PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F 15/18, 17/30, 17/60, G06N 5/02, G05B 13/02

A1

(11) International Publication Number: WO 00/70481

(43) International Publication Date: 23 November 2000 (23.11.00)

(21) International Application Number: PCT/US00/13360

(22) International Filing Date: 15 May 2000 (15.05.00)

(30) Priority Data:
60/134,105 14 May 1999 (14.05.99) US

(71) Applicant: MANNA, INC. [US/US]; 27 Mica Lane, Wellesley, MA 02481 (US).

(72) Inventors: RATSABY, Joel; 59 Gordon Street, Tel Aviv (IL). BARNEA, Gad; 17 Einstein Street, Haifa (IL).

(74) Agent: KUSMER, Toby, H.; McDermott, Will & Emery, 28 State Street, Boston, MA 02109 (US).

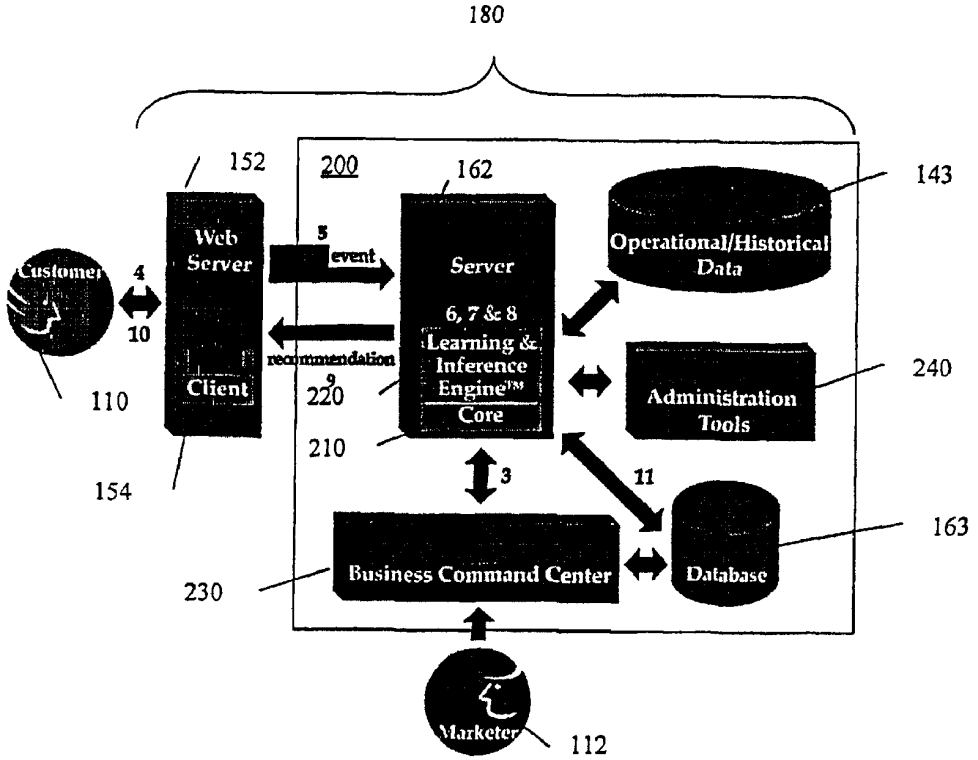
(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.
With amended claims.

(54) Title: INTELLIGENT COMPUTER SYSTEM

(57) Abstract

An intelligent computer system (180) is capable of real time learning, inference and updating of user profiles and trends to facilitate on-line, real-time recommendations or answer queries related to the user, referred to as "personalization", and may be added to, or integrated with, any of a variety of distributed computer systems (102, 104, 106, 108) to add dynamic personalization capability thereto. The intelligent computer system (180) distributes it processes across the system as processors become available and as a function of the loads across the system. The intelligent computer system (180) includes a business command center (230) for generating and maintaining enterprise specific rules, which may be entered by a non-technical enteprise user. The system also includes a core (210) that serves as an operating system and includes functionality to evaluate which rules should be applied to a consumer and his session. An artificial intelligence system includes a machine learning system for generating and updating Bayesian models, using off-line and on-line processes, which are used by its inference system to make intelligent recommendations related to the user.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

INTELLIGENT COMPUTER SYSTEM

Cross References to Related Applications

This application claims the benefit of priority from U.S. Provisional Application Serial Number 60/134,105, entitled INTELLIGENT E-COMMERCE DISTRIBUTED COMPUTER SYSTEM, filed May 14, 1999, incorporated herein by reference.

Field of the Invention

The invention relates generally to distributed computer systems accessible by a plurality of users. More specifically, the present invention relates to intelligent distributed computer systems that make recommendations or answer queries related to users (or virtual sessions that represent users either online or offline) as a function of what the system learns related to those users.

Background of the Invention

To an increasing degree, whether it is on the World Wide Web (the Web) or on other networks, network-based computer systems seek to go beyond the static, pre-scripted, or predetermined presentation of information to computer users. As an example, some systems now attempt to provide secondary or ancillary information, such as marketing information, to a user (e.g., a consumers engaged in e-commerce), wherein this secondary information is unsolicited. It is simply provided (in response to an event) in addition to the providing of the primary information sought by the user. In other instances, a system will attempt to determine and provide the primary information that the system deems relevant or optimal for the user. Still, in other instances, a system may attempt to provide information that is not directly presented to a user, but is in response to an event related to a user (or some other system activity). In order to determine what information to present or supply (whether primary, secondary, or characterized in some other way) in response to an event, some systems attempt to "learn" something about the user, and then selectively provide information as a result of a prediction based on what the system has learned or knows - sometimes referred to as "personalization". Systems that attempt to accomplish this vary, and so to does their effectiveness.

One approach to providing learning is to use collaborative filtering techniques, which employ various pattern matching algorithms and based on several heuristic metrics compute the "closeness" of a selected user's profile to other similar users. Collaborative filtering is a supervised learning approach. User profiles are constructed during a training phase where the

user rates selected items, e.g., movies, with a score in a supervised manner. Such supervised training approaches are time consuming and can pose an inconvenience for the user. Generally, collaborative filtering approaches can be viewed as learning by discovering clusters of similar user profiles and predicting the response for a selected user based on the cluster to which it belongs. Clustering is the most basic way of discovering interesting relationships between attributes (of profiles). For instance, discovering that high scores for certain movies are correlated with low scores of a selected movie.

Such techniques base their predictions or classification of the selected user using the profiles of similar users. In this respect, collaborative filtering is a basic form of case-based learning. It is based on algorithms that were developed in the 1960's and 1970's, specifically, the "weighted nearest-neighbor" algorithm and the "k-nearest neighbor" algorithm. These are non-parametric methods that rely on stored data for prediction. Because these approaches do not develop models of any sort, they require storing in memory of all the data cases in order to make predictions. In addition to high memory costs, significant computations are required at prediction time since the pattern similarity computations are done at recommendation time, rather than in advance. This consumption of system resources may undesirably limit the prediction response times.

Also, collaborative filtering approaches, and case-based learning systems in general, suffer from data over-fitting, which often leads to greater generalization and thus, lower predictive accuracy. Metrics used to measure closeness by such approaches suffer from needing all attributes for computing the similarity score. As a result, the distance between neighboring user profiles can often be dominated by the large number of irrelevant attributes, which can have a devastating effect on the accuracy of prediction. This is the well-known "curse of dimensionality" problem of non-parametric estimation and classification methods. Solutions to overcome this problem are heuristic-based and have little or no theoretical backing and hence are not failsafe, which is a critical for adaptive learning systems.

A different approach for making user-related predictions is to use neural networks. Neural networks are parametric models used to approximate multi-variable real valued mappings or functions. They can be used to learn probability functions. At a micro-level neural networks are composed of nodes which are graphically interconnected, with a moderate degree of approximation ability. On a macro-level, however, neural networks are not well suited for making predictions in a highly dynamic real-time environment.

Neural networks are like black-boxes, wherein an input layer and an output layer must be defined a priori to learning. After learning, only this single mapping can be used for inferences. The strengths between the internal nodes (also known as the hidden layers) which are learned by the neural networks are where the information lies. However, there is no easy way of interpreting this information other than requesting an output for a given input. Furthermore, neural networks cannot utilize available prior knowledge in addition to the input data. That is, a neural network is static with respect to time. If at some later stage new variables are added or deleted to the data then one needs to re-learn the neural network with these changed domain. This means that previously learned information will be lost.

Neural networks in some instances can learn incrementally, but using a heuristic approach as opposed to sound probability-based updating, which compromises the eventual results. However, since neural networks can not learn with partial incomplete data, they generally can not do "clustering" (or unsupervised learning). That is, multi-layered feed-forward neural networks cannot do clustering because they require complete data (this mode of learning is known as supervised learning).

For the most part, systems for making recommendations that are based on neural network technology use learning algorithms that are first generation (i.e., 1980's) machine learning algorithms. Although neural networks have strong predictive and estimation capabilities utilizing their inherent non-linear structure, the algorithms with which they learn are too slow for dynamic settings such as the ones encountered in today's distributed computer systems, e.g., on-line e-commerce systems. They require learning from batch data, which means they need to scan through each data case numerous times before converging to the optimal solution. As such, they do not scale efficiently with respect to the size of large, changing data sets. Also, neural networks are very hard to scale (if at all possible) mainly due to their black-box nature. For example, one cannot break a traditional single neural network into two networks, which limits the usefulness in a dynamic distributed computer architecture, where such scalability, if available, can be advantageous.

The application of collaborative filtering and/or neural networks to distributed, real-time computer systems comes with limitations. As a result, there is need for an intelligent, robust, scalable distributed intelligent computer system capable of learning and updating user profiles based on partial and changing data to facilitate on-line, real-time personalized

recommendations to such a user or supply dynamic and accurate predictions (answers to query) about him.

Summary of the Invention

The present invention is an intelligence system capable of learning trends and profiles related to a user based on, at least in part, the user's interaction with the system and possibly some domain related information or events to facilitate on-line, real-time production and communication of predictions and/or recommendations related to the user. That is, the intelligence system is characterized as having automated distributed intelligence (ADI), and is added to an enterprise computer system to add intelligence and personalization thereto, creating an intelligent enterprise system. Due to the intelligence system, the intelligent enterprise system has substantially no down-time, operates in real-time, and is structured to have unencumbered scalability, through the use of standardized and platform independent software entities and the ability of the intelligence system to monitor and redistribute its load dynamically.

The term “enterprise” is used generically to refer to any entity, organization or combination thereof having a computer system (i.e., the enterprise computer system). As one example, an enterprise may be a vendor of products and/or services engaged in e-commerce, wherein the access to the intelligent enterprise system may be controlled, as in a private network, or open, as in a publicly accessible e-commerce Web site. The intelligent enterprise system may be distributed over any of a variety of commonly known networks, such as a local area network (LAN), wide area network (WAN), world wide web (the "Web"), intranet, extranet, Internet, private network, or some combination thereof. In such a case, the intelligent enterprise system may include one or more servers and databases that may be accessed by multiple users simultaneously over a network by wired and/or wireless devices (e.g., personal computers, personal digital assistants, cellular telephones, and so on).

The enterprise computer system may take any of a variety of forms. For example, the enterprise computer system may be an application (e.g., a word processing application) running on a computer that includes an interface to the intelligence system for gaining a personalization capability. In another instance, the enterprise computer system may be an e-mail system having an interface with the intelligence system. In such a case, the e-mail system may include servers and databases and may link to an intelligence system having its own set of servers and databases, or the two systems may share servers and databases. In yet another

example, the enterprise system may be a video system wherein personalization from the intelligence system influences the video content presented to the user of the (combined) intelligent enterprise system. In another instance, the enterprise system may be a web application running in a browser with its backend running at a web server. The web application will link to the intelligence system.

The intelligence system is preferably implemented like a client - server architecture, having a set of server-side software modules and possibly a client-side software module (i.e., client module). The software modules of the present invention are, largely, implemented in accordance with object oriented design methodologies, but those skilled in the art will appreciate that other design methods may also be used to practice the present invention. The intelligence system preferably includes (or has access to) at least one wired or wireless intelligence system server (or computer) to host the intelligence system server-side software, along with at least one associated database. The client module may be hosted on any of a variety of types of wired or wireless electronic devices, such as a computer, server, or other device. Furthermore, the client module and server-side software may be hosted on the same platform, and with or without an enterprise application. That is, the distinction between the server-side software and client module of the intelligence system is primarily functional and logical, rather than physical.

The client module provides an interface between the enterprise application (e.g., e-mail, word processor, and so on) and the personalization functionality of the server-side. As an example the client module may be hosted on a front-end Web server, along with an enterprise Web-based application and the server-side software may be hosted on a back-end intelligence system server. However, generally the front-end server could be a Web server, an application server, or a wired or wireless communication server, as examples, or any other system, server, or device that seeks to take advantage of the personalization offered by intelligence system of the present invention. Furthermore, the front-end server may be integrated into the same platform as the back-end server, as discussed above.

The server-side software includes a business command center, a core module, an artificial intelligence (AI) module, and a set of administrative tools. The administrative tools include a business object developer for automated creation of business (i.e., enterprise) objects, which embody enterprise specific rules. The business command center may be run on the intelligence system (e.g., back-end) server and use the same database. In any event, the

business command center provides logic for an enterprise, non-technical user to generate and maintain enterprise specific rules (as objects). These rules are related to the goals, tasks, and processes necessary to carry out the enterprise's objectives for the system. For example, if the enterprise were an on-line food distributor, the rules may relate to enterprise product offerings, to shopping cart abandonment, and so on.

For the most part, the business command center implements standard browser-based functionality and "wizards" to allow ease of use by enterprise users. Rules may contain a variety of components, depending on the application of the intelligent enterprise system. In an e-commerce context, for example, the rules may include a "time frame" component indicating when the rule to be applied and a "situation" component that provides dynamic information about the current state of the session. Additionally, a "profile" component provides static information about the user and a "result" component indicates the nature of the system's response as a function of the previous components. The business command center is also configured to generate a variety of reports about the system, based on data it collects, that are useful to enterprise.

The core module and AI module are hosted and run on the intelligence system (e.g., back-end) server. The core module serves as an "operating system" to the intelligence system and provides centralized administration and processing functions. For example, the core module includes a messaging facility and offers dynamic views of available computer resources. Using a highly distributed architecture, plug-ins, load balancing, messaging, and events, the core module continually ensures maximum utilization of system resources and retains only those objects that relate to the active sessions. The core also includes a rule evaluator that evaluates information generated by an external event (e.g., a consumer purchase) according to the business rules established by the enterprise using the business command center.

The AI module provides for the real-time creation, maintenance and application of Bayesian models, which are used to make personalized recommendations and to infer answers for various queries. The Bayesian models are created and maintained using on-line and off-line processes as a function of previous and current user responses and the general state of the intelligent enterprise system. The Bayesian models and rules are applied to generate intelligent responses to user inputs, including providing to a user information (e.g., recommendations)

regarding a second subject matter that is related to a first subject matter being queried by the user.

The AI module, is the "brain" (or AI engine) of the intelligence system, and makes real-time recommendations based on user behavior models, continuously updating them with new information. The AI module, more than any other component, is what transforms a standard enterprise site into a dynamic learning and inference center. The AI module creates and employs intelligent virtual agents that are capable of automatically learning, utilizing and sharing learned knowledge to serve the client. The intelligence system's robust, scalable and distributed server architecture allows multiple virtual agents, learning independently, to form a unified intelligence entity, which acts as a single distributed "virtual brain".
Learning scalability enables the intelligence system (and, therefore, the intelligent enterprise system) to deal with the exponential growth of data and to learn large data sets rapidly. Learning takes place incrementally using off-line and on-line learning algorithms, which provide an expedient approach to dealing with enormous amounts of data. The weights and structure of models are updated in real-time, as soon as a new piece of data arrives. These incremental, real-time updates enable learning from massive databases or dynamic data without needing to access all the data at once or repetitively. Incremental learning also gives the intelligent enterprise system the ability to learn in a dynamic network environment, where information constantly changes.

User behavior models are implemented as Bayesian network models. Unlike other learning models (e.g., collaborative learning and neural networks), the Bayesian network models can work with incomplete data cases and levels of uncertainty. Once deterministic or probabilistic evidence has been obtained, the probability distribution for an attribute or combination of attributes can then be determined dynamically, in real-time. These user behavior models are reusable and can be combined to form new models, saving on both data resources and learning power. The model building process constructs attributes, defines their values and then gathers them into models according to logical relationships. These models are then inter-linked, based on their statistical coupling. In addition to profile attributes, models include enterprise subject matter categories and subcategories, such as products/services categories.

Bayesian network models built off-line and tailored by the intelligence system are customized for the enterprise's application, within the context of the intelligent enterprise

system. Off-line learning involves running several intelligent programs with several input data files, resulting in a set of models. Each model is an object file, in the preferred embodiment. Generally, a model includes variables and probability connections between those variables, which may be expressed in tables.

The models are updated during normal system operation to reflect knowledge gained through on-line learning. As user-related events occur, they are logged into a database. At preset intervals, the intelligent enterprise system opens the database and puts the information gathered into data files. The data files are continuously updated as new events occur. On-line learning processes the files, updating the values in the models' probability tables. This action permits an enterprise site to learn quickly and accurately about the trends and patterns of user behavior and activity.

The AI module is comprised of two main parts, a machine learning system (MLS) and an inference system (IS). The MLS automatically creates Bayesian Network Models based on consumers' past data. These models are then used by the IS as on-line intelligent resources. All of the intelligent operations preformed by the AI module can be obtained from these Bayesian models such as prediction, classification, and maximum expected utility optimization.

Both of the MLS and IS are structured using parallel distributed Java object-oriented code, allowing advantage to be taken of the distributed application server architecture. With parallel distribution, the processes of machine learning (using MLS) and inference (using IS) are scalable. This implies that for making inferences, the number of on-line sessions that require intelligent resources is scalable, while for machine learning the number of Bayesian network models is scalable with the number of computers (or servers).

Unlike collaborative filtering, the use of Bayesian Model networks is not a supervised learning approach, wherein user profiles are constructed during a training phase where the user rates selected items, e.g., movies, with a score in a supervised manner. In contrast, the Bayesian Model-based learning algorithms of the present invention can incorporate both supervised and unsupervised forms of learning. Through statistically based incremental learning algorithms, learning can take place even with partially incomplete data. User profiles containing dynamic information such as on-line browsing behavior are formed in an unsupervised manner without burdening users with questionnaires.

Unlike neural networks, Bayesian model networks are by no means black-boxes. Every node, or attribute, has a meaning and its value may be queried. A Bayesian model

network, in its whole, represents a joint probability distribution over the attributes of the domain where the functional form is depicted via its structure and probabilities. The learned structure depicts useful information in terms of inter-dependencies amongst these attributes. For instance, the fact that one Bayesian model attribute is linked to another attribute implies that there is a statistically significant (based on the data) relationship between the two variables. Moreover, no prior input-output mapping needs to be defined at the start of learning. Any attribute(s) may serve as an input or output at any particular query time. For this reason, in a dynamic environment where some of the input information may be missing, Bayesian model networks provide benefits over neural networks. The implementation of Bayesian model networks in the present invention causes the system to accumulate available observations (or evidence) and then compound this evidence in a statistically sound manner, and produce the best (in a probabilistic sense) output or recommendation. Additionally, a Bayesian model network can incrementally change both its structure and its probabilities based on a new domain and on new data. New attributes can be added and old ones can be deleted with minor loss of probabilistic interdependency information.

Brief Description of the Drawings

The foregoing and other objects of this invention, the various features thereof, as well as the invention itself, may be more fully understood from the following description, when read together with the accompanying drawings, which include:

Figure 1 is a representative computer architecture using the intelligent enterprise system, in accordance with the present invention;

Figure 2 is a program architecture for the intelligence system of Figure 1;

Figure 3 is a representative graphical user interface of the business command center of the system of Figure 2;

Figure 4 is a flowchart depicting the process of rule evaluation in accordance with the present invention;

Figure 5 is a block diagram of an artificial intelligence module, as part of the intelligent enterprise system of the present invention;

Figures 6A and 6B are examples of a representative product database file and portions of a corresponding representative serial file, respectively, used by the machine learning system of the AI module of Figure 5;

Figures 7A and 7B show examples of logical and topological linking models, respectively, of the IS and MLS of Figure 5;

Figure 8 is a representative hyperlink file generated by the MLS of Figure 5;

Figure 9 is an example of a serial session data file generated and used by the MLS of Figure 5;

Figure 10 is an object-oriented depiction of a DRV model, used by the MLS of Figure 5;

Figure 11 is a table depicting the on-line learning modules of the MLS of Figure 5;

Figures 12A and 12B are block diagrams of an architecture of the MLS of Figure 5;

Figures 13A and 13B provide a table depicting the MLS components and their interactions;

Figure 14 is a table depicting messaging objects used by the MLS, in accordance with the present invention;

Figures 15A-15C provide a table of the components of the IS of Figure 5 and their interactions;

Figure 16 is a table depicting messaging objects used by the IS, in accordance with the present invention;

Figure 17 is a representative block diagram of the IS of Figure 5;

Figure 18 is an example of a Bayesian network model, in accordance with the present invention;

Figure 19 is an example of a Bayesian model in accordance with the present invention;

Figure 20 is an example of a category change event file generated and used by the IS of Figure 5;

Figures 21A-12E are samples of QRC™ agent structures of various APIs supported by the IS;

Figure 22 is a sample recommended list returned by an IS inference agent; and

Figures 23-26 depict various absorptions techniques of evidence among models.

For the most part, and as will be apparent when referring to the figures, when an item is used unchanged in more than one figure, it is identified by the same alphanumeric reference indicator in all figures.

Detailed Description of the Preferred Embodiments

The present invention is an intelligence system capable of real-time inference and learning, and updating of user profiles and trends to facilitate on-line, real-time recommendations or answer queries related to the user, referred to as "personalization". As will be appreciated by those skilled in the art, the intelligent computer system may be added to, or integrated with, any of a variety of enterprise computer systems to add dynamic personalization capability thereto, resulting in an intelligent enterprise system. As examples, the enterprise system may be an application (e.g., a word processor, or e-mail personalization server), a network-based application with a presentation device (like a video tour on a kiosk that is personalized), or an e-commerce system. An intelligent enterprise system in accordance with the present invention may include at least three characteristics. First, the intelligent enterprise system automatically develops Bayesian models and uses them to generate intelligent responses to system events. Second, the intelligent e- enterprise system accesses data and responds to events throughout the system. Third, the intelligent e- enterprise system processes, analyzes, and applies data according to rules defined by an enterprise user (i.e., enterprise rules).

Generally, an enterprise computer system be any of a variety of systems and may include or be accessible by a network, such as, for example, a LAN, a WAN, an intranet, an extranet, a private network, the Internet, or the World Wide Web (the "Web"), or some combination thereof. Representatives of the enterprise which configure the system for a particular enterprise's needs are referred to as “enterprise users” and may include, for example, non-technical sales and marketing staff. Meanwhile, any of a variety of types of entities may generates and/or receive events to and from the intelligence system. Such other types of entities may include an e-commerce consumer searching for at least one primary product or service, a non e-commerce user (e.g., a cellular telephone user) searching for information, or a virtual entity (e.g., an application process or system seeking information). Depending on the configuration of the intelligent enterprise system, and the context of the enterprise computer system, the intelligent enterprise system may support events related to a variety of types of entities.

In the preferred embodiment, for illustrative purposes, the intelligence system is added to an e-commerce system of an enterprise to add, among other things, intelligence thereto, resulting in an intelligent e-commerce system. As an e-commerce context example, when a

consumer seeks a primary product, the intelligent e-commerce system may provide recommendations related to one or more secondary products or services, or predict the answer for a query about the consumer that could result in a dynamic site change (like offering an incentive). A secondary product or service is one not directly searched by the consumer, and may or may not be related to the primary product or service that was searched by the consumer. For example, if the consumer searched the intelligent e-commerce system for bread as a primary product, a secondary (related) product may be a certain cheese from a certain maker or supplier, but another (unrelated) secondary product may be a particular automobile. The particular secondary products or services offered or recommended are a result of the intelligence provided by the intelligent computer system, as it continually updates consumer profiles and applies probabilistic models relating to the products and services sought by the consumer. However, predictions and recommendations are not confined to the secondary products (i.e., information). The intelligence system can make predictions and recommendations regarding and user related information (i.e., primary, secondary, or characterized in some other way). Furthermore, the predictions and recommendations need not relate (directly) to a user's request for information. Rather, such predictions and requests are made in response to events generated and somehow related to said user.

In the illustrative embodiment of the present invention, the intelligent enterprise system is implemented on a distributed computer architecture 100, as shown in Figure 1. In such an architecture, consumers, or other types of users, may access the intelligent enterprise system 180 via any of a variety of known means and with any of a variety of known wired or wireless devices. For example, a software application (or system) 101 may access the intelligent enterprise system 180 by accessing an application server 110 via a network, represented by network cloud 118. As another example, such devices include a personal computer 102, a laptop computer 104, a personal digital assistant 106, which are shown in Figure 1 as accessing a Web server 152 via the Internet and World Wide Web (the Web), represented by cloud 120. Any of these devices may also include an application (like application 101) that access server 162 via the Internet and Web 120 and server 152. As yet another example, a user can access the intelligent enterprise system 180 using a typical telephone or a cellular telephone, which are shown accessing telephone server 114 via a telephone network, represented as network cloud 122. In each case, server 110, 152, and 114 access an

intelligence system server 162 via a wired or wireless means. All separation is logical and in theory 162 and 152 could run on the same computer.

The intelligent enterprise system 180 includes the intelligence system integrated with the enterprise (e.g., e-commerce) system, each of which may include one or more servers and associated databases. In some configurations, the intelligence and enterprise systems may share servers and databases. In the configuration of Figure 1, application server 110, Web server 152, and/or telephone server 114 (as examples) may be front-end servers, through which entities access the intelligent system 180. One or more intelligence system (e.g., back-end) servers and databases (e.g., server 162 and database 163) service requests or events related to activity by entities interacting with the intelligent enterprise system 180, and received via one of the front-end servers. A database server 142 (or another relevant server or system) may be linked to the intelligent system server 162 locally via a LAN or remotely via a WAN as indicated by link 126, as examples. Note that database server 142, being a general site DB server, could be also connected to 110, 152, 114.

Figure 2 shows a representative architecture for the intelligent enterprise system 180. As mentioned previously, in the illustrative embodiment, the intelligence system 200 integrates with an e-commerce solution, tying into an e-commerce Web server 152 to obtain real-time information related consumer 110 click-stream behavior or other site information, as well as tying into existing database information (e.g., database 143), which may include information such as consumer demographics and buying behavior. In the preferred embodiment, the Web server 152 hosts an intelligence system 200 client module (the "client") 154. Client 154 interacts with intelligence system server 162, which hosts various intelligence system program modules, including a core module 210, an artificial intelligence (AI) module 220 and a business command center (BCC) module 230.

Preferably, an enterprise user 112 creates and updates business rules using the business command center module 230 and the rules are stored in the intelligence system database 163 and published (i.e., made available) to the intelligence system server 162. Thereafter, consumer 110 accesses the enterprise's e-commerce Web site through the Web server 152 to interact with the intelligent e-commerce system 180. Beyond consumer activity, the intelligence system 200 may also be responsive to events generated by other (non-consumer) entities. In response to consumer activity, the client 154 sends related events to and receives results (e.g., recommendations) from the intelligence system server 162. Client 154 can be

deployed in any of a variety of known manners, for example, using ActiveX, Servlets or Sockets, depending on the system tools and platform. When using sockets, the client 154 is actually nonexistent and the e-commerce system communicates directly with 162. As the consumer 110 interacts with the system 180, the core 210, i.e., the intelligence system's "operating system", "listens" for occurrences of selected events resulting from the consumer's click-stream activity.

Using a rule evaluator, which is a component of core 210, an event is sent to all active rules to determine which rules are relevant for the given consumer 110. The rule evaluator calls on AI module 220, as needed, to determine how to apply each rule for the given consumer. In addition to click-stream data, the AI module 220 can utilize operational and historical databases 143 for specific consumer/product data. Following rule evaluation, core 210 sends a resulting recommendation for personalized products, services or other content, as examples, to client 154. In turn, client 154 communicates the recommendation to Web server 152, which in turn changes the Web page and passes a new one to consumer 110. In the background, core 210 executes basic functions such as load balancing, rule creation and messaging. The core 210 reads and updates data, including rules and configuration information, which are stored as XML files in the intelligence system database 163.

The intelligence system 200 (integrated) components combine to offer an enterprise a comprehensive, scalable and customizable on-line personalization tool, including the five components: business command center 230 , intelligence system server 162 (the actual personalization server), intelligence system client 154, intelligence system database 163, and a set of administration tools 240. Working together, these components enable the enterprise to create and test specific e-commerce initiatives, interact with and learn from consumers and their preferences, as well as analyze and report results.

1. Business Command Center (BCC) 230

The business command center 230 allows an enterprise user to create, pre-test, update, and evaluate the impact of their intended e-commerce initiatives using their own defined business rules. Preferably, this interaction takes place via a standard Web browser (e.g., Internet Explorer by Microsoft Corporation of Redmond, WA) and "wizards", in an easy to use windowing environment, without requiring support from technical personnel (e.g., computer programmers or information technology personnel). Among other things, the

business command center 230 provides in-depth reports and analyses offering enterprise users 112 the critical information necessary for making effective decisions in real-time mode.

As an example, a business command center main screen 300 is shown in Figure 3, which may be used for building and editing enterprise defined rules. Rules are built using business objects created using a business object developer (BOD), discussed with respect to the administration tools 240 below. A business object provides a tangible expression for (or embodies) a rule, wherein a rule has four components (or levels) in the preferred embodiment, including time frame 302, situation 304, profile 306 and result 308, shown in Figure 3. The latter 3 components are constructed of Boolean expressions composed of operations. Those operations could be of several types, for example, they may be:

1) Queries from a database (SQL or Stored Procedure operation type);
2) Query about the session history - events that happened in the site and sent from the client;
3) Any Java written operation that the user desired to do something of his choice; or
4) AI operation – that normally involves an inference request from the system.

Building expressions for each of the four components of a rule may be accomplished with the use of a wizard, designed for non-technical users. The time frame level 302 defines the exact times when the business rule will be activated by the system 180 (e.g., a first quarter promotion (Jan 1 – Mar 31)). The situation level 304 defines the dynamic information relating to the current state of the system (e.g., "The user has clicked to purchase cookies."). The situation 304 triggers the evaluation of the business rule by the rule evaluator (discussed in Section 2.1 below). The profile 306 provides static information about the user (e.g., "This user is age 50".; "This user bought milk last month."). This can come from the operational or historical data stored in database 143, or it may be inferred by AI module 220. The result component is the response to the user (e.g., "This month we have a sale on milk."), which delivers personalized recommendations of products, services or content.

Of critical importance for improvement, optimization, and/or adaptability of the intelligent e-commerce system 180 by an enterprise is an ability to track the effectiveness of every business rule that affects its consumers. Accordingly, the business command center 220 includes a reporting capability that permits an enterprise to track the performance (i.e., consumer response) of their initiatives and provides a powerful analysis tool in the process. Using the knowledge gained through this automated analysis, intelligence system 200 can

recommend refinements to various rules to improve the intelligent computer system's 180 performance. Ongoing analysis, refinement and changing business needs lead naturally to the creation of new rules, in turn providing better marketing intelligence for the enterprise. As indicated in the hierarchy portion 320 of the screen shown in Figure 3, in the preferred embodiment, the intelligence system 200 reporting module tracks several critical types of information and provides corresponding reports 328, including: business rule reports, intelligence system performance reports, and site behavior reports. This performance monitoring and the corresponding reports allow enterprise users to make improvements through the creation of new business rules and the modification of existing business rules.

First, for each business rule or set of rules (e.g., Holiday initiatives) the intelligence system 200 tracks the impact on consumer behavior over time. Using key metrics, such as page views, click-throughs, and purchase patterns, intelligence system 200 enables the enterprise user to determine which rules are most effective, and to analyze rule effectiveness across consumer profiles, time periods, products and categories. Second, beyond specific rules, enterprise users also need to know what kind of impact the intelligence system 200 has on their overall on-line business performance. Therefore, intelligence system 200 tracks its own effectiveness across *all* business rules, measuring bottom-line impact on consumer behavior. Again, this information can be analyzed across profiles, time periods, products and categories. Finally, at the most basic level, enterprise users must understand how consumers are behaving on-line. Therefore, intelligence system 200 provides key information on consumer profiles (e.g., who are my consumers?), page views and click-throughs (e.g., where are they going on the site?) and consumer purchase patterns (e.g., what are they buying?).

Intelligence system 200 is also equipped with a set of default tables and charts for each type of report, which are preferably output in a standard output form, such as Excel™, by Microsoft Corporation, and customized by enterprise users as needed. All reports can be run in real-time or scheduled for on-going reporting.

Additionally, the BCC 230 includes simulation capability. That is, once a new rule has been created, the enterprise user 112 can simulate it, prior to publishing it, by accessing the learning and inference engine (discussed in section 6 below) to run a prediction for success on the considered rule. Thus, marketing initiatives can be tested internally, before publication (or launch). As a result, failed initiatives are minimized and optimal on-line results may be realized.

2. The Core 210

Again with reference to Figure 2, both the core 210 (and its rule evaluator) and the AI module 220 are key components that run on the intelligence system server 162. Core 210 provides operating system-level and other services to the intelligence system applications and modules. The intelligence system server 162 is responsible for distributing events to the appropriate services and to the intelligence system database 163, to be logged for future reporting purposes.

The core 210, which is the "heart" of the intelligence system 200, also includes a messaging facility and offers dynamic views of available computer resources. Using a highly distributed architecture, plug-ins, load balancing, sophisticated messaging, and events, core 210 continually ensures maximum utilization of system resources and retains only those objects that relate to active sessions. Merging a variety of methodologies, the core provides a robust intelligence system to an enterprise's e-commerce system, delivers high scalability and automation, and supports multiple applications. The core 210 accomplishes this primarily in two ways. First, the core uses plug-ins to automate system processes, such as data retrieval and rule evaluation. The plug-ins act independently of other system components and can be deleted, edited or modified instantly without rebooting the system. Second, the core implements a highly efficient, adaptable messaging system. The core reads files in XML and transmits in XML DOM, i.e., formats supporting the intelligence system's 200 distributed architecture.

These four main features underlie the core 210's reliability: plug-ins, load balancing, messaging, and events, and one summarized below.

1) *Plug-ins:* Plug-ins are executable components of the core created for a specific business (or enterprise) function. They can be created, invoked, updated, and removed in real-time without interrupting the system operation. Plug-ins contribute to the intelligent system's flexibility by allowing rules to be created, edited and deleted during run-time. Business rules, business objects, and certain other components are plug-ins, as examples.

2) *Load Balancing:* Load balancing is the term given to the intelligent system's ability to control the distribution of sessions (or tasks) over a group of virtual machines. Load balancing facilitates optimal deployment for any given hardware configuration, ensuring that a given machine is not overloaded with sessions, relative to other machines in the distributed architecture. More demanding services are placed on the machines with the highest capacity,

as determined by a load balancing system. This combined with he ability to run on more than one physical machines achieves scalability. Generally, load-balancing functionality is known and not discussed in detail herein.

3) *Messaging:* Query request contexts (QRC™s) agents are the intelligence system's internal messaging agents (QRC™ is a trademark of Manna, Inc. of Newton, MA). In the preferred embodiment, messages are sent to components external to the intelligence system's server 162 in XML format, and transmitted internally in XML DOM format, wherein both formats are generally known in the art. A QRC™ agent contains data, a session number, a consumer ID, a list of target plug-ins (to receive the QRC™ agent), and a list of objects that are interested in receiving a reply from the target plug-ins.

4) *Consumer Events:* Typically, a system integrator, in conjunction with an enterprise system administrator, establishes or defines events. Each event is recorded in a configuration table, which is stored in the intelligence system database 163. Any visitor (e.g., consumer) action on the e-commerce Web site (or server 152) can trigger an event (e.g., a visitor registering his name). There can be usage of internal system events as well.

2.1 Rule Evaluator

A rule is a specified business scenario that can occur on-line, including an interactive event that is triggered by a scenario. The rule evaluator, a component of the core 210, is the mechanism that evaluates information generated by an external event (e.g., a consumer purchase) according to a business rule. The rule evaluator receives the rules established by the business command center 230, and evaluates each rule using data stored (e.g., in database 163) or the intelligence system's consumer behavior models combined with other available data. The result is routed to client 154 in real-time, using the intelligence system's server's messaging facility. One business rule can serve many sessions and can be in one of two states: published or unpublished (as indicated in Figure 3), wherein the state of the rule is controlled through the business command center 230 (as discussed above). A published rule is active in the personalization server, whereas an unpublished rule isn't active, but it also appears in the rule database 163.

The business rule evaluation cycle is summarized in the flow chart 400 of Figure 4. In a typical scenario, a visitor comes to the e-commerce Web site and triggers an event that is sent to the intelligence system's server 162, in step 402. The rule evaluator, in step 404, only looks at the published rules that fit the current time frame to see if they apply to the event.

The rule evaluator then, in step 406, filters out all rules that *do not* have the event in the situation parameter, not yet determining if the rule will be evaluated as "true". The next step is the evaluation of the clauses in the situation and profile components of the remaining (i.e., not eliminated) rules. If the situation evaluates to be true, then the profile is checked, in step 408. During profile evaluation, queries are performed against operational and historical databases 143 or against the consumer behavior models from AI module 220. If the profile also evaluates to be true, in step 408, the result parameter (or level) is determined and sent back to the client 154, in step 410. The system then returns to step 412 and waits for another event, in step 412. All of these could run in parallel (assume 2 events arrive at the server at the same time).

3. Client 154

Client 154 uses highly adaptable, yet standard, protocols to mediate between all leading Web applications and the intelligence system server 162. The client can support a single Web application or adapt itself to work with Web applications distributed over a number of hosts, as well as "non-sticky" sessions.

4. Intelligence System Database 163

The intelligence system database 163 stores the configuration tables required for system setup, business objects created using a business object developer, and business rules created in the business command center 230. Log files created by intelligence system 200 for reporting purposes are also stored in the intelligence system database 163. The data is stored in standard XML format so that it is easy to send the information between system components.

5. Administration Tools 240

The intelligence system 200 includes administration tools for implementing and maintaining a stable, scalable, personalized enterprise system, in this case a Web-based e-commerce system, such as the business object developer (BOD). The BOD is an automated tool for creating business objects and their operations, which are later used as building blocks by the BCC to create rule components. Implementing a user-friendly graphic interface, the BOD implements a browser-based approach and "wizards" to facilitate ease of use. Business objects are designed for three stages of the rule evaluation process 400, namely, the evaluation of the situation, profile and result components.

Toward this end, the BOD supports operations relating to:

1) *SQL*, which identifies standard SQL queries to retrieve one or more data values

2) *Stored Procedure*, which identifies standard store procedure routines with defined input and output parameters;

3) *Session history*, which relates to events that happened in the site that are sent from the client.

4) *Java*, which can execute consumer written Java code, and any other necessary operation (like accessing a 3rd party server for more information, operating on given data and returning it processed etc.); and

5) *Artificial Intelligence Module*, which is used to access Bayesian Network Models.

Business objects are the fundamental building blocks of any business rule. Once created with the BOD, the business objects are available to the enterprise through the BCC for creating or modifying enterprise related initiatives or business rules. A business object is a group of related business functions called "methods" or "operations", and provides a mechanism to group like methods. A business object may be expressed in the format: BusinessObjectName.Method(Parameter). Methods are pieces of business functionality that are derived by accessing consumer and legacy databases, data stores, or through a series of procedures. Parameters are the values required to properly evaluate methods. However, as will be understood by those skilled in the art, not all methods require parameters.

For example, a business object could be "Consumer", offering a choice of several methods, including "Age", "Address" or "Occupation", among others. Each method may have a selection of parameters that is used to determine the value of the method, such as a consumer number. Thus, an example of a complete business object would be:

Consumer.Address(ConsumerID),

When used in a rule, the business object is written as:

Consumer.Address(ConsumerID) = "NewYork" ,

This can then be evaluated for any consumer to determine if it is true or false.

As noted previously, a business rule is a series of clauses consisting of business objects, which are evaluated sequentially to determine whether the *overall rule* is relevant for a specific consumer at that moment. The sequence of evaluation of the rule is:

Timeframe => Situation =>Profile, and if all are true, then Result.

Within the core 210, the rule evaluator determines where to go to evaluate business objects. This may require invoking click-stream data stored within the historical database 143 and/or database 163 to see if the consumer has just completed a specific action on the Web

site, or by using stored demographic or past buying data (stored in enterprise databases 143 or by using the artificial intelligence module 220) to determine the best Profile or Result to target.

6. AI Module 220

The AI module 220, is the "brain" (or AI engine) of the intelligence system 200, and makes real-time recommendations and predictions based on consumer behavior models, continuously updating them with new information. The AI module 220, more than any other component, is what transforms a standard e-commerce site into a dynamic learning and inference center. The AI module 220 creates and employs intelligent virtual agents that are capable of automatically learning, utilizing and sharing learned knowledge to serve the client 154 using inference. The intelligence system's 200 robust, scalable and distributed server architecture allows multiple virtual agents, learning independently, to form a unified intelligence entity, which acts as a single distributed "virtual brain". Learning scalability enables the intelligence system 200 to deal with the exponential growth of data and to learn large data sets rapidly. Learning takes place incrementally using intelligence system's 200 learning algorithms (described in more detail below), which provide an expedient approach to dealing with enormous amounts of data. The weights and structure of models are updated in real-time, as soon as a new piece of data arrives. These incremental, real-time updates enable learning from massive amounts of data without needing to access all the data at once or repetitively. Incremental learning also gives the intelligence system 200 the ability to learn in the dynamic Internet environment, where information constantly changes.

Consumer behavior models are implemented as Bayesian network models. Unlike other learning models (e.g., collaborative learning and neural networks), the Bayesian network models can work with incomplete data cases and levels of uncertainty. Once deterministic or probabilistic evidence has been obtained, the probability distribution for an attribute or combination of attributes can then be determined dynamically, in real-time. These consumer behavior models are reusable and can be combined to form new models, saving on both data resources and learning power. The model building process constructs attributes, defines their values and then gathers them into models according to logical relationships. These models are then inter-linked, based on their statistical coupling. In addition to profile attributes, models include products/services and categories.

Bayesian network models are built off-line and tailored by intelligence system 200 and then customized for the enterprise's e-commerce Web site. Off-line learning involves running

several intelligence system 200 programs with several input data files, resulting in a set of Bayesian models that include variables and probability connections between those variables, which may be expressed in tables.

The Bayesian models are adapted during normal system operation to reflect knowledge gained through on-line learning. As consumer-related events occur, they are logged into a database (e.g., database 143). At preset intervals, intelligence system 200 opens the database and puts the information gathered into data files. The data files are continuously updated as new events occur. On-line learning processes the files, updating the values in the models' probability tables. This action permits an e-commerce site to learn quickly and accurately about the trends and patterns of consumer behavior and activity.

In the preferred embodiment, the AI module 220 is comprised of two main parts, a machine learning system (MLS) 510 and an inference system (IS) 520, as shown in Figure 5. The MLS 510 automatically creates Bayesian network models based on consumers' past data. These models are then used by the IS 520 as on-line intelligent resources. All of the following intelligent operations can be obtained from these models: prediction, classification, maximum expected utility optimization.

Both of the MLS 510 and IS 520 are designed using parallel distributed Java object-oriented code, allowing advantage to be taken of the distributed application server architecture. With parallel distribution, the processes of machine learning (using MLS) and inference (using IS) are scalable. This implies that for making inferences the number of on-line sessions that require intelligent resources is scalable, while for machine learning the number of Bayesian Network Models is scalable with the number of computers (or servers).

6.1 Machine Learning System 510

The MLS 510 provides (1) automatic building of Bayesian Models (including attribute or feature extraction) based on an existing consumer-domain knowledge as well as on available data, (2) automatic learning of the built models (including structural/parametric estimation), and (3) automatic continuous adaptation of the Bayesian models based on new data and/or new domain knowledge, which includes the capability to incrementally change the structure based on data and add/delete attributes/links without loosing learned information from older data. In the MLS 510, there are two information "channels", one for obtaining client data and the other for obtaining client domain knowledge.

6.1.1 MLS Architecture

The MLS 510 merges two independent technologies; parallel distributed processing and advanced machine learning, in implementing an efficient, fault tolerant (via its distributed agents) and fully configurable system. Architecturally, MLS 510 includes two main parts. An off-line sub-system automatically builds Bayesian network models based on input data files, runs advanced off-line learning processes that produce the structure as well as the probability parameters for the models. An on-line sub-system is deployed as any other regular plug-in on the server and updates the Bayesian. The MLS 510 continuously operates in a background manner to adapt Bayesian models and to update them with new statistics based on the newly acquired data. Note that all references to the word 'off-line' versus the word 'on-line' mean that the off-line process is carried out using a program that need not be run on the intelligence system server 162, but rather on a separate Java machine, in the preferred embodiment.

Largely due to its distributed architecture, the MLS 510 is able to provide a potentially unlimited amount of machine learning resources (limited only by the enterprise's number of computers) that are scalable and configurable, while at the same time supporting a broad range of dynamic, real-time adaptation of these resources from a variety of sources. These resources are based on Bayesian network models (or Bayesnet models) that can learn to estimate joint probability distributions used for predictive inference and clustering (i.e., unsupervised learning), which is used to discover groupings in the data.

MLS 510 is capable of using information from a variety of sources to construct and adapt the Bayesian network models. The sources of information handled by the MLS for adapting the models include qualitative knowledge about the domain, e.g., specifying associations and dependencies concerning attributes; such knowledge can be entered directly (e.g., manually by adding/deleting attributes from models) via a Java-based graphical editor, for example. Another source of information is, of course, the data itself, which leads to a machine-initiated adaptation as part of the machine learning process. The Bayesian network models, when saved in a secondary storage, are represented in the XML standard, so that they are usable by other possible sources of adaptation, including external applications that can intelligently modify such models.

A key aspect of the MLS 510 is that the above adaptations can be carried out in an interleaving manner during the lifetime of the model. For instance, initially, a model may be automatically built by off-line learning. The on-line learning processes can then take over and

adapt the model over the next month of new data, as an example. An individual (e.g., an enterprise user 112) can then edit the model, add new relevant attributes, delete some other attributes, and insert new association links between existing and/or new attributes. The on-line learning can then continue to adapt the model based on the new structure, while keeping all important probabilistic parameter information that has been learned during the past month. For instance, the probability distribution of "AGE" given "GENDER" will be maintained even if according to the new structure a new attribute named "LOCATION" has been added.

6.1.2 Distributed Machine Learning

The MLS is comprised of multiple autonomous Java "***learning agents***" (LAs), each of which is responsible for learning a single Bayesian network model. The task of learning is accomplished by efficient parallel distributed processing starting from the lowest level of the learning process, which is filtering locally relevant data for each model, to the top level, updating the Bayesian network model at a central model repository (CMR), described in more detail in section 6.2 IS 520.

Each learning agent looks at the same continuous serial stream of incoming data (from *serial.dat* files). Each learning agent then extracts from the stream only data relevant to its model. The agent re-samples this serial data and converts it to parallel data cases. If and when some of the data is incomplete, for instance, if the value of the model attribute AGE is unknown then the learning agent has at its disposal an inference engine (i.e., the same Java object discussed with respect to IS 520, described in section 6.2 herein) which is used to complete the unknown value based on an expectation-maximization (EM) algorithm, wherein an EM-algorithm is a well-known technique for estimation of statistical parameters with partially incomplete data.

Learning agents constantly adapt the model parameters. Ultimately, a model is updated to an extent that it can be used by IS 520 for making inferences (e.g., for providing recommendations to on-line consumers). The process by which a model that just completed learning becomes available as an on-line resource is fully automatic and involves certain intelligent synchronization between MLS 510 and IS 520.

Learning agents cooperate amongst themselves and also with inference agents (IA) by responding to requests that change their set of attributes. An example of this takes place as part of the adaptation of the link structure that link "***inference agents***" (see section 6.2 IS herein). In this case, an inference agent named HLA, every so often, sends a message that is

intercepted by all learning agents, which causes multiple pairs of learning agents to exchange local information, import/export attributes, and so on. The augmented models then continue to be learned without the loss of previously learned probabilistic information. Another example is the model update. Once a learning agent writes its latest version of a model to the CMR, the CMR notifies the appropriate inference agent about the new update. The inference agent, even if it is at a state of using an older version of the model, is able to read the new version of the Bayesian model and use it for new sessions. As can be appreciated, there is a high degree of cooperation between inference and learning agents. Together they form an adaptive system influenced by changing data and domain knowledge.

6.1.3 Machine Learning Processes Overview

MLS 510 is implements of two main processes: a one-time **off-line** building and learning process and an on-going **on-line** learning process. The off-line process is an automatic process by which Bayesian network models are built using feature extraction. The off-line process need not be run on the intelligence system server 162, since it is more of a background process. Once built, hyperlinks are constructed that define the association between statistically related models, and an off-line learning process is performed to arrive at a fully learned model. Building a Bayesian model is an iterative process by which important attributes are determined, grouped and associated.

On-line learning is an incremental learning process, which reads only once through a given stream of data. The primary responsibility of the on-line learning process is to ensure that the models are updated as time progresses. While the off-line learning process need not run on server 162, the on-line learning process does run on server 162, since it has real-time interaction (e.g., updating) with models available to other real-time processes. For instance, the on-line learning process is able to automatically and quickly adapt Bayesian network models based on a real-time incoming click stream of data. The on-line learning process employs algorithms that modify the structure of the Bayesian network models, as well as track the changing parametric statistics over time in order estimate each model's probabilities.

On-line learning is a local learning process. