a user 18 using input devices 14 and output devices 16. Input devices 14 include one or more of: a keyboard, a scanner, a mouse, trackball or another pointing device. Output devices 16 include one or more of a display or a printer. Computer processing system 10 communicates with network devices (not shown) over network 20. Network 20 can be a local area network (LAN), a wide area network (WAN), or the Internet.

Computer server 12 comprises: central processing unit (CPU) 22; network adapter 24; device adapter 26; bus 28 and memory 30.

CPU 22 loads machine instructions from memory 30 and performs machine operations in response to the instructions. Such machine operations include: increment or decrement a value in register (not shown); transfer a value from memory 30 to a register or vice versa; take instructions from a different location in memory if a condition is true or false (also known as a conditional branch instruction); and add or subtract the values in two different registers and put the result in another register. A typical CPU can perform many different machine operations. A set of machine instructions is called a machine code program, the machine instructions are written in a machine code language which is referred to a low level language. A computer program written in a high level language needs to be compiled to a machine code program before it can be run. Alternatively a machine code program such as a virtual machine or an interpreter can interpret a high level language in terms of machine operations.

Network adapter 24 is connected to bus 28 and network 20 for enabling communication between the computer server 12 and network devices.

Device adapter 26 is connected to bus 28 and input devices 14 and output devices 16 for enabling communication between computer server 20 and input devices 14 and output devices 16.

Bus 28 couples the main system components together including memory 30 to CPU 22. Bus 28 represents one or more of any of several types of bus structures, including a memory bus or memory controller, a peripheral bus, an accelerated graphics port, and a processor or local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures include Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA)
local bus, and Peripheral Component Interconnects (PCI) bus.

Memory 30 includes computer system readable media in the form of volatile memory 32 and non-volatile or persistent memory 34. Examples of volatile memory 32 are random access memory (RAM) 36 and cache memory 38. Generally volatile memory is used because it is faster and generally non-volatile memory is used because it will hold the data for longer. Computer processing system 10 may further include other removable and/or non-removable, volatile and/or non-volatile computer system storage media. By way of example only, persistent memory 34 can be provided for reading from and writing to a non-removable, non-volatile magnetic media (not shown and typically a magnetic hard disk or solid-state drive). Although not shown, further storage media may be provided including: an external port for removable, non-volatile solid-state memory; and an optical disk drive for reading from or writing to a removable, non-volatile optical disk such as a compact disk (CD), digital video disk (DVD) or Blu-ray. In such instances, each can be connected to bus 28 by one or more data media interfaces. As will be further depicted and described below, memory 30 may include at least one program product having a set (for example, at least one) of program modules that are configured to carry out the functions of embodiments of the invention.

In the preferred embodiment, the set of program modules configured to carry out the functions of the embodiments comprises extended decision management module 5’’ for enabling computer server 10 to perform as decision management server 5. Further program modules that support the preferred embodiment but are not shown including firmware, boot strap program, operating system, and support applications. Each of the operating system, support applications, other program modules, and program data or some combination thereof, may include an implementation of a networking environment. A prior art computer server would have prior art decision management module 5’ instead of extended decision management module 5’’.

Similar modules not shown, for instance, a decision execution module 3’ and a decision design module 4’ would enable another computer server (like computer server 10) to perform as decision execution server 3 and decision design server 4 respectively. Furthermore all three modules could execute on the same computer server as in a parallel computing embodiment described later.
Computer processing system 10 communicates with at least one network 20 (such as a local area network (LAN), a general wide area network (WAN), and/or a public network like the Internet) via network adapter 24. Network adapter 24 communicates with the other components of computer server 12 via bus 28. It should be understood that although not shown, other hardware and/or software components could be used in conjunction with computer processing system 10. Examples include, but are not limited to: microcode, device drivers, redundant processing units, external disk drive arrays, redundant array of independent disks (RAID), tape drives, and data archival storage systems.

Referring to Figure 3A, prior art decision management module 5’ comprises: domain specific language (DSL) syntax 302; business vocabulary 304; domain specific language (DSL) processor 306; domain context-free grammar 308; parser 310; abstract syntax tree 312; editor 314; and semantic actions 316.

