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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
29 December 2011 (29.12.2011)

PCT

(10) International Publication Number
WO 2011/160205 A1

(51) International Patent Classification:
H04L 12/28 (2006.01) *G06F 17/30* (2006.01)

(21) International Application Number:
PCT/CA2011/000719

(22) International Filing Date:
22 June 2011 (22.06.2011)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/357,509 22 June 2010 (22.06.2010) US
61/498,899 20 June 2011 (20.06.2011) US

(71) Applicant *(for all designated States except US)*: **PRIMAL FUSION INC.** [CA/CA]; 7 -258 King Street North, Waterloo, Ontario N2J 2Y9 (CA).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **SWEENEY, Peter, Joseph** [CA/CA]; 56 Kilbirnie Court, Kitchener, Ontario N2R 1B8 (CA).

(74) Agent: **DE FAZEKAS, Anthony**; MILLER THOMSON LLP, 40 King Street West, Scotia Plaza, Suite 5800, Toronto, Ontario M5H 3S1 (CA).

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

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK,

[Continued on next page]

(54) Title: SYSTEMS OF COMPUTERIZED AGENTS AND USER-DIRECTED SEMANTIC NETWORKING

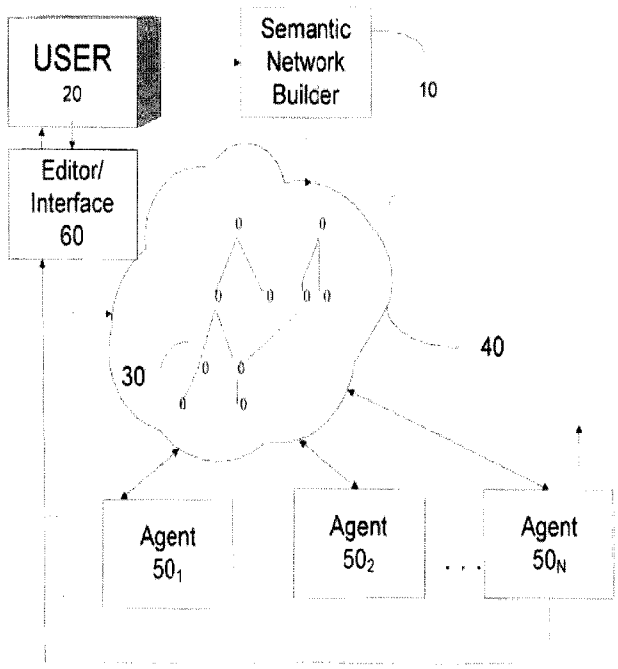

FIG. 1

(57) Abstract: A system, method and computer program product in which semi-autonomous agents interact with a semantic network. In a basic embodiment of the system, a data structure providing a semantic network is provided in a non-transitory, computer-readable medium within a computer network. A plurality of computer-implemented agents are deployed within the computer network and interactive with the semantic network. A user interface is provided and configured to permit a user to create and/or modify the semantic network. The agents are configured to read and modify the semantic network without receiving explicit instructions from a user after their initial deployment, whereby the agents operate as assistants to support the user's use of the network.

WO 2011/160205 A1

SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

SYSTEMS OF COMPUTERIZED AGENTS AND USER-DIRECTED SEMANTIC NETWORKING

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims the benefit of U.S. Provisional Application No. 61/357,509, filed on June 22, 2010, titled "Systems of Computerized Agents and Semantic Networks," and U.S. Provisional Application No. 61/498,899, filed on June 20, 2011, titled "Method and Apparatus for Preference Guided Data Exploration," the entirety of which are incorporated herein by reference.

FIELD OF INVENTION

The teachings disclosed herein relate to deployment, in an information system environment incorporating one or more user-directed semantic networks representing knowledge domains, of multiple computer-implemented software agents capable of autonomous action and of interaction with such networks to perform a variety of tasks. In so doing, an autonomous or intelligent distributed information processing system respective of user knowledge is formed, constituting another aspect of the teachings presented herein.

BACKGROUND

The Internet is a global system of interconnected computer networks that store a vast array of information. The World Wide Web (WWW) is an information sharing model built on top of the Internet, in which a system of interlinked hypertext documents are accessed using particular protocols (e.g., the Hypertext Transfer Protocol and its variants).

Because of the enormous volume of information available via the WWW and the Internet, and because the available information is distributed across an enormous number of independently owned and operated networks and servers, locating and acting upon desired content on the WWW and the Internet presents challenges. Similar challenges exist when the information of interest is distributed across large private networks, as well.

Various tools such as search engines have been developed to aid users in locating desired content on the Internet and other networks, and software agents have been developed to effectuate some desired user actions relative to such content.

A search engine is a computer program that receives a search query from a user (e.g., in the form of a set of keywords) indicative of content desired by the user, and returns information and/or hyperlinks to information that the search engine determines to be relevant to the user's search query.

Search engines typically work by retrieving a large amount of material satisfying the search criteria specified in the search query (either exactly or within some boundaries), such as a large number of WWW web pages and/or other content, using a computer program called a web crawler that traverses the WWW in an automated fashion (e.g., following every hyperlink that it comes across in each web page that it finds). The retrieved web pages and/or content are analyzed and information about the web pages or content is stored in an index. When a user issues a search query to the search engine, the search engine uses the index to identify the web pages and/or content that it determines to best match the user's search query and returns a list of results with the best-matching web pages and/or content. Frequently, this list is in the form of one or more web pages that include a set of hyperlinks to the web pages and/or content determined to best match the user's query.

Software agents have been developed as proxies for their users, to perform various tasks such as facilitating the execution of user-desired tasks relative to network information. In the field of computer science, software agents are software entities which execute on processors to perform a variety of functions, typically on behalf of users who have deployed them. More precisely, the term "agent" is applied to a software entity that is capable of acting with a certain degree of autonomy (agents have capabilities of task selection, prioritization, goal-directed behavior, decision-making without significant human intervention) in order to accomplish tasks on behalf of a user. While software (i.e., computer program code) by itself is no more than inert information, we assume when referring to a software agent that it is code actively executing on a processor for only then can it be a functional element. Additionally, agents commonly exhibit attributes such as persistence (code is not executed on demand but runs

continuously and decides for itself when it should perform some activity); social ability (agents are able to engage other components through communication and they may therefore, effectively, collaborate (i.e., act in concert) on a task); and reactivity (agents perceive the context or environment in which they operate and react to it appropriately).

However, software agents require knowledge models that provide a domain from which to operate. Conventional agents will often work from a single key word or phrase provided by the user. Further, traditional agents work in isolation and do not evolve in their function. Thus, making an agent that meets the multi-faceted needs of a user and, adapts to meet the changing needs of the user, requires the construction and maintenance of a knowledge model that is expensive and difficult to scale. The teachings disclosed herein are directed at least partly towards better addressing user needs.

SUMMARY

For purposes of the current discourse, examples of the kind of functions which may be performed by software agents include at least information harvesting (harvesting agents), deep data analytical mining (data mining agents), information retrieval (search agents), social networking and other personal tasks (social agents, also called personal agents or user agents), e-commerce tasks (shopping agents, also called shopping bots or buyer agents), and monitoring and surveillance.

Buyer agents "travel" or "crawl" around a network (e.g., the global internet) (i.e., they access different network addresses) retrieving information about goods and services, such as the prices different vendors are charging. These agents, also known as 'shopping bots', work efficiently for commodity products such as CDs, books, and electronic components.

User agents, or personal agents, are intelligent agents that, on a user's behalf, take action other than one assigned to other types of agents. In this category belong those intelligent agents that already perform, or will shortly perform, tasks such as checking e-mail and sorting it according to the user's order of preference and alerting the user when important emails arrive.

Harvesting agents may extract surface-level information, such as snippets of data or chunks of content.

Data mining agents allow pattern extraction by performing deep analytics on large data sets.

Search agents may find information on the subject of a user's choice and monitor for changes in the status of certain information.

This list of agent types and functions is not intended to be exhaustive and is only exemplary, of course.

The use of agents is increasing in number and type, to provide expanding automated functionality to a growing number of users. To effectuate such operation, users must be able to interact with agents and agents must be able to interact with the information content of the network. The user-agent interaction must occur notwithstanding the goal of agent autonomy; ultimately, the tasks are directed to helping users become more productive.

Additionally, software agents need to work together efficiently within such environments. A shopping agent, for example, may be designed and configured to use the results of a search performed by a search agent. Therefore, they must share semantics of their data elements. This can be done by having computer systems publish their metadata (defining data relationships and qualities) using shared ontologies for representing knowledge. Within such a framework, an agent uses its access methods (which need not be the same from one agent to the next) to go into local and remote databases to forage for content. These access methods may include, but are not limited to, setting up news stream delivery to the agent, or retrieval from bulletin boards, or using a so-called spider program to traverse the Web. The content that is retrieved in this way may already be partially filtered – by the selection of the newsfeed or the databases that are searched. The agent next may use its detailed searching or language-processing machinery to extract keywords or signatures from the body of the content that has been received or retrieved. This abstracted content (or event) is then passed to the agent's reasoning or inferencing machinery in order to decide what to do with the new content. This process combines the event content with the rule-based or knowledge content provided by the user. If this process finds a good hit or match in the new content, the agent may use another piece of its (i.e., code base) machinery to do a more detailed search or analysis on the content. Finally, the agent may decide to take an action based on the new content; for example, to notify the user that an important event has occurred.

This action may be verified by a security function and then given the authority of the user. The agent makes use of a user-access method to deliver that message to the user. If the user confirms that the event is important by acting quickly on the notification, the agent may also employ its learning machinery to increase its weighting for this kind of event.

Software agents thus may offer various benefits to their end users by automating complex or repetitive tasks. However, there are several potential limitations and impacts of this technology that need to be considered. For example, to provide an agent-based system capable of many different tasks for many different people, a great deal of effort must be directed to modeling the knowledge domain within which the agents and people will interact. For example, to provide an effective agent-based system for making restaurant reservations, knowledge of restaurants and how to book reservations must be modeled in the system. This knowledge requirement also compounds the complexity and sophisticated capabilities needed in the agents, reflected in the term, "intelligent" agents. This requirement for intelligent agents slows the development and proliferation of agent-based systems, and increases their development cost and the complexity of agents. Thus, the available tools used for manufacturing agents seem to be producing ever more capable, but ever more complex agents. As agents get more complicated, the task of assuring their correct functioning under all circumstances becomes more challenging and costly.

Further, agents rarely have a direct or detailed "insight" into the needs and requirements of the people they are supporting. That is, while they may be designed for autonomy, their range of potential action is limited to that which is pre-programmed, they lack self-awareness and awareness of the objectives, needs and requirements of their users. Thus, as those needs and requirements change, users must interact with their agents to make known these changes by revising agent programming, requiring both complicated interfaces and user attention. As a result, instead of the system adapting to the user, if efficiency is to be maintained, the user must effectuate the adaptation. It may be desirable, instead, for the behavior of an agent to adapt itself to a user's changed needs by responding to changes in the user's behavior manifesting changes in the user's requirements, without explicit user instruction.

This is challenging, but consistent with an important design objective for agents – that they be able to act autonomously – i.e., without significant human intervention. Designing to this objective, though, is somewhat inconsistent with the notion of agents that are responsive to user intentions. There is therefore a need for a middle ground between manual human activities and fully autonomous software agents, agents that are able to respond to their users without direct or explicit user direction.

Of course, the ability of agents to process information and derive meaningful results therefrom is highly dependent on the knowledge models upon which the agents operate (as both input and output). In general, the knowledge upon which such agents act is an "ontology," i.e. a formal, structured representation of the knowledge embodied by a set of concepts within a "particular domain" and the relationships between those concepts. An ontology allows agents to operate logically on the properties of that domain – i.e., to "reason," and may be used to describe the domain itself. Thus, an ontology provides a shared vocabulary, which can be used to model a domain — that is, the type of objects and/or concepts that exist, and their properties and relations. Within computer science, an ontology is a model for describing the world (i.e., the domain) that consists of a set of types, properties, and relationship types. Exactly what is modeled varies. There is also generally an expectation that there be a close resemblance between the real world and the features of the model in an ontology.

Contemporary ontologies share many structural similarities, regardless of the language in which they are expressed. As mentioned above, most ontologies describe individuals (instances), classes (concepts), attributes, and relations. There exist many tools today for building ontologies and it is not necessary for purposes of this discussion to single out any of them. Some employ standards-based ontology languages; some employ proprietary ontology languages. Those skilled in the art will be able to select an appropriate ontology building tool for use in practice of the inventive embodiments and concepts described herein.

Since domain ontologies represent concepts in very specific and often varied ways, they are often incompatible. A domain ontology (or domain-specific ontology) models a specific domain, or part of the world. It represents the particular meanings of terms as they apply to that domain. For example the word "card" has many different

meanings. An ontology about the domain of "poker" would model the "playing card" meaning of the word, while an ontology about the domain of "computer hardware" would model the "punched card" and "video card" meanings. Similarly, the word "green" has different meanings, not only within different domains, but also from the context of usage. It might be modeled primarily as a color in a domain about fashion, while a domain about environmental issues might model it as having low environmental impact or using little energy." A domain about people might model "green" as a surname or as a synonym to "inexperienced." So the same term might even be modeled as different parts of speech and as having different meanings in the same domain, depending on the part of speech. In other words, a useful ontology must, relative to the domain, disambiguate the meanings attributable to terms so that, among other things, the actions of multiple agents relative to the ontology will be consistent, meaningful and useful.

Changing the way computerized software agents, domain ontologies and users interact can lead to several advantages. Among them, agents can be less complex, allowing complex behaviors instead to emerge from the indirect interactions of multiple simpler agents via changes to the user-directed knowledge model on which they operate. Moreover, users can more intuitively and more simply express their intentions by operating directly on the data in the knowledge model, as well, without having to re-configure or reprogram or re-task their agents.

Accordingly, a first aspect of the invention is a computer system including, in a non-transitory, computer-readable medium within a computer network, a data structure providing a semantic network. (This medium need not be localized and may be distributed, even involving multiple types of data storage.) A plurality of computer-implemented agents are deployed within said computer network and are interactive with the semantic network. The system further includes a user interface configured to permit a user to at least modify the semantic network. The agents are configured to read and modify the semantic network without receiving substantial explicit instructions from a user. Modifying the semantic network may include changing, editing, altering, augmenting, adding to or deleting from the semantic network. In some embodiments, the agents may include at least one of a harvesting agent, data mining agent, search agent, connecting agent, personal agent or shopping agent, among other possibilities. The

autonomous nature of agent operations allows at least two of the plurality of agents to collaborate with each other, either explicitly through direct communication or implicitly through indirect communication effected by interaction through the semantic network. In some embodiments, at least one of the agents is configured to synthesize a concept to the semantic network, and add a node, or to change a value or delete a node or edge of the network from the data structure. In some embodiments, at least a second agent of the plurality of agents acts upon or in response to the changed value, addition or deletion of the entry.

According to a second aspect, a method comprises providing a semantic network within a computer network; providing a plurality of computer-implemented agents deployed within said computer network and interactive with the semantic network, the agents being configured to read, edit or otherwise influence the semantic network without receiving explicit instructions from a user; and providing a user interface configured to permit a user to edit the semantic network.

A further aspect is a method to decouple user and agent actions with respect to a semantic network. An information exchange platform is provided, comprising an editable semantic network instantiated in a non-transitory, computer-readable medium within a computer network. A plurality of computer-implemented agents are deployed within the computer network and interactive with the semantic network, the agents being configured to autonomously read and modify the semantic network. A provided user interface is configured to permit a user to at least receive reports regarding or to modify the semantic network.

According to still another aspect, a method comprises making available to users of a computer network a semantic network building took and a plurality of computer-implemented agents deployable within said computer network and interactive with a semantic network constructed by the user with the tool, the agents being configured to read and modify the semantic network without receiving explicit instructions from a user. In some embodiments, the method further includes providing a user interface configured to permit a user to modify the semantic network.

According to yet still another aspect, a non-transitory computer readable storage medium is disclosed that stores processor executable instructions including software

modules that may perform any of numerous tasks in accordance with some embodiments. For example, the instructions include a semantic network module configured to provide a data structure that includes a semantic network. The instructions also include an agent-interface module configured to allow interaction between a plurality of computer-implemented agents and the semantic network, and a user-editing module configured to permit, through a user interface, modification of the semantic network by a user.

BRIEF DESCRIPTION OF DRAWINGS

In the drawings,

FIG. 1 is a block diagram representing an embodiment of some aspects of the invention;

FIG. 2 is a block diagram further illustrating an agent ecosystem according to some embodiments;

FIG. 3 is a diagrammatic illustration of an exemplary operation of a user editing a semantic network (e.g., an ontology or knowledge model) according to some embodiments;

FIG. 4 is an illustration of activities resulting from an ecosystem of agents;

FIG. 5 is an illustration of a system architecture within which agents can be deployed;

FIG. 6 is a block diagram illustrating an exemplary computing system for use in practicing some embodiments of the present invention;

FIG. 7 is a diagram of a "query, sort, then scan" data exploration model, in accordance with prior art;

FIG. 8 is a diagram illustrating a relation, in accordance with some embodiments;

FIG. 9 is a flowchart of an illustrative preference modeling process, in accordance with some embodiments;

FIG. 10 is a diagram illustrating scopes obtained from a relation, in accordance with some embodiments;

FIG. 11 is a diagram illustrating scope comparators, in accordance with some embodiments;

FIG. 12 is a diagram illustrating conjoint preferences, in accordance with some embodiments;

FIG. 13 is a diagram of an illustrative mapping of a partial order to linear extensions, in accordance with some embodiments;

FIG. 14 is a diagram of an illustrative preference graph, in accordance with some embodiments;

FIG. 15 is a diagram of an illustrative computation of edge weights for different types of second-order preferences, in accordance with some embodiments;

FIG. 16 is a diagram of an illustrative page-rank based matrix for prioritized comparators, in accordance with some embodiments;

FIG. 17 is a diagram of an illustrative weighted preference graph and tournaments derived from it, in accordance with some embodiments;

FIG. 18 is a flowchart for an illustrative process for interactively specifying preferences, in accordance with some embodiments;

FIG. 19 illustrates a method for constructing knowledge representations using knowledge representation rules, statistical graphical models, and user feedback; and

FIG. 20 illustrates a method of AKRM employing a preference ranking engine.

DETAILED DESCRIPTION

The methods, systems and products disclosed herein can be implemented using existing agent creation tools and any of various available techniques for knowledge representation, including ontology languages and ontology building tools, as well as future agent and knowledge representation tools. While it is not intended that the claimed invention be limited to specific knowledge representations, a preferred form is the type of ontology referred to as a semantic network. Semantic networks are explained in many sources, noteworthy among them being U.S. patent application no. 12/671,846, titled Method, System, And Computer Program For User-Driven Dynamic Generation Of Semantic Networks And Media Synthesis by Peter Sweeney et al, which is hereby incorporated by reference.

Semantic networks may be used as forms of knowledge representation. A semantic network may be represented as a directed graph consisting of vertices that

represent concepts, and edges that represent semantic relations between the concepts, and encoded in a corresponding data structure in a computer-readable storage medium.

Semantic networks have a broad utility as a form of knowledge representation. As machine-readable data, they can support a number of advanced technologies, such as artificial intelligence, software automation and agents, expert systems, and knowledge management. Additionally, they can be transformed into various forms of media (i.e. other knowledge representations). In other words, the synthesis or creation of semantic networks can support the synthesis of a broad swath of media to extract additional value from the semantic network.

Thought networking, and semantic synthesis as made available, for example, by Primal Fusion, Inc. of Waterloo, ON, Canada, www.primal.com, provides a good method and system for generating user-directed semantic networks that represent a user's knowledge relative to a domain. Semantic synthesis constructs semantic networks that encode such thoughts and intentions of the user. Encoded as data in organized data structures, these thoughts and intentions are then available for computing purposes, for example, in support of agent-based systems.

A thought network, also known as a knowledge network, refers to a type of user-directed semantic network (to contrast with semantic networks that are composed by the producers of information as opposed to the end-users). It represents users' thoughts as interconnected concepts. This lattice-like structure is how thoughts are represented as data and made concrete. Ideally, however, users will remain unaware of this deep structure as they interact with their thought networks.

As stated above, thought networking has previously been employed to capture users' thoughts and intentions. It is possible, however, also to use thought networking to generate semantic networks that may be used as input to software agents. In turn, those agents may output changes to those (or additional) semantic networks. If a domain knowledge base is captured in a semantic network and both users and agents interact with the semantic network, rather than interacting directly with each other, it is possible to dispense with all or most of the need for a user to interact directly with its agents. Agents can be simplified by reducing their functionality, by eliminating a direct user interface, by reducing their need to duplicate functions, and by allowing behavior to emerge from the collective actions of a number of agents of different types. Agent

functionality may even be reduced to a single function, provided a suitable collection of agents of differing functions is made available, to accomplish a desired set of tasks.

The collection of interacting agents or groups of agents is referred to as an "agent ecosystem." Enabling people to affect changes directly to the semantic networking environment lowers the costs and complexity associated with individual software agents and software agent ecosystem development.

One might say that people (users) participate in such a system or systems by constructing landscapes of their thoughts (in semantic data) while relatively simple agents (possibly large numbers of them) can work over that landscape of user intelligence to get work done. The agents may be analogous to ants roaming over the landscapes, cooperating with each other within the environment imposed by the architects of the landscapes (that is, the users). Further, if users and agents both can act by altering the landscape, they can influence the operation of the agents without the users having to directly control their agents. Decoupling the user interactions from the agent interactions provides for new applications/design patterns. Different types of agents (for example, search agents plus harvesting agents) may collaborate to accomplish tasks of emergent complexity. That is, one type of agent may call upon other types of agents to take actions, and the sophistication of the resultant action might eclipse, or go beyond to achieve more than the capabilities of each individual agent. For example, one agent may harvest information and decorate the semantic network with this information. A second agent, operating autonomously, may simply compose a report of changes to the semantic network. Each agent may respond to the environment of the knowledge landscape without requiring any understanding of how the landscape is composed or the higher-order behaviors that may emerge as agents and users interaction with the knowledge landscape.

Referring to FIG. 1, illustrated are the components of a basic, exemplary system implementing some of the above-discussed aspects. A semantic network (ontology) building tool 10 is used by a user 20 to build a semantic network 30 in a computer network-accessible memory 40, which may be local, remote or distributed memory such as memory distributed across the Internet, for example. The semantic network building tool may be any computer (hardware and software) which is suitably adapted and configured (by software, for example) to generate a semantic network when a user

provides the necessary inputs. Software agents 50_1 - 50_N, which may execute on computers (not shown) anywhere that have access to the semantic network, interact with the semantic network and perform their respective functions on the data in the network. As illustrated, it is presumed that at least one agent, such as 50_N, performs an output function and provides output to user 20. However, the results of agent interactions need not be output to the user in applications where the user merely wants tasks completed and does not require reporting on those tasks.

Once the semantic network exists, the user 20 may use an interface tool 60 to effectuate changes to the network. The interface tool may be any suitable editor which can show the semantic network to the user (e.g., in a graphical user interface) and allow the user to enter alterations to the network. Alterations may include changing data values, as well as adding entries to or deleting entries from the semantic network. Tools such as mind-mapping software or ontology-builders may be directed to these aspects of user interactions.

Note that the user(s) need not interact directly with the agents once the agents have been deployed (i.e., started). Naturally, someone has to deploy each agent, which in turn requires identifying which agents in the system will participate.

An agent may be personal to a user or an agent may be agnostic as to user identity. Thus, a shopping agent may be the personal agent of a specific user or a shopping agent may be available to any user that needs it, as an example.

The semantic network landscape in which the software agents may operate, enabling users to express themselves both by expressly editing the network and by allowing agents to deduce user intentions from combinations of actions. Agents may be plain and simple, with limited functionality. Manifestly, the user-network and agent-network interactions do not make up a direct dialogue between users and the software agents. This lack of direct dialogue is characteristic of the current system and method. People participate by constructing landscapes of their thoughts (in semantic data) while large numbers of these simple agents work over (i.e., interact with) that highly personalized landscape to get work done. Different types of agents (for example, search agents plus harvesting agents) may collaborate to accomplish tasks that, typically, no one agent type would be able to provide.

A specific example of a simple semantic network 30'and its use herein is shown in FIG. 2. Assume a user expresses a query to a semantic network synthesis tool (also called a builder or engine) 10, having (or having access to) a knowledge base 12 appropriate to a domain of interest. One or more agents 50' (e.g., search, filter and reporting agents) act autonomously (and independently) over the network 30'. Outcomes may be fed back to the user, as by modules 14, 16 and 18, or to the network (as indicated at 19) . The modules 14, 16, and 18 may also allow a user to instruct the agents or change their conditions, parameters, functions or status. Complex knowledge then may emerge from simple interactions over the user's landscape of knowledge as expressed in the knowledge base.

FIG. 3 illustrates that a user 20 may edit the knowledge base 12 used by a synthesis engine, also. For example, a visual editor 60 may be employed (it may be either part of the engine, as shown, or external to the engine. Thus, results may emerge from the combined actions of users and agents.

Selection of a synthesized term may allow for entity disambiguation or entity resolution, for example. Thus the term "nutrient" may mean something different to a botanist than it will to a fitness aficionado. Outcomes may be presented to a user, for example, according to an intelligent ranking. Such a ranking may be in accordance with the user's preferences that can be determined with the teachings disclosed in REFERENCE A, included below, entitled System and Method Directed Towards Preference Guided Data Exploration. Further, the underlying semantic network 30' may, based on user preferences, weight some concepts or topics more heavily than others by applying teachings as disclosed in REFERENCE B, included below, entitled System and Method of Preference Guided Data Explorations Applied to Atomic Semantics. Agent functioning may then be applied to a network focused on concepts more relevant to a user. Such weighting and preferential ranking may be determined based on user-click patterns, browser history, the selection of a synthesized concept, etc. that influence the semantic network without the user's explicit editing or amendment thereof.

One scenario where the teachings disclosed herein are illustrated may be where a user is relocating his or her residence and, thus, is interested in a new location. Using traditional user-agent methods, that user would have to modify all the agents that provide location-based information such as hyper-local news for notification of nearby-events.

Leveraging a user-based semantic network with harvesting, search or other type of agents instead allows the user to merely modify the knowledge base and thereby modify the semantic network to reflect the updated city, country or any other locality of interest. Each of the agents will then act upon this updated location information and more accurately serve the user, eliminating the need for the user to determine which agents require updated location information and without the user having to directly modify each of those agents with such updated location information. The agents may further act to modify the semantic network by adding related concepts to the new location of interest, such as including the new location's county, state or country in the case of an international move.

As another example, user interested in physical fitness may have harvesting agents deliver related content such as nutrition articles or training routines, or may have shopping agents notify the user of relevant gym equipment. If that user injures his shoulder, conventional approaches would require the user to modify each of those agents directly. Using the teachings disclosed herein, the user may instead edit the semantic network to reflect the new relevance to the user of rehabilitation or specific information about the shoulder, allowing agents that are tailored accordingly to deliver articles, routines and equipment of interest without direct user-agent interaction. Note that modification of the semantic network to include concepts such as rehabilitation can occur without the user's explicit incorporation of such. The user may simply enter "injury," which the network can identify as a concept related to "rehabilitation" via use of a reference corpus. Also, merely clicking on articles associated with injury or rehabilitation may reveal a user interest in such content and result in modification of the semantic network accordingly. The network can be dynamic, changing with time so as to apply less weighting and a weaker preference towards injury or shoulder rehabilitation-related concepts. This may occur as the frequency of the user's selection of such content decreases, suggesting the recovery of the user's shoulder.

Further, traditional agents with instructions related to highly user-dependent concepts require modification. For example, selection of "heroes" may mean comic book characters to an 8-yr old child but may mean civil rights leaders to 65-yr old citizen. Religion is another example of a concept that is highly user-dependent. The concept "religion" and associated content will strongly vary from a seasoned priest to a

young Buddhist. The traditional user-agent interaction requires every agent operating on the concept of religion or heroes to be identified by the user and modified to reflect the particular subjective views, experiences or interests of the user. Instead, augmenting the "religion" or "hero" concept in one's semantic network allows agents to act in accordance with the user's meaning. The result is a reduction in user-effort and agent complexity.

Agents may act in tandem and leverage information from each other. An agent that performs location-based information may do so based on content retrieved by a search agent. An article retrieved by a search agent for shoulder rehabilitation may suggest a particular type of yoga or hydro-therapy as treatment and modify the semantic network accordingly; the location-based agent may determine that treatment is relevant from the semantic network and identify facilities that perform such treatment in the region. Alternatively, a shopping agent may identify available products that can be used to perform the article's prescribed treatment.

In all of the foregoing cases, employing traditional agents would require the user to modify each agent serving that person, or a complicated agent would need to be constructed to accurately serve the user and perform varying tasks. The teachings disclosed herein can integrate general tasks such as searching, harvesting, organizing, connecting, tracking, collaborating, or reporting with aspects related to personal knowledge such as professional interests, personalized news, travel, finances, local search, education, hobbies, or health issues, to serve the user in a vast number of ways, such as those listed in FIG. 4.

FIG. 5 provides a non-limiting example embodiment of potential system components in some implementations of an agent ecosystem as taught herein. A "harvesting" interface can be provided on, for example, a tablet computer 70. Such a user interface can utilize one or more "native" application programs provided on or for the computer (e.g., conventional operating system and browser software) to support agent tasking. For example, conventional browser software can be used to find and clip words and phrases from sites of interest and add this material to a semantic network - e.g., by adding nodes to an existing network data structure. (While the device illustrated in FIG. 5 is a tablet computer 70, any suitable computing system such as a client PC, laptop, mobile device, PDA or related computing system may be used.) A conventional

application, such as a browser or a word-processing program capable of supporting graphics, may be used to display the results of the agent operations. Designer 80 is shown as an optional supplemental application software tool to demonstrate the extensibility of the agent system to allowing use of other software, as well, for agent design, network design, and updating/alteration of either of them. User model 90 indicates that the net impact of a collection of agents is to capture user-interests and intentions in such a way that the need for recursive user-interactions and querying is greatly eliminated, the agents becoming a mechanism that provides the blueprint for services the user desires.

Core agent tasks 100 operate over the internet content and services, and include the aforementioned harvesting, connecting, reporting, etc. Content/service application programming interfaces (APIs) 110 and crawlers act as interfaces to the Internet.

An individual agent, or a collection of agents (each having a dedicated function) may function, and be thought of, as a user's assistant(s). Such assistants may serve multiple users. Indeed, when appropriate to their function, the assistant also may be monetized by multiple users. For instance, consider a first user that may not be a developer but may still like to create things using technology. The first user may create an assistant using the agents framework disclosed herein. To do so, the user may launch an assistant designer application, which performs the dual roles of designing individual agents and assembling groups of agents around specific tasks, as well as allocating resources to each type of agent. (Where allocating resources to an agent means defining those data the agent may take as inputs and supply as output.) The user typically also may be asked to specify the purpose of his or her assistant – i.e., its function.

The designer application may be set up to include certain agents in the assistant by default, by choice or both. Default agents may include, for example, a connecting agent to synthesize new connections across a domain (see below); a harvesting agent to extracts terms from retrieved content nodes, such as a relevant abstract; and a reporting agent to provide a status on the information available for a particular context (such as the identification of a new restaurant, if that is the assistant's purpose).

A "connecting agent" is simply the term used to reference an agent that can augment a concept node from a semantic network with further concepts that are identified from other domains. That is, it establishes a cross-domain connection, or

bridge, a link. The connection may be one that is ascertained from a reference corpus (e.g., one or more websites, documents, etc.) or a reference semantic lexicon (e.g., graphical lexicons such as WordNet, thesauri, dictionaries, etc.; WordNet is a lexical database for the English language, created and maintained at the Cognitive Science Laboratory of Princeton University). The agent may operate by referencing the semantic network and receiving a seed concept, following which it may then augment the seed concept with semantically related terms that are of relevance from the reference corpus/lexicon. The agent may analyze the concept node by identifying terms that comprise the concept node, and analyze related domains for literal and semantically related matches.

While both a connecting agent and a harvesting agent may augment a node of the semantic network, a connecting agent does so by incorporation of a link or attachment to another node, whereas a harvesting agent will, based on one or more nodes in the semantic network, harvest information from a reference corpora and augment the one or more nodes by attaching the harvesting information. The augmenting information to be attached by a harvesting agent is typically a finite amount of information such as typically would be characterized as a "snippet of data," "chunk of content," "paragraph" abstract," etc.

Continuing with restaurant identification as an example context, using an agent designer application (software tool) or other selection mechanism, the user may select from among various pre-existing agents to include in an assistant. These may include, for example, a review collection agent whose function is to retrieve restaurant reviews from multiple sources using terms acquired from a context within an individual's semantic network; a web search agent that may search for terms from a context within an individual's semantic network and a text message mining agent that may retrieve information from messaging accessible within an individual's semantic network.

As a specific example, a restaurant agent may be configured to keep the user informed of all restaurants in the user's city that have been reviewed in some list of publications, web sites, etc. in the last six months, that serve certain types of food, and that were rated at least three stars our of five. When the agent identifies such a restaurant, it may add that restaurants to the user's semantic network at a node we may call "new restaurants to try." A reporting agent, noting the addition of a restaurant to the

network, may send a message (text, e-mail, or other service) or simply add to a "new reports" queue the user checks from time to time, the fact of the restaurant's addition to the network. The user may try the restaurant and decide she does not like it. Another agent may note that the restaurant was visited once by the user (e.g., by monitoring the user's credit card statement) and also note that a year has gone by and the user has not returned to the restaurant, resulting in the agent either automatically purging the restaurant from the user's list of restaurants she likes, or the agent may ask the user whether to retain or purge the restaurant.

These are but simple examples of the way agents may interact with a user, with a semantic network and with each other, to act as a person's assistants, relieving the person of the otherwise time-consuming tasks the agents perform.

In some embodiments, a constructed assistant may be monetized. For instance, a licensing mechanism may be set up to meter the re-use of agents, charge the re-user and credit the original designer some amount in connection with the re-use. For example, a fixed amount, such as $0.02, may be charged each time an individual tasks a specific assistant to issue a status report. This charge may be partially credited to the constructor of the assistant and partially credited to the host system supporting the agent ecosystem. In some embodiments, the amount charged may be offset or be eliminated by the use of other revenue sources, such as advertising, for example.

To facilitate the licensing and re-use of such agents, the ecosystem preferably includes an agent-naming module that allows a creator to name the agents and assistants he/she creates, and to label/designate them as publicly available (if so desired). (For example, a public register of such agents may be maintained for this purpose and a creator may then register her agent. Registration typically would involve identifying the creator, the name of the agent, its function, its inputs and outputs, and any relevant licensing terms such as the charge for using/copying/modifying the agent.) In addition to a second user being licensed to re-use an agent or assistant, a second user also may be granted permission (again, possibly for a fee), to clone and modify the assistant, such as by altering the selection of agents as deemed advantageous by the second user. For example, a second user may add a location-filtering agent to a restaurant assistant which had included only an agent that identifies restaurants by type of cuisine. In some embodiments, the second user may assign a charge per reporting task performed by the

newly modified assistant and register it also to be publicly available on that basis, perhaps with the original designer sharing in the resulting revenue.

Thus it will be seen that there has been shown a new method and system for combining user-directed semantic networking with computerized agents, typically large numbers of simple agents, wherein both users and agents interact with a semantic network without users having to control agents expressly and directly.

The above-described embodiments of the present invention can be implemented in any of numerous ways. For example, the embodiments may be implemented using hardware, software or a combination thereof. When an embodiment or element of an embodiment is implemented in software, the software code can be executed on any suitable processor or collection of processors, whether provided in a single computer or distributed among multiple computers. It should be appreciated that any component or collection of components that perform the functions described above can be generically considered as one or more controllers that control the above-discussed functions. The one or more controllers can be implemented in numerous ways, such as with dedicated hardware, or with general purpose hardware (e.g., one or more processors) that is programmed using microcode or software to perform the functions recited above.

In this respect, it should be appreciated that one implementation of various embodiments of the present invention comprises at least one tangible, non-transitory computer-readable storage medium (e.g., a computer memory, a floppy disk, a compact disk, and optical disk, a magnetic tape, a flash memory, circuit configurations in Field Programmable Gate Arrays or other semiconductor devices, etc.) encoded with one or more computer programs (i.e., a plurality of instructions) that, when executed on one or more computers or other processors, performs the above-discussed functions of various embodiments of the present invention and elements thereof. The computer-readable storage medium can be transportable such that the program(s) stored thereon can be loaded onto any computer resource to implement various aspects of the present invention discussed herein. In addition, it should be appreciated that the reference to a computer program which, when executed, performs the above-discussed functions, is not limited to an application program running on a host computer. Rather, the term computer program is used herein in a generic sense to reference any type of computer code (e.g., software or microcode) that can be employed to program a processor to implement the above-

discussed aspects of the present invention. The semantic network element of the embodiments discussed herein may comprise one or more data structures in one or more non-transitory computer-readable storage media, which may be the same or different storage media encoded with the above-noted one or more computer programs.

End-use applications may occur on a client PC, laptop, tablet, mobile device, PDA or related computing system. Further, some embodiments may leverage native applications such as web browsers or apps on any of these computing systems.

FIG. 6 shows, schematically, an illustrative computer 1100 on which various inventive aspects of the present disclosure may be implemented. The computer 1100 includes a processor or processing unit 1101 and a memory 1102 that may include volatile and/or non-volatile memory. The computer 1100 may also include storage 1105 (e.g., one or more disk drives) in addition to the system memory 1102. The memory 1102 and/or storage 1105 may store one or more computer-executable instructions to program the processing unit 1101 to perform any of the functions described herein. The storage 1105 may optionally also store one or more data sets as needed.

References herein to a computer can include any device having a programmed processor, including a rack-mounted computer, a desktop computer, a laptop computer, a tablet computer or any of numerous devices that may not generally be regarded as a computer, which include a programmed processor.

The exemplary computer 1100 may have one or more input devices and/or output devices, such as devices 1106 and 1107 illustrated in FIG. 6. These devices may be used, among other things, to present a user interface. Examples of output devices that can be used to provide a user interface include printers or display screens for visual presentation of output and speakers or other sound generating devices for audible presentation of output. Examples of input devices that can be used for a user interface include keyboards, and pointing devices, such as mice, touch pads, and digitizing tablets. As another example, a computer may receive input information through speech recognition or in other audible format.

As shown in FIG. 6, the computer 1100 may also comprise one or more network interfaces (e.g., the network interface 1110) to enable communication via various networks (e.g., the network 1120). Examples of networks include a local area network or a wide area network, such as an enterprise network or the Internet. Such networks may

be based on any suitable technology and may operate according to any suitable protocol and may include wireless networks, wired networks or fiber optic networks.

REFERENCE A: METHOD AND APPARATUS FOR PREFERENCE GUIDED DATA EXPLORATION

Data exploration systems, such as search engines and database management systems, manage enormous volumes of information. As a result, locating information of interest to a user in response to a search query (e.g., in the form of a set of keywords) presents challenges.

Conventional approaches to search often shift the burden of finding the information of interest to the user. For example, all potentially-relevant results may be presented to the user in response to a search query. Subsequently, the user has to manually explore and/or rank these results in order to find the information of greatest interest. When the number of potentially-relevant results is large, which is often the case, the user may be overwhelmed and may fail to locate the information for which he is looking.

One conventional technique for addressing this problem is to integrate a user's preferences into the search process. By presenting search results in accordance with the user's preferences, the user may be helped to find the information he seeks. However, conventional approaches to specifying user preferences severely limit the ways in which user preferences may be specified.

Consider, for example, a data exploration model adopted by many search services and illustrated in FIG. 7. Query interface 12 is used to collect query predicates in the form of keywords and/or attribute values (e.g., "used Toyota" with price in [$2000-$5000]). Query results are then sorted (14) on the values of one or more attributes (e.g., order by Price then by Rating) in a major sort/minor sort fashion. The user then scans (16) through the sorted query answers to locate items of interest, refines query predicates, and repeats the exploration cycle (18). This "Query, Sort, then Scan" model limits the flexibility of preference specification and imposes rigid data exploration schemes as highlighted in the following example.

Example 1

Amy is searching online catalogs for a camera to buy. Amy is looking for a reasonably-priced camera, whose color is preferably silver and less preferably black or gray, and whose reviews contain the keywords "High Quality." Amy is a money saver, so her primary concern is satisfying her Price preferences followed by her Color and Reviews preferences.

The data exploration model of FIG. 7 allows Amy to sort results in ascending price order. Amy then needs to scan through the results comparing colors and inspecting reviews to find the camera that she wants. The path followed by Amy to explore search results is mainly dictated by her price preference, while other preferences are incorporated in the exploration task through Amy's effort, which can limit the possibility of finding items that closely match her requirements.

Conventional approaches to specifying user preferences suffer from a number of other drawbacks in addition to not simultaneously supporting different types of preferences. For example, preference specifications may be inconsistent with one another. A typical example is having cycles in preferences among first-order preferences (preferences among attributes of items such as preferring one car to another car based on the price or on brand), which implies non-transitivity of preferences. For instance, a user may indicate that a Honda is preferred to a Toyota, Toyota is preferred to a Nissan and a Nissan is preferred to a Honda. Even when first-order preferences are consistent, second order preferences (preferences among the first order preferences such as brand preferences are more important than price preferences) can result in further problems. For example, prioritized composition of a set of partial orders does not generally maintain the transitivity property in the resulting order. Conventional systems for data exploration are unable to rank search results when preference specifications may be inconsistent.

Inadequate incorporation of preferences in conventional data exploration systems is due at least partly to the inability of these systems to integrate different types of preferences. For instance, in the above-described example, preferences include an ordering on all prices (a "total order" preference), an ordering between some colors (a "partial order" preference), a Boolean predicate from the presence of the words "High

Quality" in the reviews, and an indication that price is more important than the other preferences.

Another situation in which it may be useful to specify different types of preferences may be a situation in which a user may have precise preferences for information in one domain because the user may possess a large amount of knowledge about the domain. Such precise preferences may be specified, for example, in the form of one or more scoring functions. However, the same user may have imprecise preferences for information in another domain because the user may not posses a large amount of knowledge about the other domain. In this case, preferences may be specified, for example, in the form of one or more partial orders on attribute values. There are many instances in which the user may need to specify both types of preferences (i.e., using a scoring function and using a partial order) as shown in Example 2 below.

Example 2

Alice is searching for a car to buy. Alice has specific preferences regarding sport cars, and more relaxed preferences regarding SUVs. Alice supplies the data exploration system with a scoring function to rank sport cars, and a set of partial orders encoding SUVs preferences. Alice expects reported results to be ranked according to her preferences.

A data exploration system capable of integrating different preference types and ranking search results in response to a user query, in accordance with user-specified preferences, may address some of the above-discussed drawbacks of conventional approaches. However, not every embodiment addresses every one of these drawbacks, and some embodiments may not address any of them. As such, it should be appreciated that embodiments of the invention are not limited to addressing all or any of the above-discussed drawbacks of these conventional approaches.

Accordingly, in some embodiments, a preference language is provided for specifying different types of user preferences among items. A data exploration system may assist a user to specify preferences using the preference language. The specified preferences may be used to construct a general preference model that, in turn, may be used to produce a ranking of items in accordance with any user preferences.

Items may be any suitable items about which a user may express preferences. In some instances, an item may be any item that may be manufactured, sold and/or purchased. For example, an item may be a car or an airplane ticket—a user (e.g., a consumer) may have preferences for one car over another car and/or may prefer one airplane ticket to another airplane ticket. In some instances, an item may comprise information. Users may prefer one item over another item based at least in part on the information that these items contain. For example, when searching for content (e.g., movie, music, images, webpages, text, sound, etc.) a user may prefer some content to other content. For instance, a user may prefer to see a webpage that contains information related to cars over a webpage that contains information related to bicycles.

An item may comprise, or have associated with it, one or more attributes. An attribute of an item may be related to the item and may be a characteristic of the item. An attribute of an item may be a characteristic descriptive of the item. For instance, if an item is an item that may be purchased, an attribute of the item may be a price related to the item. An attribute of an item may be a characteristic that may identify the item. For example, a characteristic of an item may be an identifier (e.g., name, serial number, or model number) of the item.

Attributes may be numerical attributes and may be categorical attributes. Numerical attributes may comprise one or more values. For instance a numerical attribute may comprise a single number (e.g., 5) or a range of numbers (e.g., 1-1000). Categorical attributes may also comprise one or more values. For instance, a categorical value for the category "Color" may comprise a single color (e.g., Red) or a set of colors (e.g., {"Red", "Green"}). Though, it should be recognized that attribute values are not limited to being numbers and/or categories and may be any of numerous other types of values. For instance, values may comprise alphabetic and alphanumeric strings.

An item may be represented by one or more tuples comprising values for one or more attributes associated with the item. In some cases, a tuple representing an item may comprise a value for each attribute associated with the item. In other cases, a tuple representing an item may comprise a value for only a portion of the attributes associated with the item.

FIG. 8 shows an illustrative example of a set of items, each item being represented by a tuple comprising values for the attributes of the items. In the illustrative

example of FIG. 8, each item is a car and is associated with six attributes: "ID," "Make," "Model," "Color," "Price," and "Deposit." Though in this example all items share the same attributes, this is not a limitation of the present invention as different items may have different attributes from one another and some attributes may have unknown values. Each item is represented by a tuple (i.e., a set) of attribute values. Accordingly, the first item has characteristics indicated by the first set of attribute values. For instance, the first item is represented by the tuple in the first row of the table shown in FIG. 8. As illustrated, this first item is an $1800 Red Honda Civic identified by identifier "t_1". A deposit of $500 may be required to purchase this car.

A user may express preferences for one item over another item in a set of items. User preferences may be of any suitable type and may be first-order user preferences, second-order user preferences, and even further-order preferences.

First-order preferences are preferences associated with attributes of items. First-order preferences may be based on values of attributes of items. For example, a first-order preference may express a preference for an item over another item based on values of one more attribute of the two items. For instance, a first-order preference may indicate an item with a lower price (value of the attribute "price") is preferred to an item with a higher price. As another example, a first-order preference may indicate that a red (value of the attribute "color") item (e.g., car) is preferred to a blue item.

Second-order preferences are preferences across first-order preferences. Second-order preferences may indicate which first-order preferences are more important to a user. For example, first-order preference A may be based on values of one attribute (e.g., "price") while first-order preference B may be based on values of another attribute (e.g., "color"). A second-order preference may indicate that first-order preference B is preferred to first-order preference A (i.e., color may be more important than price).

There may be many different types of first-order and second-order preferences and these types of preferences, along with other aspects of first-order and second-order preferences are discussed in greater detail below in Sections II and III, respectively.

The data exploration system may be any system for exploring data, information or knowledge. The data exploration system may allow one or more users to query the system. For instance, a data exploration system may be a search engine such as an Internet search engine or a domain-specific search engine (e.g., a search engine created

to search a particular information domain such as a company's or institution's intranet, or a specific subject-matter information repository). In another example, a data exploration system may be a database system that may allow user queries.

A query input by a user into a data exploration system may be any of numerous types of queries. For instance, a query may comprise one or more keywords indicating what the user is seeking. In some cases, a query may comprise user preferences. Though, it should be appreciated that user preferences may be specified separately and/or independently from any user query. For instance, a user may specify preferences that may apply to multiple user queries. The specified preferences may comprise preferences of any suitable type such as first-order and/or second-order user preferences.

Regardless of the types of preferences that a user may wish to specify, a data exploration system may assist a user to specify preferences. A data exploration system may assist a user to specify preferences using the preference language, for example. Some example approaches to how a data exploration system may assist a user to specify preferences are described in greater detail in Sections I and VI, below.

After user-specified preferences are obtained (e.g., from a user-specified query or any other suitable source), a preference model may be constructed from these preferences. The preference model may be constructed from different types of preferences and may be constructed from first-order preferences of different types and/or from second-order preferences of different types.

A preference model may be represented by a data structure encoding the preference

model. The data structure may comprise any data necessary for representing the preference model and, for example, may comprise any parameters associated with the preference model.

A data structure encoding a preference model may be stored on any tangible computer-readable storage medium. The computer-readable storage medium may be any suitable computer-readable storage medium and may be accessed by any physical computing device that may use the preference model encoded by the data structure.

In some embodiments, the preference model may be a graph-based preference model and the data structure encoding the preference model may encode a graph, termed a preference graph, characterizing the graph-based preference model. The preference

graph may comprise a set of nodes (vertices) and a set of edges connecting nodes in the set of nodes. The edges may be directed edges or may be undirected edges. Accordingly, the data structure encoding the preference graph may encode the preference graph by encoding the graph's vertices and edges. Any of numerous data structures for encoding graphs, as are known in the art, may be used to encode the preference graph, as the invention is not limited in this respect.

In some embodiments, nodes of the graph may be associated with items. For instance, a node in the graph may be associated with a tuple that, in turn, represents an item. The graph may represent items that are related with one or more keywords in a query. For instance, a set of items may be selected in response to a user-provided query.

A first-order preference for one item over another item may be represented as an edge in the graph, with the edge connecting nodes associated with the tuples associated with the two items. A weight may be associated to each node in the graph to provide an indication of a degree of preference for one of the nodes terminating the edge. The weight may be computed based on first-order and/or second preferences. Aspects of a graph-based preference model, including how such a preference model may be constructed from user-specified preferences, are described in greater detail in Section IV, below.

The preference model may be used to obtain a ranking of items in a set of items. For instance, a graph-based preference model may be used to construct such a ranking. A graph-based preference model may be used to construct such a ranking in any of numerous ways. For instance, a complete directed graph may be obtained from the graph-based preference model and a ranking of items may be obtained based on the completed directed graph. As another example, a Markov-chain based algorithm may be applied to the graph-based preference model to obtain a ranking of items. These and other approaches to obtaining a ranking of items in a set of items are described in greater detail in Section V, below.

It should be appreciated that though a preference graph may be a convenient abstraction, which is helpful for reasoning about user preferences, in practice, a preference graph may be implemented on a physical system via a data structure that may encode the preference graph. Similarly, many constructs described below (e.g., relations, scopes, scope comparators, and etc.) are convenient abstractions used in various fields

such as computer science, but each construct may be realized, in practice, by a data structure representing data characterizing the construct and/or processor-executable instructions for carrying out functions associated with the construct. Such data structures and processor-executable instructions may be encoded on any suitable tangible compute-readable storage medium.

Accordingly, for ease of reading, every reference to a construct (e.g., a graph, a relation, scope, scope comparator, etc.) is a reference to a data structure encoding the construct and/or processor-executable instructions that when executed by a processor perform functions associated with the construct, since explicitly referring to such data structures and processor-executable instructions for every reference to a construct is tedious.

It should also be appreciated that the above-described embodiments of the present invention, can be implemented in any of numerous ways. For example, the embodiments may be implemented using hardware, software, or a combination thereof. When implemented in software, the software code may be embodied as stored program instructions that may be executed on any suitable processor or collection of processors (e.g., a microprocessor or microprocessors), whether provided in a single computer or distributed among multiple computers.

Software modules comprising program instructions may be provided to perform any of numerous of tasks in accordance with some embodiments. For example, one or multiple software modules for constructing a preference model may be provided. As another example, software modules for obtaining a ranking for a set of items based on (a data structure representing) the preference model may be provided. As another example, software modules comprising instructions for implementing any of numerous functions associated with a data exploration system may be provided. Though, it should be recognized that the above examples are not limiting and software modules may be provided to perform any functions in addition to or instead of the above examples.

I. Design Goals

In some embodiments, a data exploration system that utilizes user preference may reflect some or all of the following design goals:

- Guidance: The system may assist users to formulate their preferences. The system may support interactive preference management. For instance, the system may provide users with information to help users specify and/or modify preferences. As a specific example, the system may provide users with information about how to modify their preferences to widen or narrow the scope of their search. As another specific example, the system may provide users with information about how to modify their preferences such that the ranking of items presented to a user is modified. Though, these are only examples and the system may aid the user to formulate their preferences in any of numerous ways as described in greater detail below, in Section VI.
- Flexibility: Specification of different types of preferences may be supported for arbitrary subsets of items, sometimes referred to as "contexts." The system may accept natural descriptions of preferences and map these descriptions into preference constructs.
- Provenance: The system may be able to provide justification of how search results are generated and ranked by relating generated results to input preferences.

FIG. 9 illustrates flowchart for an example process of modeling preferences that reflects the above-mentioned design goals. As illustrated in FIG. 9, the data exploration system may be a system that may receive a query from one or more users. For instance, the system may be a database system or a search engine and the query may comprise one or more keywords.

Toward the guidance goal, the system may assist a user to specify preferences. In some embodiments, such support may be based on pre-computed summaries in the form of facets that may be used for guiding data exploration. Each facet may be associated with a number that may provide the user with an estimate on the expected number of results. Accordingly, facets may allow a user to get a quick and dirty view of the underlying set of items and/or domain, and how search results may be affected by tuning preferences

For example, the system may comprise a memory configured to store a plurality of tuples (recall that each tuple comprises one or more values for one or more attributes)

and may receive a range of desired values for an attribute from a user. In response the system may output an integer indicative of a number of tuples comprising a value for the attribute such that the value is in the range of values. As a specific example, for a categorical attribute, a facet may comprise a possible attribute value (e.g., 'Color = Red'); while for a numerical attribute, a facet may comprise a range of possible values (e.g., 'Price in [$1000-$5000]'). Moreover, the user may be able to define custom facets as Boolean conditions over multiple attributes (e.g., 'Color=Red AND price < $5000'). The system may associate a number to each of these facets, the number indicating an expected number of tuples consistent with these facets.

Toward the flexibility goal, the system may adopt the concept of contextualized preferences, where a user can assign different preference specifications to different subsets (contexts) of items. A user may define a context by using predetermined facets or by defining custom facets. As discussed below in Sections II and III, the user has the flexibility of expressing first-order and second-order preferences within and across contexts. Contextualized preferences may also part of a user's profile, which may be ascertained by any of the techniques disclosed herein as well as those disclosed in U.S. Non-Provisional Application Serial No. 12/555,293, filed September 08, 2009, and titled Synthesizing Messaging Using Context Provided By Consumers, which is hereby incorporated by reference. This way, they may be loaded, saved, and/or refined upon the user's request.

Toward the provenance goal, the data exploration system illustrated in FIG. 9 may maintain information regarding which preferences among the input preferences, affect the relative order of each pair of items in the final results ranking. This feature may be useful for the analysis and refinement of preferences in different scenarios. Examples include finding preference constructs that have dominating effect on results' ranking, decreasing/increasing the influence of some preference constructs, and understanding the effect of removing a certain preference construct.

Additional ways in which a data exploration system may assist a user to input preferences are discussed below in Section VI.

II. First-Order Preferences

In some embodiments, the preference language may be based on capturing pairwise preferences on different granularity levels. An items' description may follow a relational model, where each item may be represented as a tuple. Preferences may be cast against a relation R with a known schema.

Our first construct is used to define a context for expressing first-order preferences.

Definition 1 [Scope]: *A scope R_i is an arbitrary non-empty subset of tuples in R.*

A scope defines a Boolean membership property that restricts the space of all possible tuples to a subset of tuples that are interesting for building preference relations. Such a membership property may be defined using a SQL query posed against R. For example, FIG. 10 shows six different scopes $R_1 \ldots R_6$ in the relation "Car" illustrated in FIG. 8, where scopes are defined using SQL queries. Though, it should be recognized that such a membership property may be defined using any of numerous other ways. As one example, a database query language other than SQL may be used to define such a membership property. As another example, the membership property may be defined using a set of variables and a database language may not be needed.

As shown in the illustrative diagram of FIG. 10, scopes may intersect. Thus, a tuple in the relation R may belong to zero, one or two or more scopes. Tuples that do not belong to any scopes may be non-interesting with respect to a preference specification. Thus, for clarity, all subsequent discussion is with respect to tuples that belong to at least one scope.

Definition 2 [Scope Comparator]: *Let R_i and R_j be two scopes in R. The scope comparator $f_{i,j}$ is a function that takes a pair of distinct tuples (one is from R_i and the other is from R_j), and returns a first value such as 1 (e.g., if the tuple from R_i is preferred), a second value such as -1 (e.g., the tuple from R_j is preferred), or a null value "⊥" (e.g., if there is no preference).*

A scope comparator is a preference language construct for defining first-order preferences. In some instances, the scope comparator may be user-defined. Though, in other instances, a scope comparator may be defined, automatically, by a computer. Still,

in other embodiments a scope comparator may be defined by a combination of manual and automatic techniques.

A generic interface to a scope comparator may accept two tuples and return either a preference of one tuple over the other, or no preference can be made. Whenever a tuple t_i is preferred to a tuple t_j, we say that t_i dominates t_j, denoted as $t_i \succ t_j$.

FIG. 11 shows illustrates 5 different scope comparators defined on the scopes shown in FIG. 10. In FIG. 11, the scope comparators $f_{3,4}$ and $f_{1,5}$ are unconditional (i.e., they produce first-order preferences without testing any conditions beyond the conditions captured by scope definition). On the other hand, the scope comparators $f_{1,2}$, $f_{5,6}$, $f_{6,2}$ are conditional (i.e., they produce preference relations conditioned on some logic).

Algorithm 1 Score-based Preferences

```
SCORE-PREFS (t_i: tuple, t_j: tuple, S: scoring function)
1  if (S(t_i) > S(t_j))
2     then return 1
3     else if (S(t_j) > S(t_i))
4             then return -1
5     else return ⊥
```

Conditional scope comparators allow defining composite preferences that span multiple attributes given in scope definition and/or comparator logic (e.g., $f_{6,2}$ defines a composite preference on Price and Make attributes).

The generality of scope definitions and preference comparators allow encoding different types of preferences, with different semantics. In the following we give templates for encoding different types of preferences using the above-described language constructs.

Template 1 [Score-based Preferences]. *Preferences are defined using a scoring function S, where tuples achieving better scores are preferred. Without loss of generality and without limitation, assume that higher scores are better, then score-based preferences can be specified using the template given by Algorithm 1.*

A total order on a scope R_i (which can be the whole relation R) may be encoded by defining a comparator $f_{i,i}$, using the template in Algorithm 1, where $f_{i,i}$ operates on pairs of distinct tuples belonging to R_i.

Template 2 [Partial Order Preferences]. *For an attribute x, let P_x be a partial order defined on the domain of x. The partial order can be expressed as a set $P_x = \{(v_i > v_j)\}$ for values v_i and v_j in the domain of x, such that P_x is:*

- *irreflexive (i.e., $(v_i > v_i) \notin P_x$).*
- *asymmetric (i.e., $(v_i > v_j) \in P_x \Rightarrow (v_j > v_i) \notin P_x$).*
- *transitive (i.e., $\{(v_i > v_j), (v_j > v_k)\} \subseteq P_x \Rightarrow (v_i > v_k) \in P_x$).*

Partial order-based preferences may be encoded using the template given by Algorithm 2.

Template 3 [Skyline Preferences]. Given a set of attributes A, a tuple t_i is preferred to tuple t_j if there exists a non-empty subset $X \subseteq A$, *where* $\forall x \in X : t_i.x$ is preferred to $t_{j.x}$, while for any other attribute $x' \in A - X$, no preference can be made between $t_i.x'$ *and* $t_j.x'$. Skyline preferences may be encoded as shown in the template given by *Algorithm 3*.

Algorithm 2 Partial Order Preferences

PARTIAL ORDER-PREFS (t_i:tuple , t_j: tuple, P_x: partial order on attribute x)

```
1  if ((t_i.x > t_j.x) ∈ P_x)
2     then return 1
3     else if ((t_j.x > t_i.x) ∈ P_x)
4             then return -1
5     else return ⊥
```

Algorithm 3 Skyline Preferences

SKYLINE-PREFS (t_i: tuple, t_j:tuple, A: subset of attributes)

```
1  p_i ← 0
2  p_j ← 0
3  for all x ∈ A
4       do
5          if (t_i.x is preferred to t_j.x)
6            then p_i ← p_i + 1
7            else if (t_j.x is preferred to t_i.x)
8                    then p_j ← p_j + 1
9          if (p_i > 0 AND p_j > 0)
10           then return ⊥
11 if (p_i > 0)
12    then return 1
13    else if (p_j > 0)
14            then return -1
```

Template 4 [Conjoint Analysis Preferences]. *Given a set of attributes A, conjoint analysis encodes preferences among attribute values in A when taken conjointly. This can be expressed as a function C_A that maps each combination of values in A to a unique rank. The function C_A is partial on the domains of all possible combinations of values in A. Hence, there can be combinations of values in A that are not mapped to ranks under C_A. Conjoint analysis preferences based on C_A may be expressed using the template given by Algorithm 4.*

The next example is an example for specifying and managing conjoint analysis preferences.

Example 3:

Alice's preferences regarding cars may be expressed conjointly over the attribute pairs (Make, Color), and (Make, Price), as shown in FIG. 12. The value in each cell is the rank assigned to each combination of attribute values.

Conjoint analysis may be based on an additive utility model in which ranks, assigned to combinations of attribute values, may be used to derive a utility (part-worth) of each attribute value. The objective is that the utility summation of attribute values reconstructs the given ranking. In FIG. 12, for example, 'Honda' is assigned utility value 40, while 'Red' is assigned utility value 50. Hence, the score of 'Honda, Red' is 90, which matches the assigned rank 1 in the given Make-Color preferences. Utility values may be computed using regression. For instance, they may be computed using linear regression. Note the mapping between combinations of attribute values and ranks is modeled.

III. Second-Order Preferences

Our main language construct for defining second-order preferences is a preferences order (POrder), defined as follows:

Definition 3 [POrder]: *given a set of scope comparators F, a POrder is a permutation of comparators in F.*

A POrder represents an ordering of scope comparators based on their relative importance. A POrder may quantify the strength of different first-order preferences based on the semantics of second-order preferences, as discussed in greater detail below in Section IV.

Definition 4 [POrder Projection]: *Let A be a POrder defined on the set of comparators F. For* $F' \subseteq F$ *we denote with* $(\Pi_{F'} A)$ *a total order of comparators in* F' *ordered according to A. It follows that* $\Pi_F A = A$.

Algorithm 4 Conjoint Analysis Preferences

CONJOINT ANALYSIS-PREFS (t_i: tuple, t_j:tuple, A: subset of attributes, C_A: conjoint analysis map)

```
1  if (C_A({t_i.x : x ∈ A}) is undefined
   OR C_A({t_j.x : x ∈ A}) is undefined)
2     then return ⊥
3     else if (C_A({t_i.x : x ∈ A}) < C_A({t_j.x : x ∈ A})
4           then return 1
5     else return -1
```

For example, for the POrder $A = (f_1, f_2, f_3)$, and the subset of comparators $F' = \{f_1, f_3\}$, we have $\Pi_{F'} A = (f_1, f_3)$.

Given a POrder projection A', we say that $(t_i \succ t_j)$ under A' if for a scope comparator $f_a \in A'$, we have $f_a(t_i, t_j) = 1$, and there is no other scope comparator $f_b \in A'$, where $f_b \succ f_a$according to A', and $f_b(t_i, t_j) = -1$.

Different types second-order preferences may be encoded using POrders.

- **Prioritized Preference Composition.** In this case, second-order preferences are defined as a total order of comparators $O = (f_1 \succ f_2 \succ \cdots \succ f_m)$, which expresses the requirement that the first-order preferences corresponding to f_i are more important than the first-order preferences corresponding to f_{i+1}. Prioritized composition of preferences is formulated as a single POrder with the same comparators order given by O.
- **Partially Ordered Preferences**. A partial order PO on the set of scope comparators may encode partial information on the relative importance of different scope comparators. Let Ω be a set of comparator orderings consistent with PO, where an ordering ω is consistent with PO if the relative order of any two scope comparators in ω does not contradict with PO. The set Ω is called the set of linear extensions of PO. For example, FIG. 13 shows a partial order defined on four comparators and the corresponding set of linear extensions. The set of linear extensions may be obtained using a simple recursive algorithm on the PO graph. Partially-ordered preferences may be formulated as the set of POrders given by Ω.

- **Pairwise Preferences:** A set $PW = \{(f_i \succ f_j)\}$ of pairwise second-order preferences on scope comparators. The pairwise second-order preference $(f_i \succ f_j)$ expresses the requirement that the first-order preferences corresponding to f_i are more important than the first-order preferences corresponding to f_j. Pairwise second-order preferences *PW* may be formulated as the set of POrders $\{(f_i, f_j) : (f_i \succ f_j) \in PW\}$.
- **Pareto Preference Composition:** The importance of all scope comparators is equal. The first-order preference $(t_i \succ t_j)$ is produced if and only if at least one scope comparator states that $(t_i \succ t_j)$, and no other scope comparator states that $(t_j \succ t_i)$. Pareto preference composition is formulated as a set of singleton POrders, where each POrder is composed of a single comparator.
- **Preferences Aggregation:** The scope comparators act as voters on preference relations. The first-order preference $(t_i \succ t_j)$ is produced if and only if at least one scope comparator states that $(t_i \succ t_j)$. Preferences aggregation may be formulated as a set of singleton POrders, where each POrder may be composed of a single comparator.

IV. Compilation

Given a set of scopes and scope comparators, a graph-based representation of the preferences, termed a preference graph, may be obtained. In this Section, an algorithm for "compiling" the given set of scope and scope comparators (first-order preferences) is described. A preference graph may be formally defined as follows:

Definition 5 [Preference Graph]: *A directed graph (V,E), where V is the set of tuples in R and an edge* $e_{i,j} \in E$ *connects tuple* t_i *to tuple* t_j *if there exists at least one comparator applicable to* (t_i, t_j) *and returning 1, or applicable to* (t_j, t_i) *and returning -1. The label of edge* $e_{i,j}$*, denoted* $l(e_{i,j})$ *is the set of comparators inducing preference of* t_i *over* t_j.

The compilation algorithm is described in Algorithm 5. The algorithm constructs the set of vertices also termed nodes of the preference graph using the union of tuples involved in all input scopes. In other words, each node in the preference graph is associated with a tuple. Accordingly, each node in the preference graph may represent an item. For each pair of distinct tuples, the set of applicable scope comparators may be found and used to compute graph edges and their labels. Accordingly, an edge in the

preference graph may correspond to a first-order preference, which may indicate a user preference for one of the two items represented by the nodes terminating the edge.

Edges of the preference graph may be directed edges and may be directed to the node associated with a preferred data item as indicated by the first-order preference associated with the edge. Though, in some embodiments, edges may be undirected and an indication of which of nodes terminating the edge is preferred may be provided differently. For instance, such an indication may be provided by using a signed weight, with a negative weight indicating a preference for one node and a positive weight indicating a preference for the other node.

FIG. 14 illustrates example for the output of the compilation algorithm. In particular, FIG. 14 shows the preference graph obtained from the set of scope comparators $\{f_{1,2}, f_{3,4}, f_{5,6}, f_{6,2}, f_{1,5}\}$ described with reference to FIG. 10. Each edge is labeled with a set of supporting comparators. For example, for the edge $e_{2,6}$, we have $l(e_{2,6}) = \{f_{1,2}, f_{6,2}\}$, since the tuple t_2 is preferred over the tuple t_6 according to the scope comparators $f_{1,2}$ and $f_{6,2}$.

Since scopes may intersect and arbitrary scope comparator logic may be allowed, the induced preference graph may be a cyclic graph. For example, in FIG. a $t_1 - t_6$ cycle exists since t_1 is preferred over t_6 according to $f_{6,2}$, while t_6 is preferred over t_1 according to $f_{1,2}$. Construction of a preference graph according to Algorithm 5 does not guarantee transitivity of graph edges. For example, in FIG. 14, the existence of the edges $e_{2,6}$ and $e_{6,1}$ does not imply the existence of the edge $e_{2,1}$.

Algorithm 5 Preferences Compilation

```
COMPILE-PREFS (S: a set of scopes, F: a set of comparators)
1  V ← ⋃_{s_i ∈ S} {t : t ∈ s_i}  {find the union of all scopes}
2  E ← {}  {initialize set of graph edges as empty}
3  for all (t_i, t_j) ∈ (V × V): t_i ≠ t_j
4      do
5          for all f ∈ F
6              do
7                  if (f is applicable to (t_i, t_j))
8                    then
9                          p ← f(t_i, t_j)
10                         if (p = 1)
11                           then
12                                 e_{i,j} ← 1
13                                 append f to l(e_{i,j})
14                                 if (e_{i,j} ∉ E)
15                                     then add e_{i,j} to E
16                         else if (p = −1)
17                                 then
18                                       e_{j,i} ← 1
19                                       append f to l(e_{j,i})
20                                       if (e_{j,i} ∉ E)
21                                           then add e_{j,i} to E
22 return G(V, E)  {return Preferences Graph}
```

The computational complexity of constructing and processing a preference graph is quadratic in the number of tuples. There is a tradeoff between a preference graph's expressiveness and the scalability of its implementation. Though in some embodiments, preferences may be highly "selective" and, consequently, the preference graph may be sparse.

Scalability issues due to the size of the preference graph may be addressed in any of numerous ways. One approach is to use distributed processing in a cloud environment, where storing and managing the preference graph is distributed over multiple nodes in the cloud. For example, a ranking algorithm described below in Section V.A may be easily adapted to function in a cloud environment. Other approaches include sacrificing the precision of preference query results by conducting approximate processing, or thresholding managed preferences to prune weak preferences early, to reduce the size of the preference graph.

A preference graph allows heterogeneous user preferences to be encoded using a unified graphical representation. Though, in some embodiments, computing a ranking of query results using such representation may require additional quantification of preference strength. Preference strength may be quantified based on the semantics of first-order and second-order preferences, while preserving the preference information encoded by the preference graph. Preference strength may be represented by weights on edges of the preference graph.

Given a preference graph *G(V,E),* the set of graph edges *E* may represent pairwise first-order preferences. Specifically, an edge $e_{i,j}$ may express the preference for tuple t_i over tuple t_j according to one or more scope comparator(s). In some instances, a weight $w_{i,j}$ may be associated with an edge $e_{i,j}$. The weight $w_{i,j}$ may be a weight indicative of a degree of preference for the first node over the second node. Stronger preferences may be indicated by higher weights. In some instances, the weight may be a weight between 0 and 1, inclusive and the sum of the weights $w_{i,j}$ and $w_{j,i}$ may equal 1. Disconnected vertices in the preference graph indicate that their corresponding tuples are indifferent with respect to each other.

In some embodiments, computing the weight may comprise dividing the number of first-order preferences for item A relative to item B by the number of all first-order preferences indicating any preference (either for or not for) item A.

For instance, let *F* be the set of all scope comparators associated with the preference graph. Let *A* be the set of POrders of *F* according to the chosen semantics of second-order preferences. Let $F_{i,j} = l(e_{i,j}) \cup l(e_{j,i})$. That is, $F_{i,j}$ is the set of scope comparators that state a preference relationship between tuples t_i and t_j . Let $A_{i,j}$ be the multiset of nonempty projections of POrders in *A* based on $F_{i,j}$. Let $A^+_{i,j} \subseteq A_{i,j}$ be the set of POrder projections under which $t_i \succ t_j$, and similarly let $A^-_{i,j} \subseteq A_{i,j}$ be the set of POrder projections under which $t_j \succ t_i$ It follows that $A_{i,j} = A^+_{i,j} \cup A^-_{i,j}$, and that $A^+_{i,j} \cap A^-_{i,j}$ is empty. The weight $w_{i,j}$ may be computed as follows:

$$w_{i,j} = |A^+_{i,j}|/|A_{i,j}| \qquad (1)$$

That is, $w_{i,j}$ corresponds the proportion of POrder projections, under which $t_i \succ t_j$, among the set of POrder projections computed based on comparators relevant

to the edge (t_i, t_j). The weight $w_{j,i}$ may be similarly defined using the set $A_{i,j}^{-}$. It follows that $w_{i,j} + w_{j,i} = 1$. For the case of Pareto composition, at most one of the two edges $e_{i,j}$ and $e_{j,i}$ can exist in the preference graph, since otherwise t_i and t_j would be incomparable. Hence, under Pareto composition, we remove any graph edge $e_{i,j}$ whenever an edge $e_{j,i}$ exists.

We next give an example illustrating how to compute preference weights under different semantics of second-order preferences.

Example 4

FIG. 15 shows three weighted preference graphs, corresponding to the preference graph in FIG. 14, produced under different semantics of second-order preferences. The different semantics of second-order preferences result in different edge weights and/or the removal of some edges in the original preference graph:

- Under prioritized comparators, $e_{1,6}$ is removed since, based on the shown comparator priorities, it may be determined that $(t_6 \succ t_1)$.
- Under partially-ordered comparators, we have that w_{23}=w_{32}=.5, since for the relevant (t_2,t_3) set of comparators is $\{f_{5,6}, f_{1,5}\}$ and the given partial order induces four POrder projections $\{\langle f_{1,5}, f_{5,6}\rangle, \langle f_{1,5}, f_{5,6}\rangle, \langle f_{5,6}, f_{1,5}\rangle, \langle f_{5,6}, f_{1,5}\rangle\}$, where $(t_2 \succ t_3)$ under the two POrder projections $\langle f_{5,6}, f_{1,5}\rangle, \langle f_{5,6}, f_{1,5}\rangle$, while $(t_3 \succ t_2)$ under the other two POrder projections $\langle f_{1,5}, f_{5,6}\rangle, \langle f_{1,5}, f_{5,6}\rangle$.
- Under pairwise preferences, $w_{5,6}$ = 0:33 since $(t_5 \succ t_6)$ based on $\langle f_{6,2}\rangle$, which is one out of three POrder projections $\{\langle f_{5,6}\rangle, \langle f_{6,2}\rangle, \langle f_{5,6}\rangle\}$.

V. Ranking

The graph-based preference model described in Section IV may be used to obtain a ranking (a total order) of items in a set of items. This may be done in any of numerous ways. One approach described in Section V.A obtains a ranking based on authority-based ranking algorithms. Another approach described in Section V.B is a probabilistic algorithm based on inducing a set of complete directed graphs called tournaments from

the graph-based preference model and computing a ranking for at least one tournament from the set.

A. Importance Flow Ranking

A total order of items (or, equivalently, tuples representing these items) may be obtained by estimating an importance measure for each tuple using the preference weights encoded by the weighted preference graph. Techniques related to the PageRank importance flow model may be used to compute such importance measures. Under the PageRank model, scores may be assigned to Web pages based on the frequency with which they are visited by a random surfer. Pages are then ranked according to these scores. Intuitively, pages pointed to by many important pages are also important.

The PageRank importance flow model lends itself naturally to problems that require computing a ranking based on binary relationships among items. In the context of preferences, the model may be applied based on the notion that an item may be important if is preferred over many other important items.

Let $G = (V, E)$ be a dominance graph (i.e., a directed graph in which an edge $e_{i,j}$ means $v_i \succ v_j$), and let $L(v)$ and $U(v)$ be the set of nodes dominated by and dominating v, respectively. Let $\alpha \in [0, 1]$ be a real number called a damping factor. The PageRank algorithm, as known in the art, computes the PageRank score of node v_i, denoted γ_i, according to:

$$\gamma_i = \frac{1-\alpha}{|V|} + \alpha \cdot \sum_{v_j \in L(v_i)} \frac{\gamma_j}{|U(v_j)|} \qquad (2)$$

The PageRank score of a node v is determined by summing PageRank scores of all nodes v' dominated by v, normalized by the number of nodes dominating v'. It is well known that when Equation 2 corresponds to a $\sum_{v_i \in V} \gamma_i = 1$ stationary distribution of a Markov chain, and that a unique stationary distribution exists if the chain is irreducible (i.e., the dominance graph is strongly connected), and aperiodic. Nodes that have no incoming edges (i.e., nodes that are not dominated by any other nodes) lead to sinks in the Markov chain, which makes the chain irreducible. This problem may be handled by adding self loops at sink nodes, or (uniform) transitions from sink states to all other states in the Markov chain. The damping factor α captures the requirement that each node is reachable from every other node. The value of α is the

probability that we stop following the graph edges, and start the Markov chain from a new random node. This may help to avoid getting trapped in cycles between nodes that have no edges to the rest of the graph.

Accordingly, in some embodiments a pagerank-based algorithm may be used to calculate a total order of items from the weighted preference graph. Herein, a pagerank-based algorithm refers to any algorithm based on calculating a value from a graph based on characteristics of a Markov chain defined with respect to the graph. Note that a difference between the above-described weighted preference graph and the graphs that the PageRank algorithm to which is conventionally applied is that the weighted preference graph has preference weights associated to edges. The preference weights bias the probability of transition (flow) from one state to another, according to weight value, in contrast to the conventional case in which transitions are uniformly defined.

A pagerank-based algorithm may proceed as follows. Given a starting tuple t_0 (node) in the weighted preference graph, assume a random surfer that jumps to a next tuple t_1, among the set of tuples dominating t_0, biased by the edge weights. Intuitively, this corresponds to a process where a tuple is constantly replaced by a more desired tuple (with respect to given preferences). Note that visiting tuples takes place in the opposite direction of edges (jumps are from a dominated tuple to a dominating tuple). Hence, it follows that tuples that are visited more frequently, according to this process, are more likely to be desirable than tuples that are visited less frequently. Ranking tuples based on their visit frequency (pagerank-based scores) defines an ordering that corresponds to their global desirability.

The weighted preference graph may be represented using a square matrix M, where each tuple may corresponds to one row and one column in M. Let E_j be the set of incoming edges to tuple t_j in the weighted preference graph. The entry M [i, j] may be computed as follows:

$$M[i,j] = \frac{w_{i,j}}{\sum_{e_{k,j} \in E_j} w_{k,j}} \qquad (3)$$

Hence, the sum of all entries in each column in M is 1.0 unless the tuple corresponding to that column has no incoming edges. Matrices in which all the entries are nonnegative and the sum of the entries in every column is 1.0 are called column

stochastic matrices. A stochastic matrix defines a Markov chain whose stationary distribution is the set of importance measures we need for ranking. In order to maintain the irreducibility of the chain, we need to eliminate sinks (nodes with no incoming edges in the preference graph). We handle the problem of sinks by adding a self loop, with weight 1.0, at each sink node.

Let Γ be the pagerank scores vector. Then, based on the previous matrix representation, the pagerank scores are given by solving the equation $\Gamma = M \cdot \Gamma$, which is the same as finding the eigenvector of M corresponding to eigenvalue 1. The solution that has been used in practice for computing pagerank scores is using the iterative power method, where Γ is computed by first choosing an initial vector Γ^0, and then producing a next vector $\Gamma^1 = M \cdot \Gamma^0$. The process is repeated to generate a vector Γ^T, at iteration T, using the vector Γ^{T-1}, generated at iteration $T - 1$. For convergence, at each iteration T, entries in Γ^T are normalized so that they sum to 1.0. In practice, the number of iterations needed for the power method to converge may be any suitable of iterations. For instance, tens or hundreds of iterations may be used.

FIG. 16 illustrates the pagerank matrix for the weighted preference graph with prioritized comparators illustrated in FIG. 15. Note that t_4 is a sink node with no incoming edges (i.e., t_4 has no other dominating tuples). Hence, we add a self loop with weight 1.0 to t_4, represented by the matrix entry $M[4, 4]$. A typical value of the damping factor α may be a value such as 0.15, but may be any value between 0 and 0.5.

B. Probabilistic Ranking

A total order of items (or top-ranked items) may be obtained from a complete directed graph derived from the preference model. Computing a total order of items from a complete directed graph (also known as a tournament) is termed finding a tournament solution. This problem may be stated as follows. Given an irreflexive, asymmetric, and complete binary relation over a set, find the set of maximal elements of this set. Example methods for finding tournament solutions are computing Kendall scores, and finding a Condorcet winner.

It should be appreciated, however, that the preference graph described in Section IV is not necessarily a tournament. In particular, the preference graph may be symmetric and incomplete:

- Symmetry: both edges $e_{i,j}$ and $e_{j,i}$ may exist in the preference graph,

- Incompleteness: both edges $e_{i,j}$ and $e_{j,i}$ may be missing from the preference graph.

The symmetry problem implies that some pairwise preferences may go either way with possibly different weights, while incompleteness implies that some pairwise preferences may be unknown.

In some embodiments, a probabilistic approach to obtaining a ranking from the preference graph may be used. Such an approach may rely on deriving one or more tournaments from the preference graph. Each tournament may be associated with a probability. As such, a weighted preference graph may be viewed as a compact representation of a space of possible tournaments, wherein each tournament is obtained by repairing the preference graph to obtain an asymmetric and complete digraph. In order to construct a tournament, two repair operations may be applied to the preference graph:

- Remove an edge. Applying this operation eliminates a 2-length cycle by removing one of the involved edges.
- Add an edge. Applying this operation augments the graph by adding a missing edge.

As discussed earlier, the value of the weight $w_{i,j}$ represents the probability of selecting a POrder, among the set of all POrders relevant to (t_i, t_j), under which $(t_i \succ t_j)$. We thus interpret $w_{i,j}$ as the probability with which tuple t_i is preferred to tuple t_j. We further assume the independence of $w_{i,j}$ values of different tuple pairs. For each tuple pair (t_i, t_j), if both $w_{i,j} > 0$ and $w_{j,i} > 0$ (i.e., t_i and t_j are involved in a 2-length cycle), the operation *remove edge* removes the edge $e_{j,i}$ with probability $w_{j,i}$ and removes the edge $e_{j,i}$ otherwise. Alternatively, if $w_{i,j} = 0$ and $w_{j,i} = 0$ (i.e., t_i and t_j are disconnected vertices), the operation *add edge* adds one of the edges $e_{i,j}$ or $e_{j,i}$ with the same probability 0.5.

Based on the probabilistic process described above, repairing the weighted Preference graph generates a tournament (irreflexive, asymmetric, and complete digraph) whose probability is given by the product of the probabilities of all remaining graph edges. Let c be the number of 2-length cycles in the Preference graph, and d be the number of disconnected tuple pairs. Then, the number of possible tournaments is 2^{c+d}.

FIG. 17 illustrates a weighted preference graph, and the corresponding set of possible tournaments $\{T_1, \ldots, T_8\}$. The illustrated preference graph has two 2-length cycles $(t_1 - t_2$ and $t_2 - t_3)$ and one pair of disconnected tuples (t_2, t_4), and hence the number of possible tournaments is 8. The probability of each tournament is given by the product of the probabilities associated with its edges. For example, the probability of T_1 is 0.09, which is the product of 0.3, 0.6, and 0.5 representing $w_{2,1}$, $w_{2,3}$, and $w_{4,2}$, respectively.

Given a tournament T and a total order of tuples O, we say that O violates T, with respect to the relative order of (t_i, t_j), if $t_i \succ t_j$ under O, while $t_j \succ t_i$ under T. The problem of computing a total order of tuples with a minimum number of violations to tournament is known to be NP-hard. Multiple heuristics have been proposed to compute a total order from a tournament. We focus on using Kendall score for computing a total order. The Kendall score of tuple t is the number of tuples dominated by t according to the tournament.

The space of possible tournaments allows computing a total order of tuples under any of numerous probabilistic ranking measures. Two specific measures are described below.

- *Most probable tournament ranking*. Compute a total order of tuples based on the tournament with the highest probability.
- *Expected ranking*. Compute a total order of tuples based on the expected ranking in the space of all the possible tournaments.

Finding the most probable tournament is done by maintaining the edge with the higher weight for each 2-length cycle in the preference graph, and adding an arbitrary edge for each pair of disconnected tuples. According to this method, there may be multiple tournaments with the highest probability among all possible tournaments. The computed total order under any of these tournaments is the required ranking. In the illustrative example of FIG. 17, tournaments T_2 and T_6 are the most probable tournaments, each with probability 0.21. A total order of tuples in T_2 using Kendall scores is $\langle t_1, t_4, t_2, t_3 \rangle$ while a total order of tuples in T_6 is $\langle t_1, t_2, t_3, t_4 \rangle$. Let n be the number of tuples in the preference graph, the complexity of the algorithm is $O(n^2)$, since we need to visit all edges of the preference graph.

Finding the expected ranking may be done by computing the expected Kendall score for each tuple using the space of possible tournaments. We model the score of tuple t_i as a random variable s_i whose distribution is given by the space of possible tournaments. In the illustrative example of FIG. 17, t_1 dominates one tuple in $\{T_1, T_3, T_5, T_7\}$ with probability summation 0.3, while t_1 dominates two tuples in $\{T_2, T_4, T_6, T_8\}$ with probability summation 0.7. Hence, the random variable s_i may take the value 1 with probability 0.3, and takes the value 2 with probability 0.7. The expected value of s_i is thus 1*0.3+2*0.7=1.7.

Computing the exact expected score of each tuple requires materializing the space of possible tournaments, which is infeasible due to the exponential number of possible tournaments. We thus propose a sampling-based algorithm to approximate the expected value of s_i of each tuple t_i, and then rank tuples based on their estimated expected scores. Let $L(t_i)$ be the set of tuples dominated by t_i in the weighted preference graph.

For a tuple t_i, a sample Z is generated by adding $t_j \in L(t_i)$ each tuple *to* Z with probability $w_{i,j}$. All samples may be generated independently. Hence, a score sample from s_i distribution is given by $|Z|$. The expected value of s_i is estimated as the mean of the generated score samples. It is well known that sample mean, computed from a sufficiently large set of independent samples, is an unbiased estimate of the true distribution mean. Let n be the number of tuples in the preference graph, and m be the number of drawn samples for each tuple, the complexity of the algorithm is $O((nm)^2)$, since we access the dominated set of each tuple m times to generate m score samples.

VI. Interactive Preference Specification

A data exploration system may help a user to specify preferences. In some embodiments, preferences may be specified interactively. A system may interact with a user through a series of prompts, displays, and/or indications of the type of input a user may provide the system. The system may provide the user with information that may assist the user in specifying preferences.

A data exploration system may assist a user to query the system. To this end, the data exploration system may assist the user to specify preferences and may output query results, to the user, ranked in accordance with the specified preferences.

FIG. 18 shows a flowchart of an illustrative process 1200 for assisting a user to query a data exploration system. Process 1200 may be used to assist a user specify user preferences in conjunction with a query, and may assist a user specify preferences associated with attributes related to one or more keywords in a query.

Process 1200 begins in act 1202 when a user query may be inputted. The inputted query may be any suitable query and may be a text query. The inputted query may be a multimedia query, for example, received through an audio input device that may be translated into text using any appropriate speech-recognition/speech-to-text software. The inputted query may comprise one or more keywords. The query may be, for example, a query for an item to purchase and/or may be a query for an item comprising information desired by a user. For instance, the query may be a query containing the keyword "car" and may indicate that a user may be interested in looking at items related to cars. As another example, the user may input a query "television" into an Internet search engine, which may indicate that a user may be interested in looking at any webpages containing information about television. Though a query may be any suitable query, as known in the art.

In response to receiving a user query, one or more attributes related to the query may be identified, in act 1204 of process 1200. Attributes may be related to one or more keywords contained in the query. For instance, attributes may be a characteristic of a keyword in the query. Attributes may be of any suitable type. For instance, attributes may be categorical attributes or numerical attributes. For instance, if a query for a "car" were inputted in act 1202, then attributes related to car may be the attributes "Make," "Color," "Price," and any other attributes of car such as the attributes illustrated in FIG. 8. Attributes related to one or more keywords contained in a query may be identified in any suitable way as known in the art. They may be identified automatically by a computer or may be manually specified.

Regardless of the way in which attributes are identified, in act 1204, a user may be

presented with these attributes, in act 1206. The user may be shown these attributes visually using a display screen that contains these attributes. The display screen may be any suitable screen containing a representation of the attributes, such as a text representation of the attributes.

The user may be prompted to select one or more of the presented attributes such that the system may assist the user to specify preferences associated with the selected attributes. For instance, a user may be presented with a list of previously-mentioned attributes associated with the keyword "car" and may select the attributes "Price" and "Color." In act 1208, attributes selected by the user may be received.

In response to receiving the selected attributes, the user may be prompted to specify first-order preferences associated with one or more selected attributes, in act 1210. For each attribute, the user may specify a first-order preference of any suitable type. For instance, the user may specify score-based preferences, partial order preferences, skyline preferences, and/or conjoint analysis preferences as discussed with reference to Section II.

The user may be assisted in specifying any of the above-mentioned first-order preferences in any of numerous ways. In some embodiments, a graphical user interface may be used. The graphical user interface may allow the user to graphically represent the first-order preferences (e.g., by drawing preferences). In some embodiments, the user may be provided with a series of prompts designed to obtain information required to specify first-order preferences.

In response to receiving first-order preferences, the user may be prompted to specify a second-order preference among the received first-order preferences, in act 1212. The user may specify a second-order preference of any suitable type. For instance, the user may specify prioritized preference composition preferences, partial order preferences, pairwise preferences, and/or Pareto preference composition preferences as discussed with reference to Section III.

Similar to the case of first-order preferences, a user may be assisted in specifying any of the above-mentioned second-order preferences in any of numerous ways. In some embodiments, a graphical user interface may be used. The graphical user interface may allow the user to graphically represent the second-order preferences. In some embodiments, the user may be provided with a series of prompts designed to obtain information required to specify second-order preferences. After first-order and second-order preferences have been specified, process 1200 completes.

The above-described embodiments of the present invention can be implemented in any of numerous ways. For example, the embodiments may be implemented using

hardware, software or a combination thereof. When implemented in software, the software code may be embodied as stored program instructions that may be executed on any suitable processor or collection of processors (e.g., a microprocessor or microprocessors), whether provided in a single computer or distributed among multiple computers.

It should be appreciated that a computer may be embodied in any of numerous forms, such as a rack-mounted computer, a desktop computer, a laptop computer, or a tablet computer. Additionally, a computer may be embodied in a device not generally regarded as a computer, but with suitable processing capabilities, including a Personal Digital Assistant (PDA), a smart phone, a tablet, a reader, or any other suitable portable or fixed electronic device.

Also, a computer may have one or more input and output devices. These devices may be used, among other things, to present a user interface. Examples of output devices that may be used to provide a user interface include printers or display screens for visual presentation of output, and speakers or other sound generating devices for audible presentation of output. Examples of input devices that may be used for a user interface include keyboards, microphones, and pointing devices, such as mice, touch pads, and digitizing tablets.

Such computers may be interconnected by one or more networks in any suitable form, including networks such as a local area network (LAN) or a wide area network (WAN), such as an enterprise network, an intelligent network (IN) or the Internet. Such networks may be based on any suitable technology and may operate according to any suitable protocol and may include wireless networks, wired networks, and/or fiber optic networks.

Thus, in an embodiment, there is provided a method for querying a data exploration system managing a plurality of items, the method comprising: querying the data exploration system with a query comprising a plurality of first-order user preferences indicative of a user's preferences among items in the plurality of items, and a second-order user preference indicative of the user's preferences among first-order user preferences in the plurality of first-order user preferences; calculating, with a processor, a ranking of an item in the plurality of items based at least in part on a data structure encoding a preference graph that represents the plurality of first-order user preferences

and the second-order user preference; and outputting at least a subset of the plurality of items to the user, in accordance with the ranking.

In an embodiment, calculating the ranking comprises: applying a pagerank-based algorithm to the data structure encoding the preference graph to calculate the ranking.

In another embodiment, the preference graph comprises a plurality of nodes, wherein each node represents an item, and calculating the ranking comprises: calculating a pagerank score of a node in the plurality of nodes.

In another embodiment, calculating the ranking comprises: computing a total order of nodes in a complete directed graph derived from the preference graph, wherein each node represents an item.

In another embodiment, computing the total order comprises calculating a Kendall score for a node in the complete directed graph.

In another embodiment, the preference graph comprises: a plurality of nodes, wherein each node corresponds to an item in the plurality of items; and a plurality of edges, wherein each edge corresponds to a first-order preference in the plurality of first-order preferences, the first-order preference indicating a user preference for one of the two items represented by nodes terminating the edge.

In another embodiment, each edge is a directed edge, directed to a node associated with a preferred item as indicated by the corresponding first-order preference.

In another embodiment, a weight is associated to an edge between a first node and a second node in the preference graph, the weight being indicative of a degree of preference for the first node over the second node.

In another embodiment, each item in the plurality of items is represented as a tuple, the tuple comprising a plurality of attributes of the item.

In another aspect, there is provided a computer-readable storage medium article storing a data structure encoding a preference graph and a plurality of processor-executable instructions that when executed by a processor, cause the processor to perform the acts of: receiving a plurality of first-order user preferences indicative of user preferences among a plurality of items; receiving a second-order user preference indicative of user preferences among the first-order preferences in the plurality of first-order user preferences; computing a weight for an edge of the preference graph based on the plurality of first-order user preferences and the second-order user preference,

wherein: the edge connects a first node associated with a first item and a second node associated with a second item, and the weight is indicative of a degree of preference for the first item over the second item; and outputting at least two of the plurality of items according to the preference graph.

In an embodiment, the preference graph comprises a node for each item in the plurality of items and an edge for every pair of nodes associated with items related by a first-order preference in the plurality of first-order preferences.

In another embodiment, the computing the weight comprises: computing a first number of first-order user preferences in the plurality of first-order user preferences indicating a user's preference for the first item relative to the second item; computing a second number of all first-order user preferences in the plurality of first-order user preferences indicating any preference associated with the first item; and setting the weight based on the first number divided by the second number.

In another embodiment, receiving the plurality of first-order user preferences comprises receiving a first-order preference from a user.

In another embodiment, each item in the plurality of data items is represented as a tuple, the tuple comprising values of a plurality of attributes; and each first-order user preference in the plurality of first-order user preferences indicates a user preference of one item over another item based at least in part on a value of an attribute of a first tuple, representing the one item, and a value of an attribute of a second tuple representing the other item.

In another embodiment, the plurality of first-order user preferences comprises at least two types of first-order preferences selected from the group comprising score-based preferences, partial order preferences, skyline preferences, and conjoint analysis preferences.

In another embodiment, the second-order user preference comprises a plurality of second-order user preference relations that comprises at least two types of second-order preferences selected from the group comprising prioritized preference composition, partial order preferences, pairwise preferences, and Pareto preference composition.

In another aspect, there is provided a database system comprising: a memory configured to store a plurality of tuples, a data structure encoding a preference graph to represent user preferences, wherein the user preferences comprise a plurality of first-

order preferences representing user preferences among tuples and a second-order user preference representing user preferences among first-order preferences in the plurality of first-order preferences; and a processor configured to access contents of the memory and compute a ranking of a tuple in the plurality of tuples based on the data structure encoding the preference graph.

In another aspect, there is provided a system for interactive preference management, the system comprising: a memory configured to store a plurality of tuples, each tuple comprising a value for at least one of a plurality of attributes; at least one processor configured to receive a range of values for an attribute in the plurality of attributes from a user, output an integer indicative of a number of tuples comprising a value for the attribute such that the value is in the range of values.

In another aspect, there is provided a computer-implemented method for interactive preference management, the method comprising: receiving, with a processor, a query from a user, the query comprising a keyword; prompting the user to provide a plurality of first-order preferences associated with one or more attributes related to the keyword; and in response to receiving the plurality of first-order preferences, prompting the user to provide a second-order preference among the first-order preferences in the plurality of first-order preferences.

In an embodiment, prompting the user to provide a plurality of first-order preferences comprises: presenting a list of attributes related to the keyword to the user; receiving a selection of attributes in the list of attributes from the user; and prompting the user to specify a first-order preference associated with the selected attribute.

REFERENCE B: SYSTEM AND METHOD OF PREFERENCE GUIDED DATA EXPLORATIONS APPLIED TO ATOMIC SEMANTICS

Broadly, knowledge representation is the activity of making abstract knowledge explicit, as concrete data structures, to support machine-based storage, management, and reasoning systems. Conventional methods and systems exist for utilizing knowledge representations (KRs) constructed in accordance with various types of knowledge representation models, including structured controlled vocabularies such as taxonomies, thesauri and faceted classifications; formal specifications such as semantic networks and ontologies; and unstructured forms such as documents based in natural language.

The above-mentioned knowledge representation models, and knowledge representations in general, are tools for modeling human knowledge in terms of explicit concepts and the relationships among those concepts, and for making that knowledge accessible to machines such as computers for performing various knowledge-requiring tasks. As such, human users and software developers conventionally construct KR data structures using their human knowledge, and manually encode the completed KR data structures into machine-readable form to be stored in machine memory and accessed by various machine-executed functions.

It has been recognized that the conventional non-automated approaches to constructing knowledge representations lead to a number of problems including the inability to scale with increasing size of data, inability to deal with complex and large data structures, dependence on domain experts, cost of large-scale data storage and processing, and integration and interoperability challenges.

It has been recognized that methods for automated construction of knowledge representations are required in order to address the above-mentioned shortcomings of conventional approaches. Accordingly, some embodiments in accordance with the present disclosure provide a system that encodes knowledge creation rules to automate the process of creating knowledge representations, and employs probabilistic methods to check the semantic coherence of the data structures that result from the application of knowledge creation rules.

Many approaches for using knowledge creation rules to automate the creation of knowledge representations are possible. For instance, methods for automating the creation of knowledge representations based on knowledge creation rules were disclosed in U.S. Provisional Application No. 61/357,266, filed 06/22/2010, and entitled "Systems and Methods for Analyzing and Synthesizing Complex Knowledge Representations."

Some embodiments of the above-mentioned approach combine a compressed (atomic) data set with a set of generative rules that encode the underlying knowledge creation, instead of modeling all the knowledge in the domain as explicit data. Such rules may be applied by the system in some embodiments when needed or desired to create new knowledge and express it explicitly as data. A benefit of such techniques may be, in at least some situations, to substantially reduce the amount of stored data in the system

by more efficiently representing the stored data, as well as to provide new capabilities and applications for machine-based creation (synthesis) of new knowledge.

By incorporating an underlying set of rules of knowledge creation within the KR, the amount of data in the system may be reduced, providing a more economical system of data management, and providing entirely new applications for knowledge management. Thus, in some embodiments, the cost of production and maintenance of KR systems may be lowered by reducing data scalability burdens, with data not created unless it is needed. Once created, the data structures that model the complex knowledge in some embodiments are comparatively smaller than in conventional systems, in that they need not store the data that is not relevant to the task at hand. This in turn may reduce the costs of downstream applications such as inference engines or data mining tools that work over these knowledge models.

It has been recognized that methods are needed for checking the semantic coherence for the knowledge representation data structures resulting from application of knowledge creation rules. For instance, in some embodiments evidence may be gathered as to whether the resulting data structures present in existing knowledge models. These existing knowledge models may be internal to the system (as complex knowledge representation data structures) or external (such as knowledge models encoded on the Semantic Web). In some embodiments, a search engine may be used to investigate whether terms (symbols or labels) associated with concepts of the resulting data structures present in external knowledge representations (such as documents). The term-document frequency (e.g., number of search engine hits) may provide one exemplary metric for the semantic coherence of the resulting knowledge representation data structures.

It has been further recognized that probabilistic methods for synthesis of semantic networks may be used for checking the semantic coherence for the knowledge representation data structures resulting from application of knowledge creation rules.

A semantic network is a type of knowledge representation, and it comprises a directed graph consisting of vertices, which represent concepts, and edges, which represent semantic relationships between concepts. Semantic networking is a process of developing these graphs, and provides a way to model and store knowledge so that a computer-implemented program may process and use it. A key part of developing

semantic graphs is the provisioning of concept definitions and concept relationships based on existing knowledge representations such as documents and unstructured text.

It has been recognized that semantic coherence among a set concepts in a knowledge representation may be ascertained by constructing a semantic network that comprises these concepts and indicates the degree of uncertainty pertaining to the existence of a relationship (edge) between any two concepts (vertices) in the semantic network. It has been recognized that statistical graphical models and associated methods are uniquely suited to provide this probabilistic representation of semantic networks, especially because the computational complexity of associated inference methods scales favorably with the size of the semantic network, in turn, yielding computationally-efficient methods and systems. It has also been recognized that statistical inference methods for graphical models may be used to determine whether a relationship exists between any set of concepts and to quantify the uncertainty associated to each such relationship.

In some embodiments, statistical inference techniques may be used to efficiently compute the joint probability distribution of all the concepts in the graph, while taking into account any a priori assumptions about the dependence structure among concepts. For instance, it may be known that certain concepts are independent, or it may be known that some concepts are strongly correlated. The joint probability distribution of all the concepts in the graph may be used to answer any queries about relationships among any concepts included in the graph. For example, the extent to which any two concepts are related, semantically coherent, or whether one concept is relevant to another, may be obtained by computing the appropriate marginal posterior probabilities.

In some embodiments, knowledge representations constructed through the application of knowledge creation rules (i.e., elemental semantics) and a probabilistic method for evaluating semantic coherence among concepts, may be further refined by user feedback.

Embodiments of the present disclosure may be further appreciated through an illustrative knowledge representation construction system illustrated in FIG. 19. An inputting unit (1) of the KR construction system may be configured to receive a first complex knowledge representation. The first complex knowledge representation may comprise complex vocabularies. Complex vocabularies may include lexicons, and upper

ontologies. An analysis engine (2) may decompose the inputted knowledge representation into atomic level semantic units using knowledge generation rules. The resultant atomic level semantic units may be stored for subsequent use in synthesis operations in an atomic semantics database (3).

A rules database (4) may store knowledge generation rules for composition (synthesis) and decomposition (analysis) of complex semantics. Statistical graphical models that may provide statistical evidence of coherence between atomic concepts in the formation of more complex semantics are stored as an atomic semantic kernel in the statistical model database (5). The statistical graphical models may be constructed by sampling reference corpora within knowledge domains.

A synthesis engine (6) may compose a second complex knowledge representation by applying knowledge generation rules to atomic level semantic units in view of statistical evidence of coherence among atomic level semantic units as provided by a statistical graphical model. The composed knowledge representation may be outputted in any suitable knowledge representation format and may be stored as user models (7) for subsequent use. For instance, the composed knowledge representation may be outputted to a user interface such as a monitor, a screen on a mobile device, or any otherwise suitable interface. A feedback engine (8) may be configured for facilitating maintenance and quality improvements of constructed complex knowledge representations using a complex-adaptive feedback loop, wherein output complex semantics are returned for re-analysis and refinement.

Another aspect involves the incorporation of preference ranking engine (9) as shown in FIG. 20. The preference ranking engine (9) is described in "PrefEx: Preference Guided Data Exploration" by Ihab F. Ilyas and Mohamed A. Soliman. Generally, preference ranking engine (9) allows for integration of a user's preference relating to attributes. In the context of AKRM shown in FIG. 20, user-intent based on input is ranked. User-intent can be the user's preferences with respect to attributes, or stated another way, what the user intends to find on varying levels of importance to the user.

A query received by the user, that can include and be described as labels, is translated into a semantic representation (as concepts and concept relationships) within the system. This is also known as label-to-concept translation or LCT. The preference ranking engine (9) employs its user-preference approaches to assign weights to the

concepts from LCT. This weighting can then be used as the basis for synthesis operations, influencing the resultant topology and timing of the semantic representation. (For example, a more heavily weighted concept will have more additional concepts synthesized around it than a less heavily weighted concepts. Also, a more heavily weighted concept can have concepts synthesized before a less heavily weighted concept, thus providing temporal priority.)

The preference ranking engine (9) can also add aspects to the ordering of the output. The synthesis engine composes and outputs complex semantic representations, or stated another way outputs synthesized concepts and concept relationships. These semantic representations can, instead of being directly displayed to the user or user-model, fed into preference ranking engine (9). Based on the ability of preference ranking engine (9) to order objects based on a user's preference of attributes, the synthesized semantic representation fed into preference ranking engine can be ordered based on user-preferences. The resultant ordered concepts and concept relationships can then be delivered or outputted to the user or semantic user model (7).

Thus, in an aspect, there is provided a system, the system comprising: preference ranking component configured to establish attribute preferences based on user-intent; a synthesis engine configured to assign weights to semantic representations retrieved based on queries; wherein the assigned weights are based on the attribute preferences user intent; wherein the preference ranking component structures outputted synthesized semantic representations according to a rank based on the user-intent; and wherein the outputted synthesized semantic representations are delivered to a user-interface or a user model.

In an embodiment, a larger number of additional concepts are synthesized around a more heavily weighted concept.

In another embodiment, additional concepts are synthesized earlier around a more heavily weighted concept.

In another embodiment, method is implemented in software executing on at least one hardware processor of at least one computer.

Various aspects of the present invention may be used alone, in combination, or in a variety of arrangements not specifically discussed in the embodiments described in the foregoing and are therefore not limited in their application to the details and arrangement of components set forth in the foregoing description or illustrated in the drawings. For example, aspects described in one embodiment may be combined in any manner with aspects described in other embodiments.

All definitions, as defined and used herein, should be understood to control over dictionary definitions, definitions in documents incorporated by reference, and/or ordinary meanings of the defined terms.

The phraseology and terminology used herein is for the purpose of description and should not be regarded as limiting. The use of "including," "comprising," "having," "containing", "involving", and variations thereof, is meant to encompass the items listed thereafter and additional items.

Also, embodiments of the invention may be implemented as one or more methods, of which an example has been provided. The acts performed as part of the method(s) may be ordered in any suitable way. Accordingly, embodiments may be constructed in which acts are performed in an order different than illustrated, which may include performing some acts simultaneously, even though shown as sequential acts in illustrative embodiments.

Use of ordinal terms such as "first," "second," "third," etc., in the claims to modify a claim element does not by itself connote any priority, precedence, or order of one claim element over another or the temporal order in which acts of a method are performed. Such terms are used merely as labels to distinguish one claim element having a certain name from another element having a same name (but for use of the ordinal term).

The indefinite articles "a" and "an," as used herein, unless clearly indicated to the contrary, should be understood to mean "at least one."

As used herein, the phrase "at least one," in reference to a list of one or more elements, should be understood to mean at least one element selected from any one or more of the elements in the list of elements, but not necessarily including at least one of

each and every element specifically listed within the list of elements and not excluding any combinations of elements in the list of elements. This definition also allows that elements may optionally be present other than the elements specifically identified within the list of elements to which the phrase "at least one" refers, whether related or unrelated to those elements specifically identified. Thus, as a non-limiting example, "at least one of A and B" (or, equivalently, "at least one of A or B," or, equivalently "at least one of A and/or B") can refer, in one embodiment, to at least one, optionally including more than one, A, with no B present (and optionally including elements other than B); in another embodiment, to at least one, optionally including more than one, B, with no A present (and optionally including elements other than A); in yet another embodiment, to at least one, optionally including more than one, A, and at least one, optionally including more than one, B (and optionally including other elements); etc.

The phrase "and/or," as used herein, should be understood to mean "either or both" of the elements so conjoined, i.e., elements that are conjunctively present in some cases and disjunctively present in other cases. Multiple elements listed with "and/or" should be construed in the same fashion, i.e., "one or more" of the elements so conjoined. Other elements may optionally be present other than the elements specifically identified by the "and/or" clause, whether related or unrelated to those elements specifically identified. Thus, as a non-limiting example, a reference to "A and/or B", when used in conjunction with open-ended language such as "comprising" can refer, in one embodiment, to A only (optionally including elements other than B); in another embodiment, to B only (optionally including elements other than A); in yet another embodiment, to both A and B (optionally including other elements); etc.

As used herein, "or" should be understood to have the same meaning as "and/or" as defined above. For example, when separating items in a list, "or" or "and/or" shall be interpreted as being inclusive, i.e., the inclusion of at least one, but also including more than one, of a number or list of elements, and, optionally, additional unlisted items.

Having described several embodiments of the invention in detail, various modifications and improvements will readily occur to those skilled in the art. Such modifications and improvements are intended to be within the spirit and scope of the invention. Accordingly, the foregoing description is by way of example only, and is not

intended as limiting. The invention is limited only as defined by the following claims and the equivalents thereto.

CLAIMS:

1. A computer network implemented system employing a semantic network, the system including a computer network of one or more networked computer devices with a processor and a memory and comprising:

a) at least one computer device storing a data structure providing a semantic network;

b) a plurality of computer-implemented agents deployed within said computer network, executing on one or more processors within the computer network, and interactive with the semantic network; and

c) a user interface configured to permit a user to at least create or modify the semantic network;

d) wherein the agents are configured to read and modify the semantic network without receiving explicit instructions from a user after their initial deployment.

2. The system of claim 1, wherein modifying the semantic network includes changing, editing, altering, augmenting, adding to or deleting from the semantic network.

3. The system of claim 1, wherein the agents include at least one of a harvesting agent, data mining agent, search agent, connecting agent, personal agent or shopping agent.

4. The system of claim 1, wherein at least two of the plurality of agents collaborate with each other, explicitly or implicitly.

5. The system of claim 1, wherein the system is configured to allow at least one of the plurality of agents to be selected by the user.

6. The system of claim 1, wherein the user interface is a graphical user interface.

7. The system of claim 6, wherein the user interface further includes mind-mapping software or ontology-building software.

8. The system of claim 6, wherein the user-interface presents values for the user to change, the option to add entries or the option to delete entries.

9. The system of claim 8, wherein the values or entries are modifiable by user-selection with a cursor, mouse-pointer or keyboard.

10. The system of claim 9, wherein the option to add entries includes a synthesized concept presented to the user for adding to the semantic network.

11. The system of claim 1, wherein at least one of the agents synthesizes a concept to the semantic network.

12. The system of claim 1, wherein the semantic network includes at least two nodes that each represent a distinct concept and at least one edge that represents a semantic relationship between two distinct concepts.

13. The system of claim 12, wherein:

at least one of the concepts is weighted based on preferences of a user;

at least one of the edges is weighted based on the strength of the semantic relationship; or

at least one of the results is delivered to the user in an order based on preferences of the user.

14. The system of claim 1, wherein at least one of the agents of the plurality of agents changes a value, adds or deletes an entry in the semantic network.

15. The system of claim 14, wherein at least a second agent of the plurality of agents acts upon or in response to the changed value, addition or deletion of the entry.

16. The system of claim 1, wherein a fee is charged when a user assigns a reporting task to at least one of the agents created by another user.

17. A method comprising:

providing a semantic network in a non-transitory, computer-readable medium within a computer network;

providing a plurality of computer-implemented agents deployed within said computer network and interactive with the semantic network, the agents being configured, collectively, to read and modify the semantic network without receiving explicit instructions from a user; and

providing a user interface configured to permit a user to modify the semantic network.

18. A method to decouple user and agent actions with respect to a semantic network, comprising:

providing an information exchange platform comprising an editable semantic network instantiated in a non-transitory, computer-readable medium within a computer network;

providing a plurality of computer-implemented agents deployed within said computer network and interactive with the semantic network, the agents collectively being configured to autonomously read and modify the semantic network; and

providing a user interface configured to permit a user to at least receive reports regarding or to modify the semantic network.

19. A method comprising:

making available to users of a computer network a semantic network building tool and a plurality of computer-implemented agents deployable within said computer network and interactive with a semantic network constructed by the user with the tool, the agents collectively being configured to read and modify the semantic network without receiving explicit instructions from a user.

20. The method of claim 19, further including providing a user interface configured to permit a user to modify the semantic network.

21. A method comprising:

providing an on-line facility configured to permit a user to deploy a plurality of computer-implemented agents within a computer network in which a semantic network

is embodied in a non-transitory, computer-readable medium, at least one agent being configured to read the semantic network and at least one agent being configured to modify the semantic network, without receiving explicit instructions from a user.

22. The method of claim 21, further including providing a user interface configured to permit a user to modify the semantic network.

23. The method of claim 22, wherein one or more agents communicate results to the user.

24. A method to decouple user and agent actions with respect to a semantic network, comprising:

providing an information exchange platform comprising an editable semantic network instantiated in a non-transitory, computer-readable medium within a computer network, or a tool permitting a user to create an editable semantic network instantiated in a non-transitory, computer-readable medium within a computer network; and

providing a facility configured to allow a user to deploy a plurality of computer-implemented agents within said computer network and interactive with the semantic network, the agents collectively being configured to autonomously read and modify the semantic network.

25. The method of claim 24, further including providing a user interface configured to permit a user to at least receive reports regarding, or to modify, the semantic network.

26. The method of any of claims 17-25, wherein at least one of said agents is configured to, selectively, augment the semantic network with a connection to another semantic network or with information from a source external to the network.

27. The method of claim 26, wherein the source external to the network is another semantic network.

28. At least one non-transitory computer-readable storage medium encoded with a plurality of computer-executable instructions that includes:

a semantic network module configured to provide a data structure that includes a semantic network;

an agent-interface module configured to allow interaction between a plurality of computer-implemented agents and the semantic network; and

a user-editing module configured to permit, through a user interface, modification of the semantic network by a user.

29. The at least one computer-readable storage medium according to claim 21, wherein the instructions, when executed, further perform outputting to data to the user, wherein said outputting is based on a function of one of the plurality of agents.

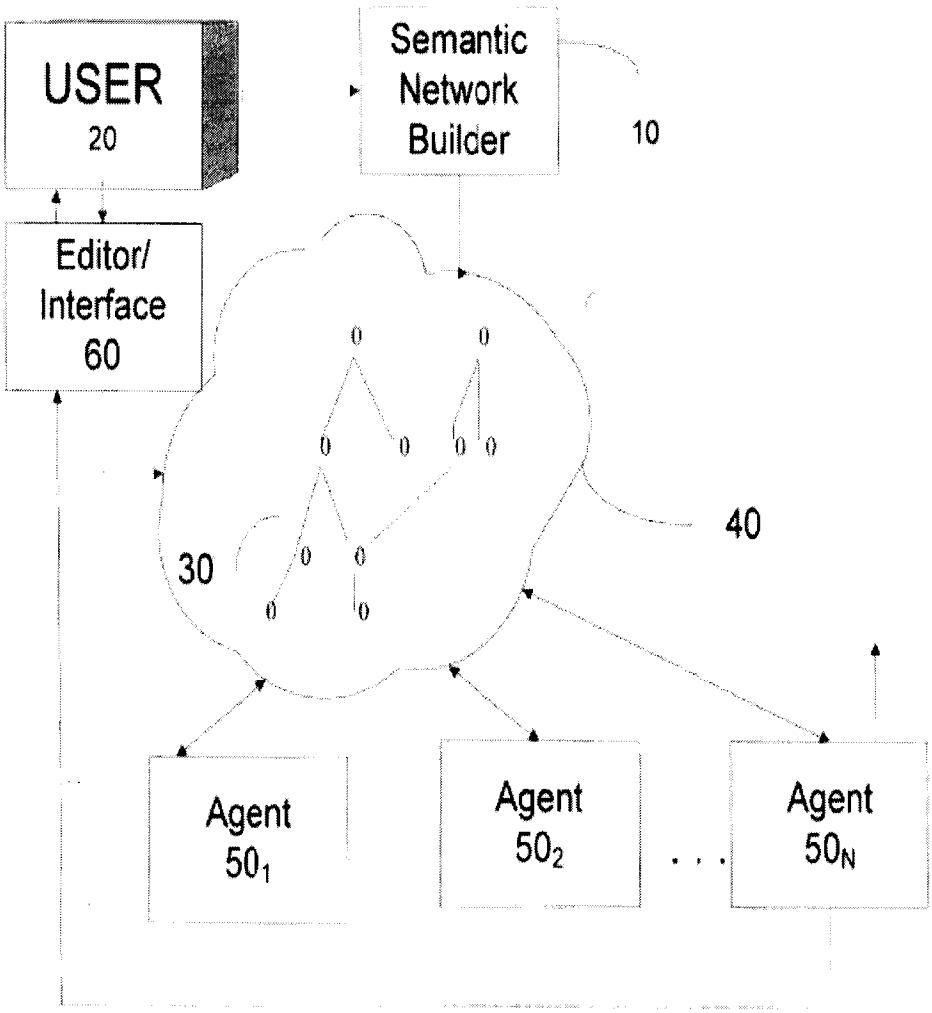
USER
20
Semantic
Network
Builder
10
Editor/
Interface
60
30
40
Agent
50_1
Agent
50_2
. . .
Agent
50_N

FIG. 1

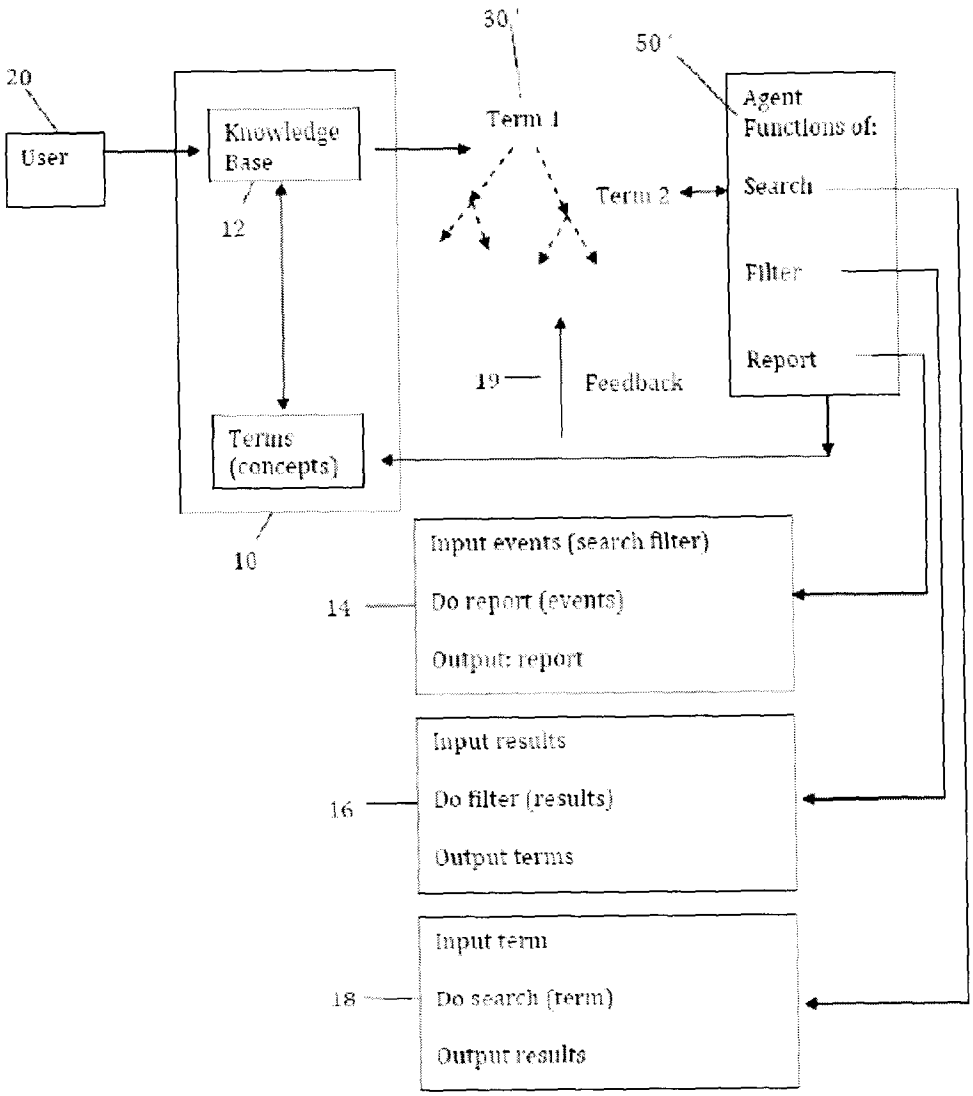
30
50
20
User
Knowledge Base
12
Term 1
Term 2
Agent Functions of:
Search
Filter
Report
19
Feedback
Terms (concepts)
10
Input events (search filter)
14
Do report (events)
Output: report
Input results
16
Do filter (results)
Output terms
Input term
18
Do search (term)
Output results

FIG. 2

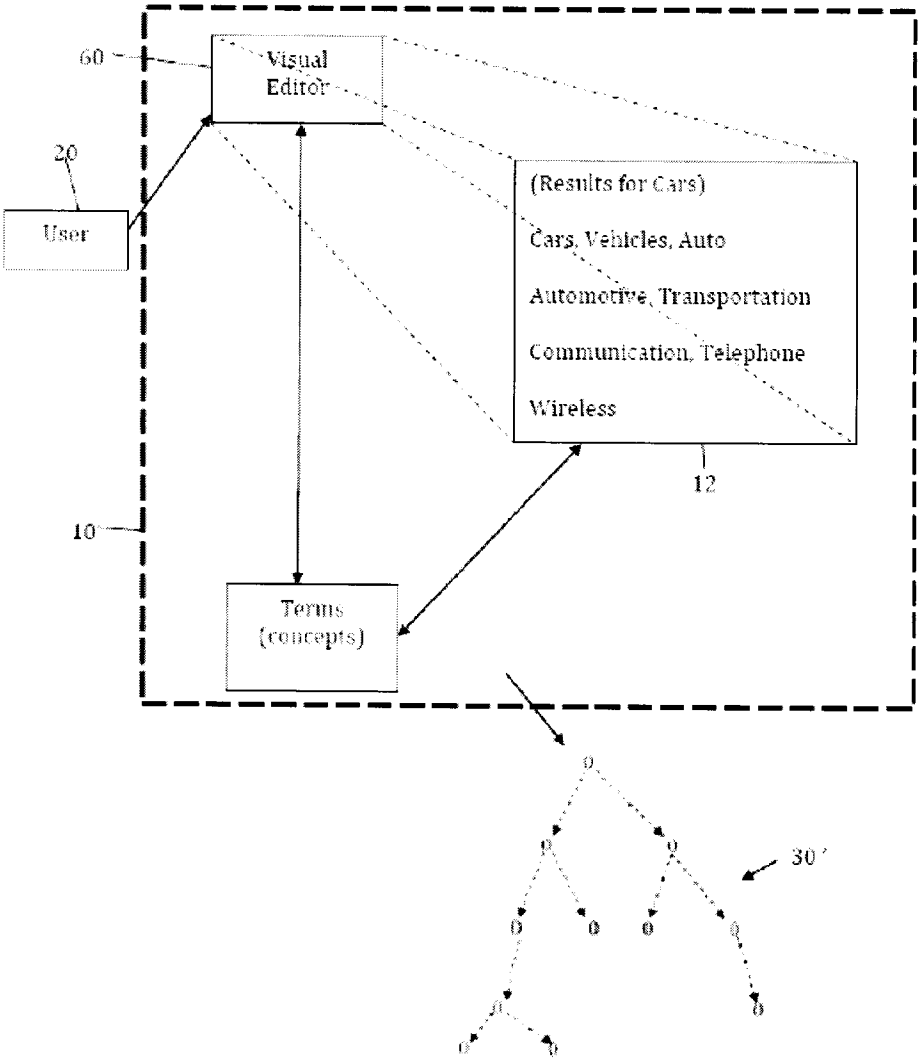
60
Visual Editor
20
User
(Results for Cars)
Cars, Vehicles, Auto
Automotive, Transportation
Communication, Telephone
Wireless
12
10
Terms (concepts)
30'

FIG. 3

Building an Ecosystem of Activities for Personal Assistants
Genera Tasks
Personal Knowledge
Activities for Personal Assistants
Searching
Harvesting
Organizing
Connecting
Tracking
Collaborating
Reporting
Professional interests
Personalized news
Travel
Finances
Local search
Education
Hobbies
Health issues
More...
Recommending restaurants I would like
Creating reports for work and school
Discovering if your invention already exists
Summarizing large websites and documents
Tracking news about a health issue
Collaborating with friends on travel plans
Generating a website for your group
Building a knowledge base about your expertise
Surveying the market for a purchase
More...

FIG. 4

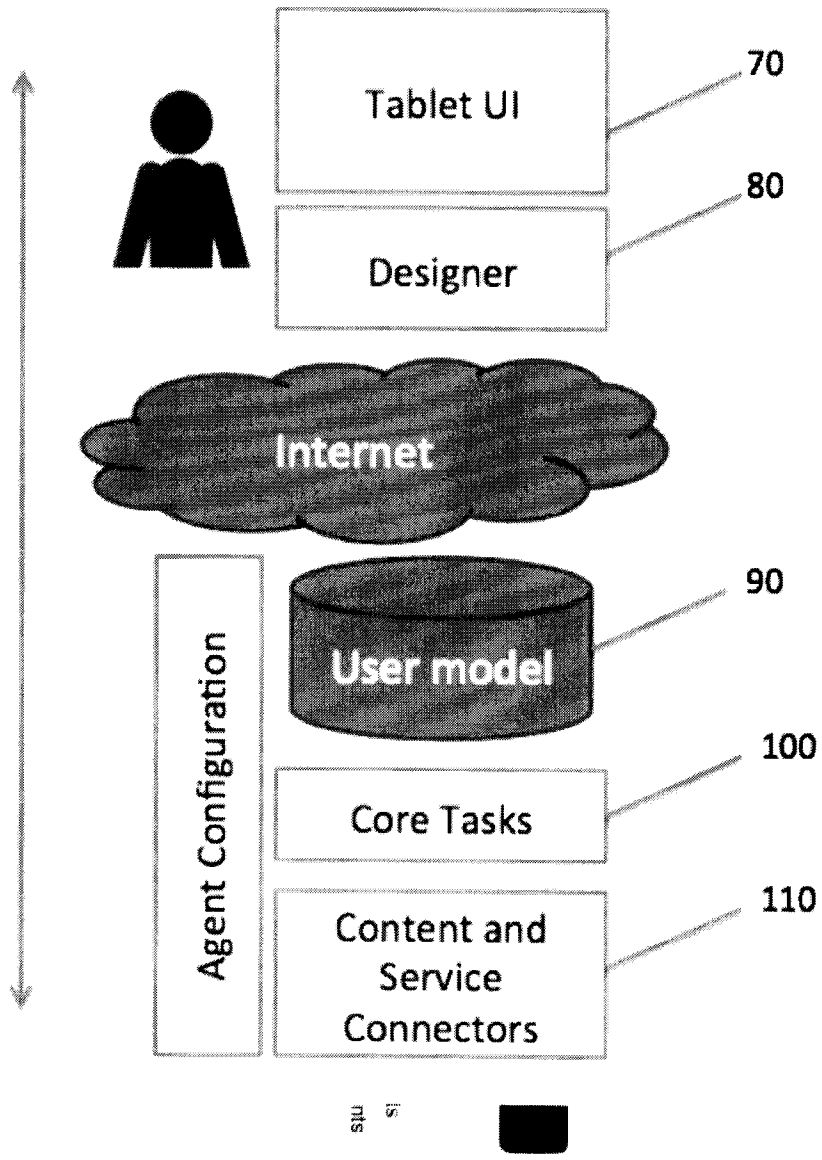
Tablet UI
70
Designer
80
Internet
User model
90
Agent Configuration
Core Tasks
100
Content and Service Connectors
110

FIG. 5

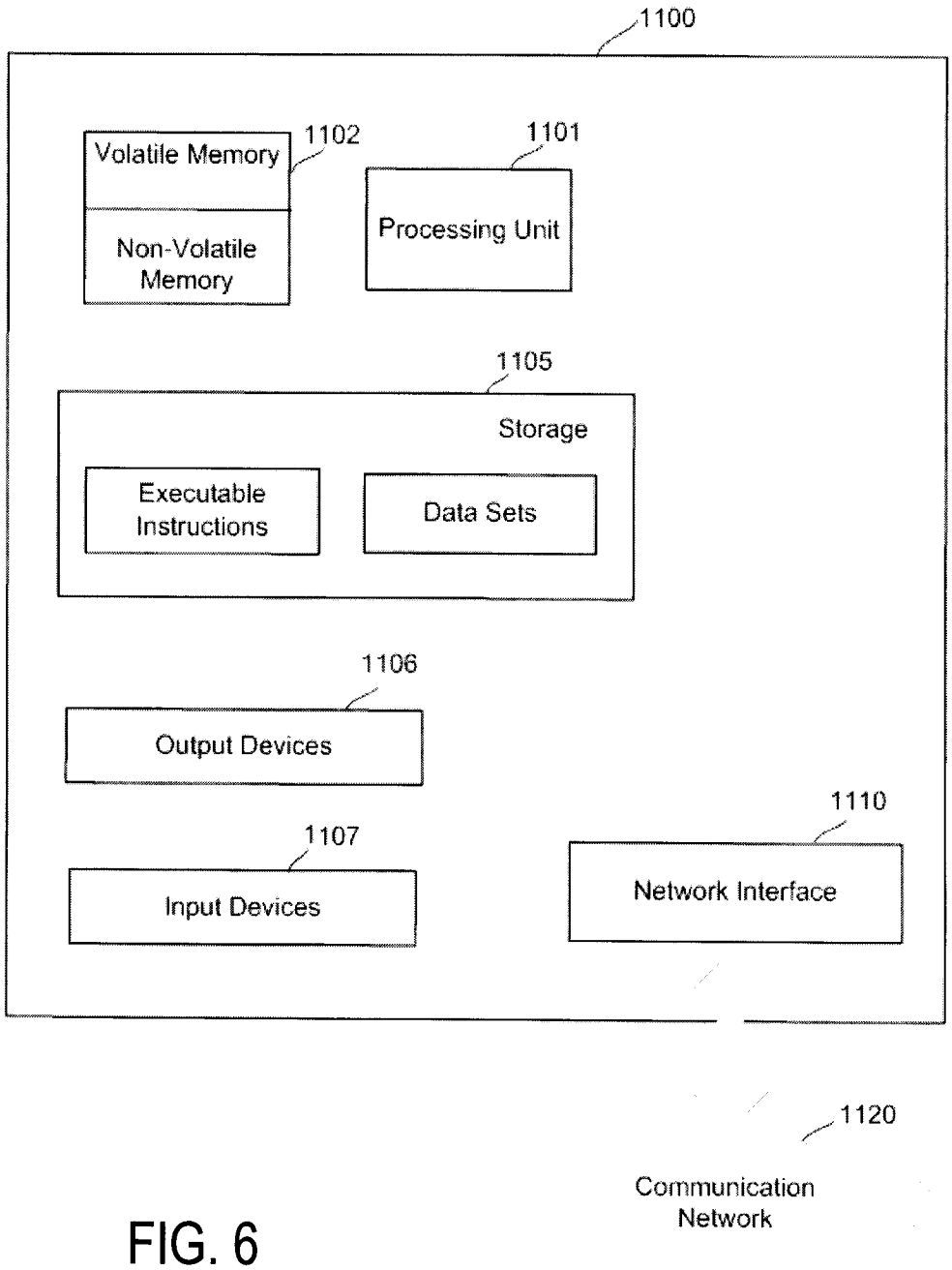
1100
Volatile Memory
1102
Non-Volatile Memory
1101
Processing Unit
1105
Storage
Executable Instructions
Data Sets
1106
Output Devices
1107
Input Devices
1110
Network Interface
1120
Communication Network

FIG. 6

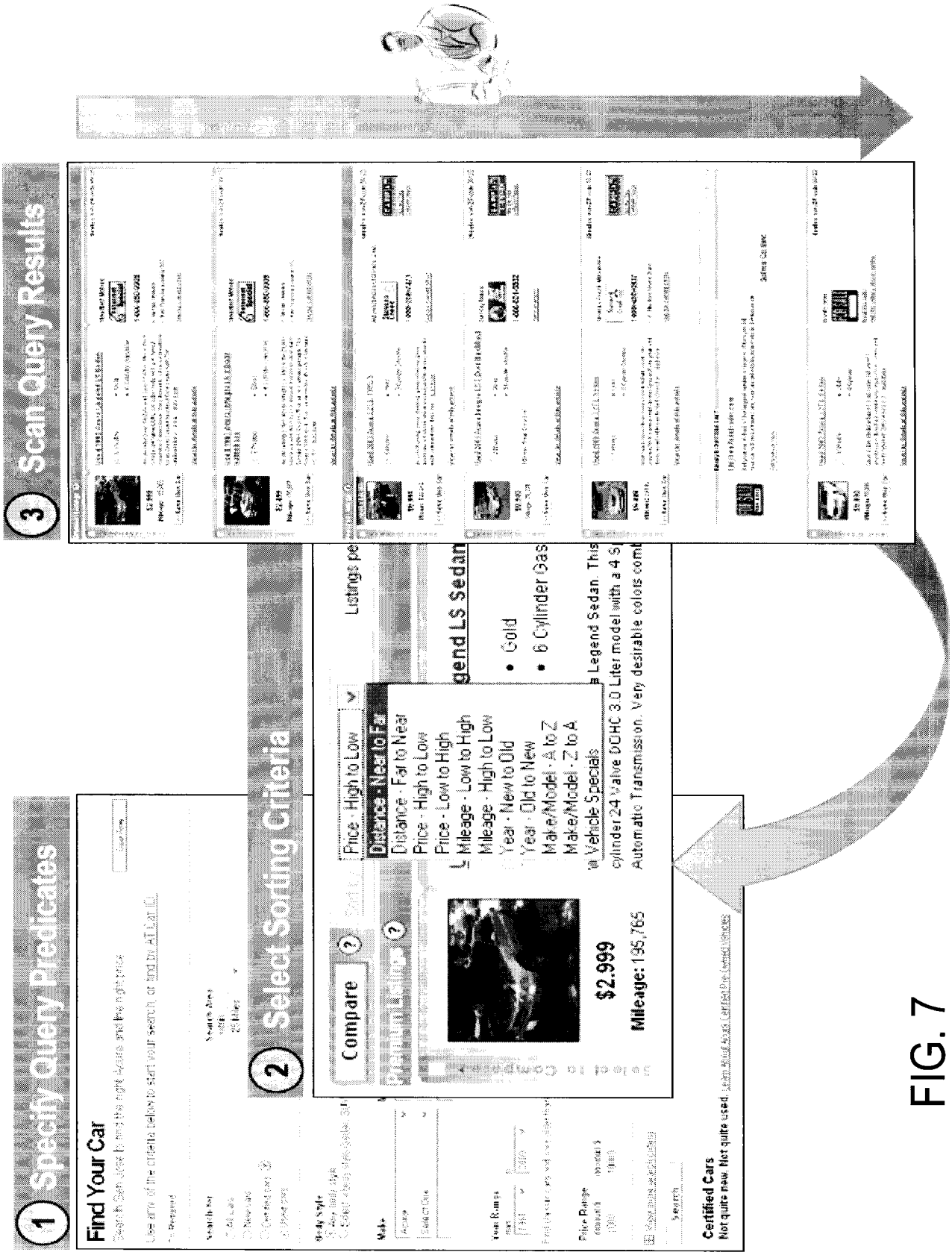
1 Specify Query Predicates
Find Your Car
2 Select Sorting Criteria
Compare
Price - High to Low
Distance - Near to Far
Distance - Far to Near
Price - High to Low
Price - Low to High
Mileage - Low to High
Mileage - High to Low
Year - New to Old
Year - Old to New
Make/Model - A to Z
Make/Model - Z to A
Gold
6 Cylinder Gas
Vehicle Specials
$2,999
Mileage: 135,765
Certified Cars
3 Scan Query Results

FIG. 7

ILLUSTRATIVE RELATION CAR

ID	Make	Model	Color	Price	Deposit
t_1	Honda	Civic	Red	1800	500
t_2	Honda	Odyssey	Blue	1500	300
t_3	Jeep	Liberty	Black	5200	700
t_4	Jeep	Wrangler	Red	5000	600
t_5	Ford	Focus	Black	5100	600
t_6	Ford	Mustang	White	1700	400

FIG. 8

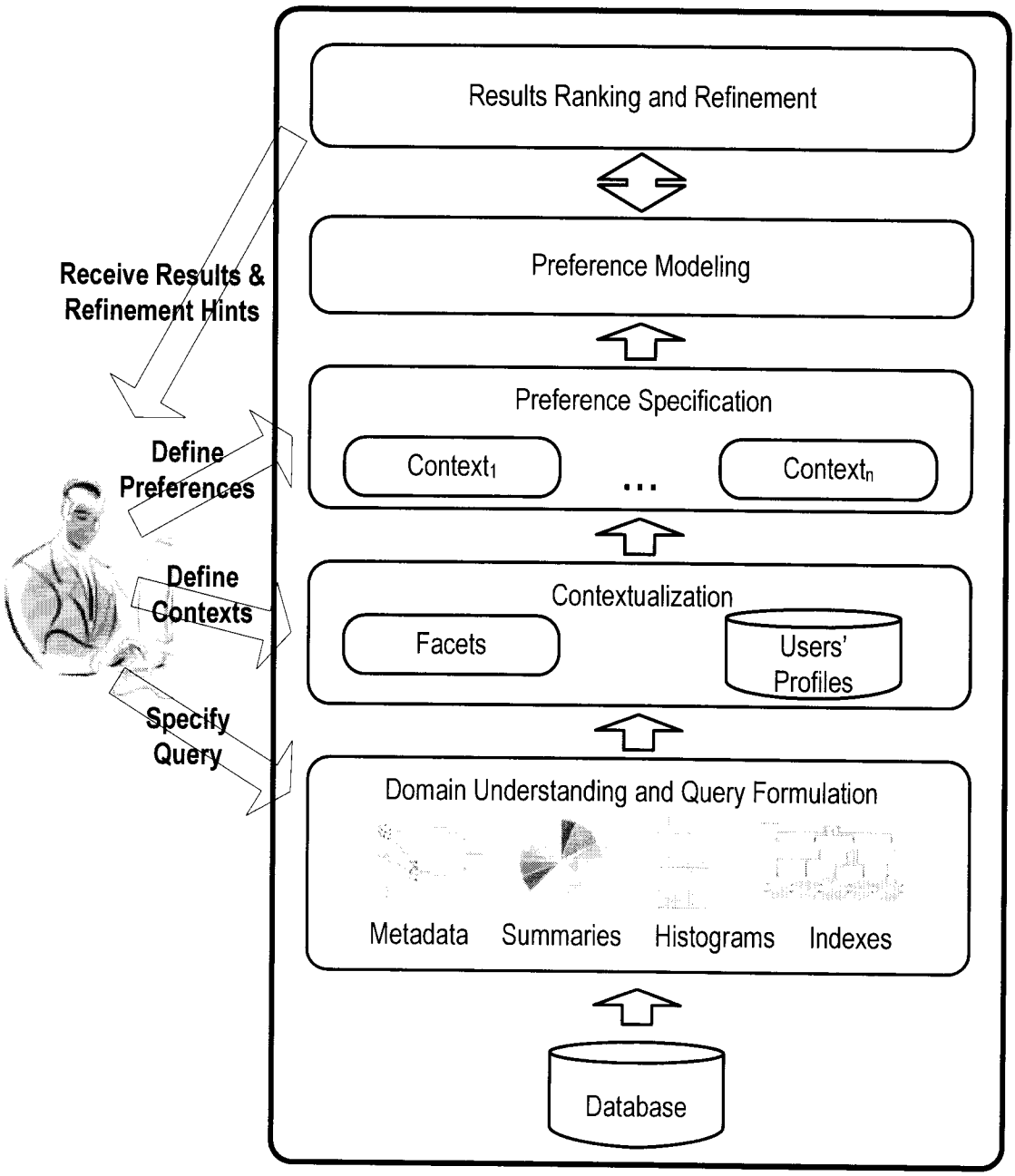
Results Ranking and Refinement
Preference Modeling
Preference Specification
Context1
...
Contextn
Contextualization
Facets
Users' Profiles
Domain Understanding and Query Formulation
Metadata
Summaries
Histograms
Indexes
Database
Receive Results & Refinement Hints
Define Preferences
Define Contexts
Specify Query

FIG. 9

ILLUSTRATIVE SCOPES OBTAINED FROM RELATION CAR

R_1: Select * from R where Make='Honda'
R_2: Select * from R where Make='Ford'
R_3: Select * from R where Color='Red'
R_4: Select * from R where Color='Black'
R_5: Select * from R where Price > 2000
R_6: Select * from R where Price < 2000

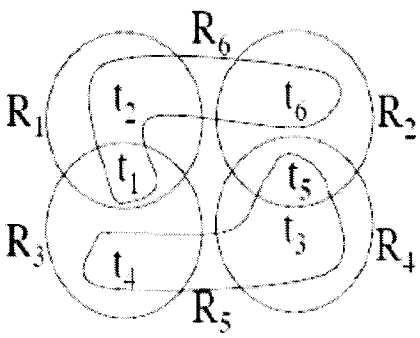

FIG. 10

ILLUSTRATIVE SCOPE COMPARATORS

$f_{1,2}$

```
if (t1.Price <t2.Price)
        return 1
else if (t1.Price > t2.Price)
        return -1
else
        return ⊥
```

$f_{5,6}$

```
if (t5.Color = 'Red' ^ t6.Color = 'Blue')
        return 1
else if (t6.Make = 'Ford')
        return -1
else
        return ⊥
```

$f_{3,4}$

```
return 1
```

$f_{6,2}$

```
if (t6.Make = 'Honda')
        return 1
else if (t6.Price < t2.Price)
        return -1
else
        return ⊥
```

$f_{1,5}$

```
return -1
```

FIG. 11

CONJOINT PREFERENCES EXAMPLE

Color

Make	Red	Blue
Honda	1	3
Toyota	2	4

Make-Color preferences

Price

Make	<=2000	>2000
Honda	1	3
Toyota	2	4

Make-Price preferences

Color

Make	Red (50)	Blue (35)
Honda (40)	1 (90)	3 (75)
Toyota (30)	2 (80)	4 (65)

Make-Color utilities

Price

Make	<=2000 (30)	>2000 (10)
Honda (40)	1 (70)	3 (50)
Toyota (30)	2 (60)	4 (40)

Make-Price utilities

FIG. 12

ILLUSTRATIVE MAPPING OF A PARTIAL ORDER TO LINEAR EXTENSIONS

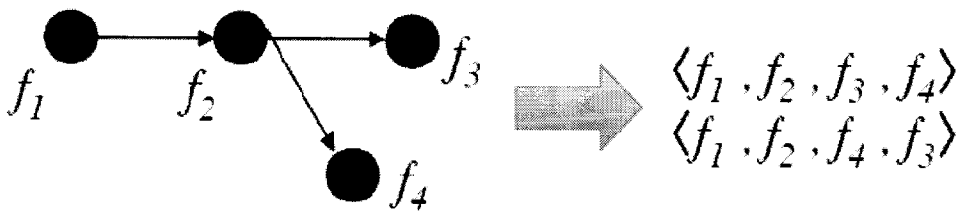

FIG. 13

ILLUSTRATIVE REFERENCE GRAPH

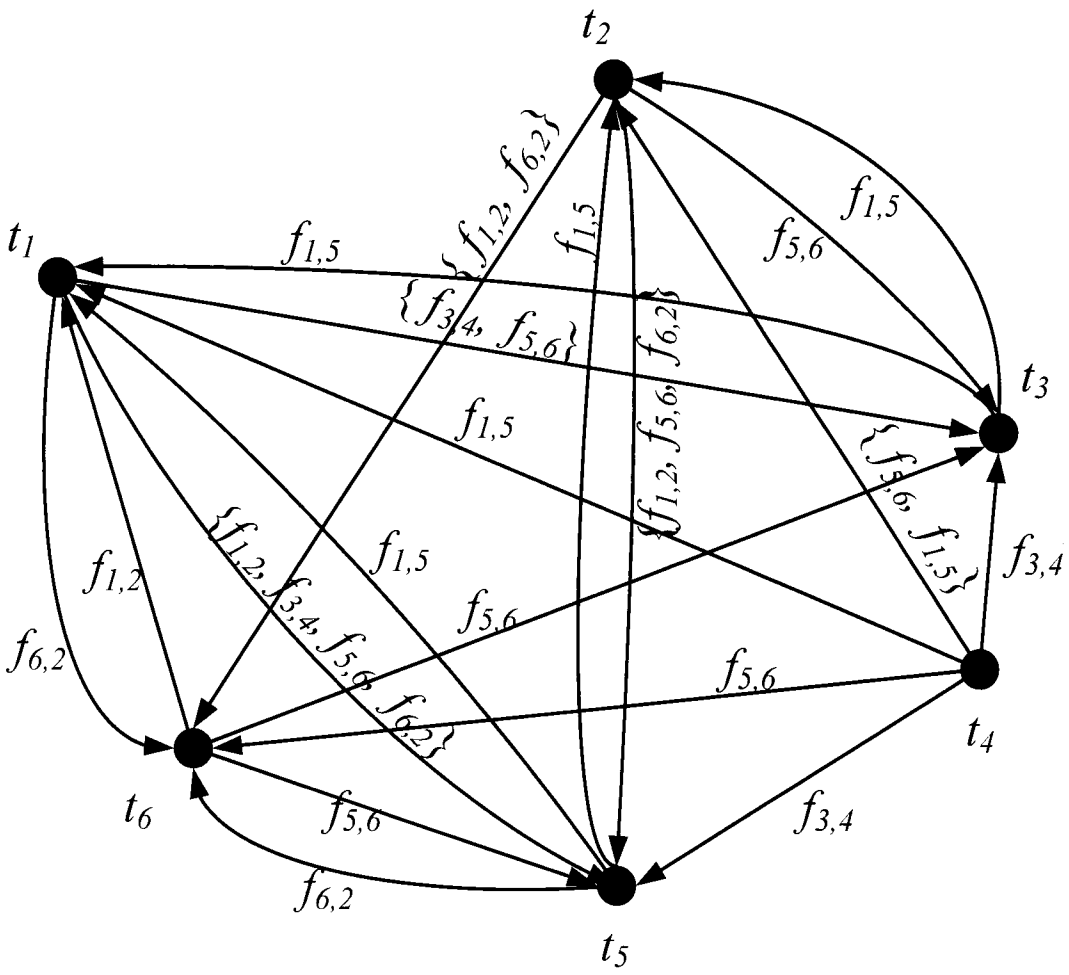

FIG. 14

ILLUSTRATIVE COMPUTATION OF EDGE WEIGHTS FOR DIFFERENT TYPES OF SECOND-ORDER PREFERENCES

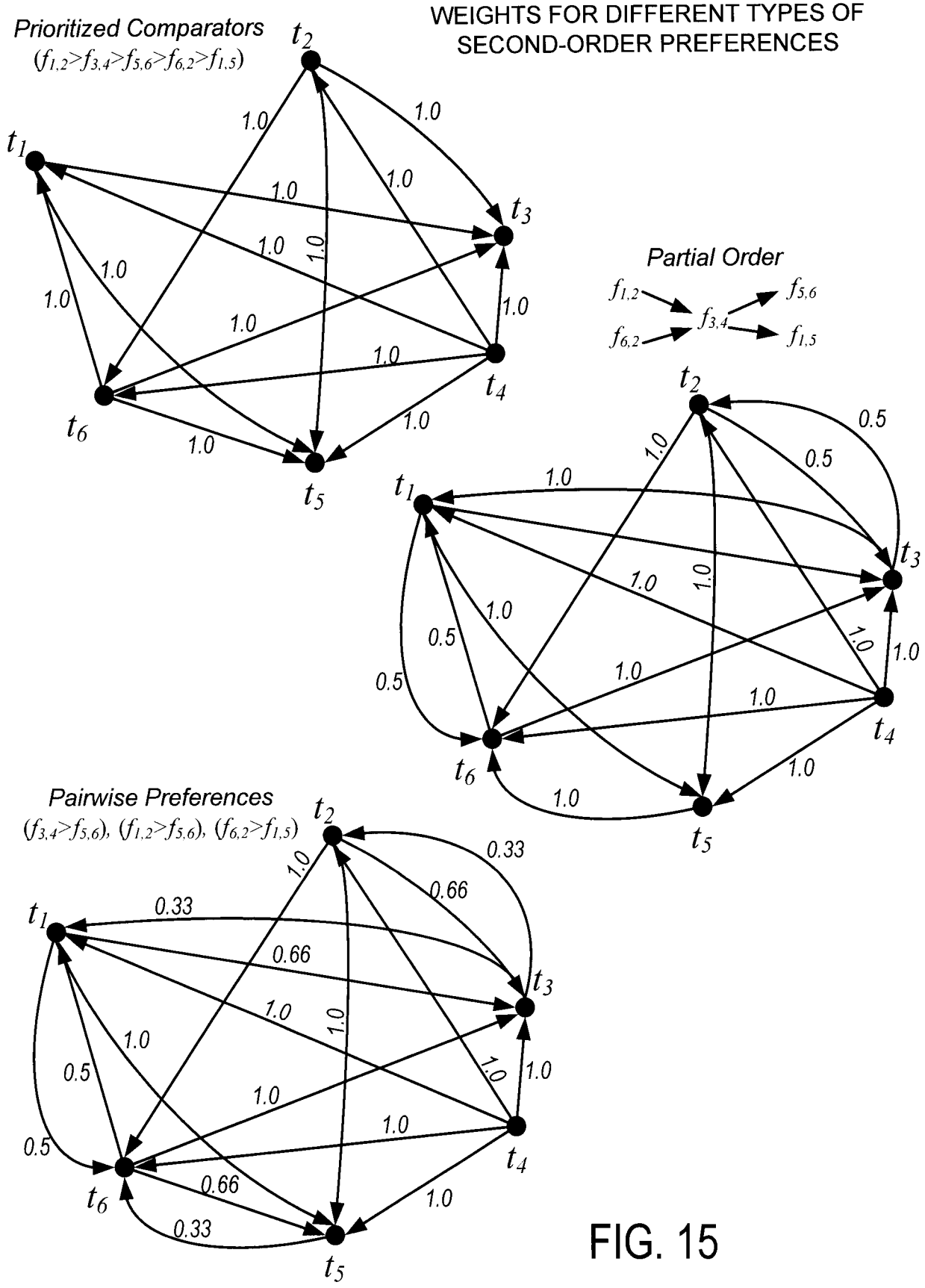

FIG. 15

ILLUSTRATIVE PAGERANK-BASED MATRIX FOR PRIORITIZED COMPARATORS

$$\begin{pmatrix}\gamma_1\\ \gamma_2\\ \gamma_3\\ \gamma_4\\ \gamma_5\\ \gamma_6\end{pmatrix}^{T} = \alpha\begin{pmatrix}0 & 0 & 0.25 & 0 & 0.25 & 0\\ 0 & 0 & 0.25 & 0 & 0.25 & 0.5\\ 0 & 0 & 0 & 0 & 0 & 0\\ 0.5 & 1.0 & 0.25 & 1.0 & 0.25 & 0.5\\ 0 & 0 & 0 & 0 & 0 & 0\\ 0.5 & 0 & 0.25 & 0 & 0.25 & 0\end{pmatrix} * \begin{pmatrix}\gamma_1\\ \gamma_2\\ \gamma_3\\ \gamma_4\\ \gamma_5\\ \gamma_6\end{pmatrix}^{T-1} + (1-\alpha)\begin{pmatrix}1/6\\ 1/6\\ 1/6\\ 1/6\\ 1/6\\ 1/6\end{pmatrix}$$

FIG. 16

ILLUSTRATIVE WEIGHTED PREFRENCE GRAPH AND DERIVED TOURNAMENTS

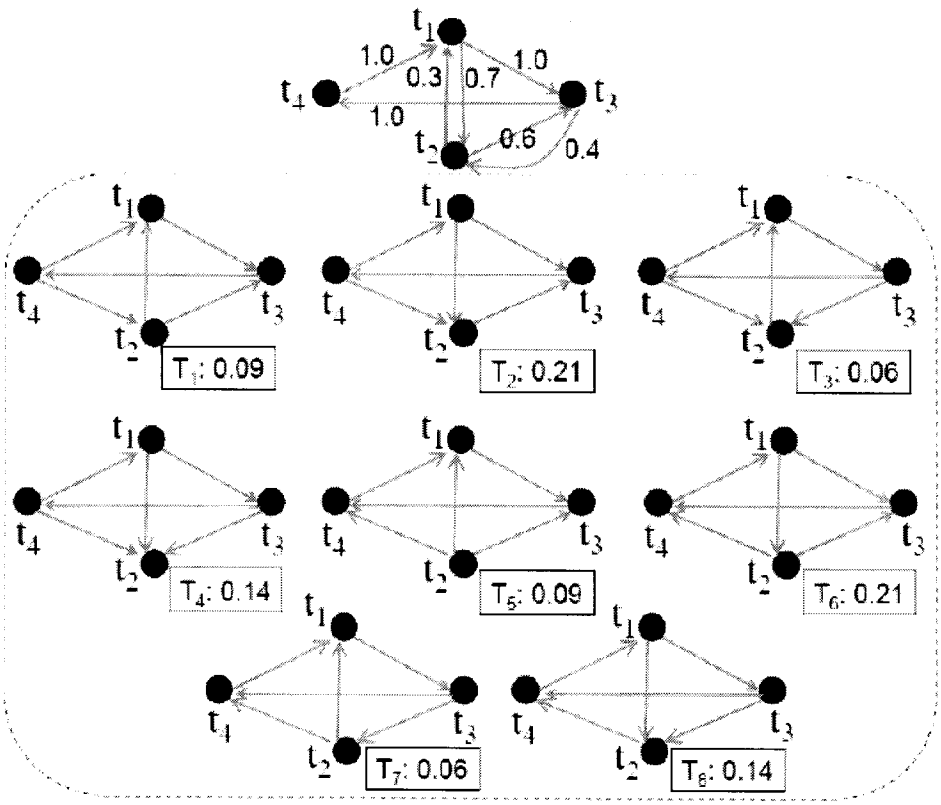

FIG. 17

ILLUSTRATIVE PROCESS FOR INTERACTIVE PREFERENCE SPECIFICATION

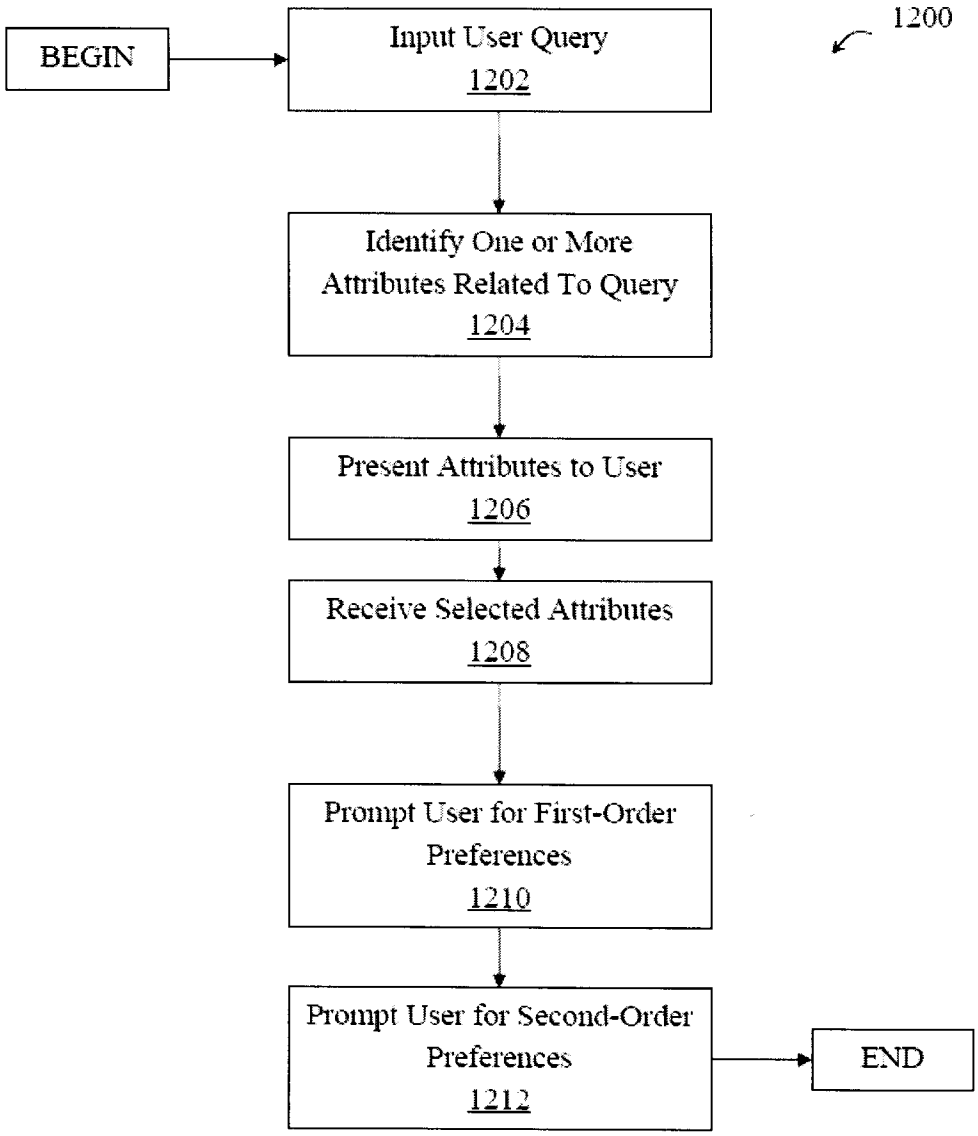

FIG. 18

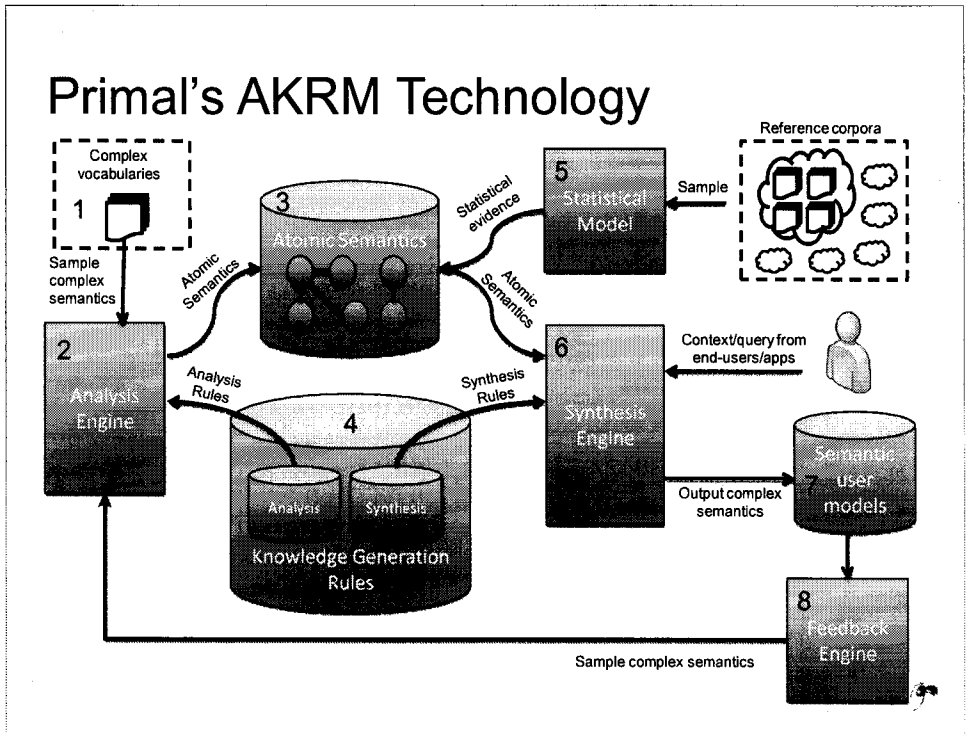
Primal's AKRM Technology
Complex vocabularies
1
Sample complex semantics
2
Analysis Engine
Atomic Semantics
3
Atomic Semantics
Statistical evidence
5
Statistical Model
Sample
Reference corpora
Atomic Semantics
6
Synthesis Engine
Context/query from end-users/apps
Analysis Rules
Synthesis Rules
4
Analysis
Synthesis
Knowledge Generation Rules
Output complex semantics
7
Semantic user models
8
Feedback Engine
Sample complex semantics

FIG. 19

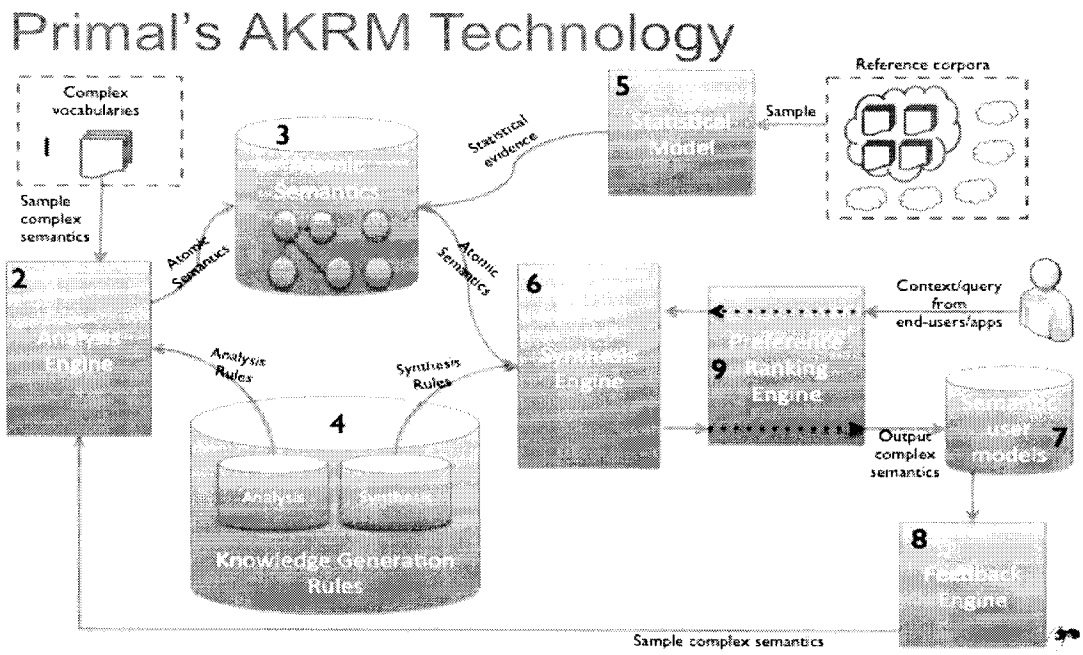
Primal's AKRM Technology
Complex vocabularies
1
Sample complex semantics
2
Analysis Engine
3
Semantics
Statistical evidence
5
Sample
Reference corpora
Atomic Semantics
Atomic Semantics
6
Synthesis Engine
Context/query from end-users/apps
9
Ranking Engine
Output complex semantics
7
models
Analysis Rules
Synthesis Rules
4
Knowledge Generation Rules
8
Feedback Engine
Sample complex semantics

FIG. 20

INTERNATIONAL SEARCH REPORT

International application No.
PCT/CA2011/000719

A. CLASSIFICATION OF SUBJECT MATTER
IPC: ***H04L 12/28*** (2006.01) , ***G06F 17/30*** (2006.01)
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC (2006.01): H04L 12/28, G06F 17/30

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

Electronic database(s) consulted during the international search (name of database(s) and, where practicable, search terms used)
Databases: EPOQUE (EPODOC); Canadian Patent Database; Google; IEEE
Keywords: semantic; network; agent; automatic; autonomous; ontolog; interfac; search; edge; weight; fee; processor; memor; exchang; perfer; order; data; structure; thought; web; harvest; mind; concept; synthesi; monetiz; primal fusion; sweeney

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X Y	WO2009132442 A1 (Sweeney et al.) 05 November 2009 (05-11-2009) * Figs. 1, 5 and 7 * page 2 lines 5-11 * page 3 line 22-page 4 line 30 * page 6 lines 26-28 * page 7 line 19-page 8 line 2 * page 9 line 1-page 10 line 13 * page 26 lines 5-8 * page 28 line 24-page 30 line 2 * page 35 line 10-page 36 line 26 * page 41 line 22-page 42 line 12	1-3, 5-14, 16-25, 28 and 29 4, 15, 26 and 27
Y	US20080154906 A1 (McDavid et al.) 26 June 2008 (26-06-2008) * Fig. 1 * paragraphs [0012, 0021, 0028, 0038]	4, 15, 26 and 27

[X] Further documents are listed in the continuation of Box

[X] 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 but published on or after the international filing date

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

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

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

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

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

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

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
19 September 2011 (19-09-2011)	28 September 2011 (28-09-2011)

Name and mailing address of the ISA/CA	Authorized officer
Canadian Intellectual Property Office Place du Portage I, C114 - 1st Floor, Box PCT 50 Victoria Street Gatineau, Quebec K1A 0C9 Facsimile No.: 001-819-953-2476	Leslie Yeow (819) 934-0345

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT		
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	CA2734756 A1 (Sweeney et al.) 04 March 2010 (04-03-2010) * pages 1-8	
A	Payne et al. (June 2002) Calendar Agents on the Semantic Web. *IEEE Intelligent Systems*. Vol. 17 (3) * pages 84-86	

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/CA2011/000719

Patent Document Cited in Search Report	Publication Date	Patent Family Member(s)	Publication Date
WO2009132442A1	05 November 2009 (05-11-2009)	CA2723179A1	05 November 2009 (05-11-2009)
		CN102016887A	13 April 2011 (13-04-2011)
		EP2300966A1	30 March 2011 (30-03-2011)
		IL208603D0	30 December 2010 (30-12-2010)
		JP2011521325A	21 July 2011 (21-07-2011)
		US2010235307A1	16 September 2010 (16-09-2010)
US2008154906A1	26 June 2008 (26-06-2008)	US2008154906A1	26 June 2008 (26-06-2008)
		US2008177870A1	24 July 2008 (24-07-2008)
CA2734756A1	04 March 2010 (04-03-2010)	CA2734756A1	04 March 2010 (04-03-2010)
		CN102177514A	07 September 2011 (07-09-2011)
		EP2329406A1	08 June 2011 (08-06-2011)
		IL211242D0	28 April 2011 (28-04-2011)
		US2010057664A1	04 March 2010 (04-03-2010)
		WO2010022505A1	04 March 2010 (04-03-2010)