Domain specific language (DSL) syntax 302 comprises syntax terms for a domain specific language that enables the parser to check for syntax errors in statements. DSL syntax can be specific to one language (for example syntax for a specific business rules language) or to a type of language (for example a generic business rules language syntax). DSL syntax 302 and business vocabulary 304 are abstract parts of domain context-free grammar 308. DSL processor 306 builds domain context-free grammar 308 from DSL syntax 302 and business vocabulary 308.

Business vocabulary 304 is a set of concepts and the relationships between the concepts. Natural language terms are associated to these concepts to designate them so that a human can reference it. “Car” is, for instance, the English term used to designate the concept of a car. Synonyms terms can be used to designate the same concept. Phrases can be associated to relationships between concepts to “manipulate” these relationships. For instance, a car concept could be linked to a person concept and “a person drives a car” is an English sentence that references this relationship. Furthermore, “a car is driven by a person” is another sentence associated to the same relationship. In many cases, business vocabulary is mapped to a data model whose structure could be slightly different to the business vocabulary. The business vocabulary usually provides a simplified view of this data model. For instance, a person concept may be associated a zip code property in the vocabulary while the corresponding data model features this zip code in the context of an address class, associated to the person class.
Domain specific language (DSL) processor 306 is for computing controlled language grammar 308 from DSL syntax 302 and business vocabulary 304.

Domain context-free grammar 308 is built by DSL processor 306 from DSL syntax 302 and business vocabulary 304.

Parser 310 is for parsing business statements, for instance as shown in editor 314, against domain context-free grammar 308 and semantic actions 316 and transforming them into abstract syntax tree 312.

Abstract syntax tree 312 is an internal representation of the concepts and actions defined by the statements.

Editor 314 is for allowing a user to edit statements 318 compliant with DSL syntax 302 of a given domain specific language. Editor 314 is for providing syntax highlighting, error reporting, and code completion functionalities that assist the user in capturing business statements. Editor 314 is interactive and allows a user to input and modify business statements, taking into account suggestions and assistance tips proposed to him.

Semantic actions 316 provide action code corresponding to statements so that the abstract syntax tree 312 can be executed by the decision execution server 3.

Referring to Figure 3B, extended decision management module 5’’ of the preferred embodiment comprises: all the described components of decision management module 5.’

Extended decision management module 5’’ further comprises: augmented business vocabulary 350; extended domain grammar 352; discovery patterns 353; extended parser 354; discovery pattern mechanism 356; and extended editor 358.

Augmented business vocabulary 350 is for the temporary storage of new business vocabulary for discovered patterns. At a later time, the new business vocabulary will be merged with the main business vocabulary 304.

Extended domain grammar 352 comprises domain grammar 308 decorated with discover patterns 353.
Discovery patterns 353 are added to a grammar to make an extended domain grammar. These patterns are abstract grammar rules for pattern matching when existing grammar rules are not able to exactly match the processing text to the existing language syntax and vocabulary. A discovery pattern decorates one or more grammar construct and is associated with an action that has an impact to existing vocabulary. Depending on the discovery pattern, the action can add a new concept, add a new constant or add a new phrase. A discovery pattern is made of a kind of regular expression, specific to a given human language (English or French for instance) that allow for matching of a sequence of words and for recognizing them as given type of vocabulary element. The exact form of the regular expression could be specific to framework in which these patterns are implemented. For instance, the regular expressions could take into account singular or plural form of words to refine the matching; if the pattern matches “premium clients” sequence of words, the pattern could infer that we talk about a “premium client” business concept.

More generally, the way the regular expression is specified and the way the matching is performed is non-trivial and not limited to what is described in this disclosure. In addition, several discovery patterns may apply, and a strategy can be elaborated to select one given pattern, based on its probability or some other strategy. For instance, the discovery pattern may want to avoid matching keywords of the language or words that are usually not significant in a given language to refer to elements of the vocabulary. Words like “if”, “of”,” the”, and “a” are not likely used in the designation of a business concept. The discovery pattern mechanism 356 can decide to not take them into account.

Extended parser 354 is for applying discovery patterns 353 as part of the discovery pattern mechanism 356. Extended parser 354 is activated when the input text does not match the specified domain grammar 308.

Discovery pattern mechanism 356 is for discovering patterns defined by discovery patterns 352. When discovered, discovery pattern mechanism 356 is for creating a vocabulary element that is added to the abstract syntax tree and collected in temporary augmented business vocabulary 350.

Extended editor 358 is for providing feedback to a user on business vocabulary modifications suggested by discovery pattern mechanism 356.
Referring to Figure 4, decision management system 2, including extended decision management module 5", performs extended decision management process 400 of the preferred embodiment comprising logical process steps 401 to 413.

Step 401 is for forming extended domain grammar 352 with discovery patterns 353 and domain context-free grammar 308. In the initial state, extended editor 358 is ready to capture business statements for a given business vocabulary and a given domain specific language. Extended parser 354 has been initialized with the extended domain grammar 352. Extended parser 354 has also been extended to communicate with discovery pattern mechanism 356.

Step 402 is for receiving statements at extended parser 354. Using extended editor 358, a user inputs the text of the desired business statement. For the purposes of explaining the embodiment, the text refers to a vocabulary element that does not exist in the current version of the vocabulary. For example “If the number of clients, where the income of each client is more than $100,000, is at least 5 then notify the supervisor”. In this example, the underlined words “client” and “income” are concepts that do not exist in the existing business vocabulary 304.

Step 403 is for parsing and analyzing a statement. Extended parser 354 parses the text of the statement and activates extended error recovery to report errors of missing terms. In the example “client” and “income” are not part of the extended domain grammar 352 or the business vocabulary 304.

Step 404 is for determining if type of error is one that can lead to a new concept and, if so determined, moves the process flow to step 405. If the type of error could not lead to a new concept then the process moves back to step 402 and error and statement processing of the prior art continues.

Step 405 is for matching discovery patterns that have been added to the extended grammar, in order to analyze the statement.

Step 406 is for parsing the non-recognized fragments and to find a best match for potential new vocabulary elements.
Step 407 is for deciding if a pattern has been matched and for forwarding the process to step 408 if no pattern is found. If a pattern is found then further processing continues at step 409. If more than one pattern is found then the selection of a best match pattern may be driven by a best match analysis according to a best match strategy.

Step 408 is the end of the extended process as the statement is in error and no pattern has been matched.

Step 409 is for creating a vocabulary element from the pattern that matches. A vocabulary element is created for all the concepts and phrases associated to that element.

Step 410 is for deciding if the vocabulary element is a type of element or an instance of an element. If the new vocabulary element is not a typed one, for example an action, the extended process continues with no type inference to step 412. Otherwise the type of the new element is inferred at step 411.

Step 411 is for inferring the type of vocabulary element. The type could be inferred according to where the pattern matched in the syntax tree. For instance, a concept property used in an arithmetic expression forces the type to be numerical. The type inference mechanism could be more complex than that and may use information coming from the underlying data model.

Step 412 is for adding the new vocabulary element to the augmented business vocabulary 350.

Step 413 is for re-generating the extended domain grammar 352 from the augmented business vocabulary 350. The process continues to analyze user input with new extended domain grammar 352 at step 403.

The extended process iterates until either the statement has no more errors or the statement has an error that cannot be recovered by the discovery pattern mechanism (for example the vocabulary cannot be extended to support the statement).

Figure 5 is an example screen showing a graphical user interface (GUI) 500 for the editor of the preferred embodiment. GUI 500 comprises: window control 502; and document frame
Window control 502 provides for minimizing; maximizing and closing of the GUI 500.

Document frame 504 is for displaying and editing of code statements. In this case an example set of statements for a business rule named “checkIncome” is displayed.

The example code comprises: definitions (labeled Definitions); a conditional part (labeled If); an action part (labeled Then). Underlining signifies a functional or operational aspect to the terminology and further underlined terms include: set; to; is at least; in; and refuse the loan with the message. Terms inside double quotes signify plain text to be treated as text (in this case by outputting a message). Terms inside single quotation marks signify known business vocabulary. Terms that are double underlined are not recognised in the present vocabulary.

**Definitions**

- set ‘minimum income’ to 0.37 * the yearly income of ‘the borrower’;
- If
  - the yearly repayment of ‘the loan’ is at least ‘minimum income’
- Then
  - in ‘the loan report’, refuse the loan with the message “Too big Debt/Income ratio:” + the yearly repayment of ‘the loan’ / the yearly income of ‘the borrower’;

In the above example, the terms yearly repayment and yearly income are not recognized during parsing and sent for pattern matching and concept creating. In this case, the same pattern was recognized and comprised: a preceding article ‘the’; followed by some unknown text; followed by ‘of’; and followed by a known concept (‘the borrower’ and ‘the loan’).

Other examples of discovery patterns that would be part of an extended grammar could include text that is preceded by any grammar article: [“a” words], [“an” words], [“the” words], [“each” words], [“the” “number” “of” plural words]. An extended grammar would be decorated with such discovery patterns that match the reference to a concept. This discovery pattern is associated to the grammar rule that match concepts in BAL grammar. The pattern is activated if a statement refers to a concept which is not part of the vocabulary. ‘words’ is a pattern of text that matches many words excluding keywords that are listed separately.
‘plural_words’ is a pattern that matches many words whose the last letters are in a plural form, as specified by the verbalization rules.

A discovery pattern is applied with a lower priority than the other grammar rules because it is more likely that an actual grammar rule applies than a new concept is intended to be created.

A property concept involves a relationship between two concepts, an ‘owner’ concept and an ‘owned’ concept (the property). Usually several phrases may exist in the vocabulary to refer to this relationship. A property concept is usually strongly typed and the pattern is responsible for recognizing a property concept, as well as its type. For instance in the example given above, ‘income’ is a property of the ‘client’ concept. Since this property concept is used with operators that usually apply to number or other comparable types, the type could be inferred.

A property concept pattern has the following description: [“the” words “of” expression], [expression “’s” words]. In the previous example, this pattern matches the ‘income’ concept property. In this case, discovery pattern action does two different things: adds the property concept to the business vocabulary; and determines the property concept type.

The preferred embodiment checks which type is expected at pattern location. This mechanism is called “propagated type” and allows retrieval of the information that ‘income’ should be typed as ‘dollars’ for the full expression to be semantically correct.

Further embodiments of the invention are now described. It will be clear to one of ordinary skill in the art that all or part of the logical process steps of the preferred embodiment may be alternatively embodied in a logic apparatus, or a plurality of logic apparatus, comprising logic elements arranged to perform the logical process steps of the method and that such logic elements may comprise hardware components, firmware components or a combination thereof.

It will be equally clear to one of skill in the art that all or part of the logic components of the preferred embodiment may be alternatively embodied in logic apparatus comprising logic elements to perform the steps of the method, and that such logic elements may comprise components such as logic gates in, for example a programmable logic array or application-specific integrated circuit. Such a logic arrangement may further be embodied in enabling
elements for temporarily or permanently establishing logic structures in such an array or circuit using, for example, a virtual hardware descriptor language, which may be stored and transmitted using fixed or transmittable carrier media.

In a further alternative embodiment, the present invention may be realized in the form of a computer implemented method of deploying a service comprising steps of deploying computer program code operable to, when deployed into a computer infrastructure and executed thereon, cause the computer system to perform all the steps of the method.

It will be appreciated that the method and components of the preferred embodiment may alternatively be embodied fully or partially in a parallel computing system comprising two or more processors for executing parallel software.

Referring to Figure 6, an example parallel computing embodiment is described comprising parallel sets of comprises parallel computing system 10P for parallel processing of instructions. Parallel computing system 10P may be described in the general context of parallel computer system executable instructions, such as parallel program modules, being executed by parallel computing system 10P. Generally, parallel program modules may include routines, programs, objects, components, logic, data structures, that perform particular tasks or implement particular abstract data types. Parallel computing system 10P comprises: parallel computer server 12P; input devices 14P; and output devices 16P. Parallel computing system 10P communicates with user 18P via computer client 40 through network 20P (network 20P can be a cloud network). User 18P provides input and receives output from the parallel computing system 10P but is not part of parallel computing system 10P. Network 20P provides access to network attached devices and is not part of the parallel computing system 10P.

Parallel computer server 12P comprises: CPU 22A, CPU 22B, CPU 22C; network adapter 24P; bus 28P and memory 30P.

Bus 28P represents one or more of any of several types of bus structures, including a memory bus or memory controller, a peripheral bus, an accelerated graphics port, and a processor or local bus using any of a variety of bus architectures.
Memory 30P includes computer system readable media in the form of volatile memory 32P (such as random access memory and cache memory (not shown)) and in the form of non-volatile or persistent memory 34P.

In parallel computing embodiment, program modules comprise: decision execution server module 3’; decision designer module 4’; and extended decision management module 5’’. Modules are stored in persistent memory 34P, until needed, as well as an operating system, one or more application programs, a database management system and other program modules. Each of the operating system, one or more application programs, other program modules, and program data or some combination thereof, may include an implementation of a networking environment. Modules 3’, 4’ and 5’’ are provided to carry out the functions and/or methodologies of the embodiments in a parallel environment as described herein.

Modules 3’, 4’ and 5’’ are autonomous parts of the embodiment. In operation, these parts are extracted from persistent memory 34P and loaded into volatile memory 32P so that they may be executed separately and therefore in parallel by respective CPUs 22A, 22B and CPU 22C.

In this example, three CPUs are shown but two or more than three CPUs can be used to build alternative parallel embodiments. In this example, three separate CPUs are used but a single processing unit having multiple cores could be used to build an alternative embodiment. In this example, the CPUs are physical CPUs but. In the described parallel computing embodiment the parallel computer server 12P comprises multiple processing units. In an alternative parallel computing embodiment, a parallel computer server comprises a single processor having multiple cores. In a first virtual parallel computing embodiment, a computer server comprises a virtual computing environment and virtual parallel processing units could be used to build a virtual parallel computing embodiment. A computer server comprises a virtual computing environment having a virtual processing unit with multiple virtual cores. Further embodiments can comprises combinations of: real processing units; real processing unit cores; virtual processing units; and virtual parallel processing cores.

As will be appreciated by one skilled in the art, aspects of the present invention may be embodied as a system, method or computer program product. Accordingly, aspects of the present invention may take the form of an entirely hardware embodiment, an entirely software embodiment (including firmware, resident software, micro-code, etc.) or an embodiment
combining software and hardware aspects that may all generally be referred to herein as a “circuit,” “module” or “system.” Furthermore, aspects of the present invention may take the form of a computer program product embodied in one or more computer readable medium(s) having computer readable program code embodied thereon.

Any combination of one or more computer readable medium(s) may be utilized. The computer readable medium may be a computer readable signal medium or a computer readable storage medium. A computer readable storage medium may be, for example, but not limited to, an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system, apparatus, or device, or any suitable combination of the foregoing. More specific examples (a non-exhaustive list) of the computer readable storage medium would include the following: an electrical connection having one or more wires, a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), an optical fiber, a portable compact disc read-only memory (CD-ROM), an optical storage device, a magnetic storage device, or any suitable combination of the foregoing. In the context of this document, a computer readable storage medium may be any tangible medium that can contain, or store a program for use by or in connection with an instruction execution system, apparatus, or device.

A computer readable signal medium may include a propagated data signal with computer readable program code embodied therein, for example, in baseband or as part of a carrier wave. Such a propagated signal may take any of a variety of forms, including, but not limited to, electro-magnetic, optical, or any suitable combination thereof. A computer readable signal medium may be any computer readable medium that is not a computer readable storage medium and that can communicate, propagate, or transport a program for use by or in connection with an instruction execution system, apparatus, or device.

Program code embodied on a computer readable medium may be transmitted using any appropriate medium, including but not limited to wireless, wireline, optical fiber cable, RF, etc., or any suitable combination of the foregoing.

Computer program code for carrying out operations for aspects of the present invention may be written in any combination of one or more programming languages, including an object
oriented programming language such as Java, Smalltalk, C++ or the like and conventional
procedural programming languages, such as the "C" programming language or similar
programming languages. The program code may execute entirely on the user's computer,
partly on the user's computer, as a stand-alone software package, partly on the user's computer
and partly on a remote computer or entirely on the remote computer or server. In the latter
scenario, the remote computer may be connected to the user's computer through any type of
network, including a local area network (LAN) or a wide area network (WAN), or the
connection may be made to an external computer (for example, through the Internet using an
Internet Service Provider).

Aspects of the present invention are described below with reference to flowchart illustrations
and/or block diagrams of methods, apparatus (systems) and computer program products
according to embodiments of the invention. It will be understood that each block of the
flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart
illustrations and/or block diagrams, can be implemented by computer program instructions.
These computer program instructions may be provided to a processor of a general purpose
computer, special purpose computer, or other programmable data processing apparatus to
produce a machine, such that the instructions, which execute via the processor of the
computer or other programmable data processing apparatus, create means for implementing
the functions/acts specified in the flowchart and/or block diagram block or blocks.

These computer program instructions may also be stored in a computer readable medium that
can direct a computer, other programmable data processing apparatus, or other devices to
function in a particular manner, such that the instructions stored in the computer readable
medium produce an article of manufacture including instructions which implement the
function/act specified in the flowchart and/or block diagram block or blocks.

The computer program instructions may also be loaded onto a computer, other programmable
data processing apparatus, or other devices to cause a series of operational steps to be
performed on the computer, other programmable apparatus or other devices to produce a
computer implemented process such that the instructions which execute on the computer or
other programmable apparatus provide processes for implementing the functions/acts
specified in the flowchart and/or block diagram block or blocks.
The flowchart and block diagrams in the Figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods and computer program products according to various embodiments of the present invention. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of code, which comprises one or more executable instructions for implementing the specified logical function(s). It should also be noted that, in some alternative implementations, the functions noted in the block may occur out of the order noted in the figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, can be implemented by special purpose hardware-based systems that perform the specified functions or acts, or combinations of special purpose hardware and computer instructions.

It will be clear to one skilled in the art that many improvements and modifications can be made to the foregoing exemplary embodiment without departing from the scope of the present invention.
CLAIMS

1. A system for managing and executing interpreted language code comprising:
a parser for parsing controlled language code against both a language grammar and for building an execution model comprising parsed known concepts;
a pattern matching engine for matching patterns in the controlled language code with a patterns for a new concept; and
a concept engine for creating a new concept from the matched pattern and controlled language code and for saving the new concept into a object model and into a language grammar so that current and future parsing errors will not occur when recognizing the new concept.

2. A system as claimed in claim 1 wherein an error during parsing of the language grammar triggers a search to match the error with a new concept.

3. A system as claimed in claim 1 or 2 whereby the interpreted language code comprises rules and actions for making decisions and the system is a decision management system.

4. A system as claimed in claim 1, 2 or 3 further comprising matching patterns in the controlled language against patterns for types of new concepts.

5. A system as claimed in any one of claims 1 to 4 wherein the pattern matching engine is for matching patterns in the controlled language against patterns for new properties of concepts; and the concept engine is for creating a new properties of a concept from the matched pattern and the controlled language code and saving the property of a concept with a corresponding action into the object model; and

wherein execution engine is for executing the concepts and actions in the object model.

6. A system as claimed in claim 5 wherein the pattern matching engine is for matching patterns in the controlled language against patterns for types of new properties.

7. A system as claimed in any one of claims 1 to 6 further comprising an execution engine for executing the concepts of the object model.

8. A method for managing and executing interpreted language code comprising:
parsing controlled language code against a language grammar to build an execution model comprising known concepts;
matching patterns in the controlled language against patterns for new concepts;
creating a new concept from the matched pattern and the controlled language code;
saving the new concept into a object model and into a language grammar so that current and future parsing errors will not occur when recognizing the new concept.

9. A method as claimed in claim 8 wherein an error during parsing of the language grammar triggers a search to match the error with a new concept.

10. A method as claimed in claim 8 or 9 whereby the interpreted language code comprises rules and actions for making decisions.

11. A method as claimed in claim 8, 9 or 10 further comprising matching patterns in the controlled language against patterns for types of new concepts.

12. A method as claimed in any one of claims 8 to 11 further comprising:
matching patterns in the controlled language against patterns for new properties of concepts;
creating a new properties of a concept from the matched pattern and the controlled language code; and
saving the property of a concept with a corresponding action into the object model; and executing the concepts and actions in the object model.

13. A method as claimed in claim 12 further comprising matching patterns in the controlled language against patterns for types of new properties.


15. A computer program product stored on a computer readable medium and loadable into the internal memory of a digital computer, comprising software code portions, when said program is run on a computer, for performing the method of any of claims 8 to 14.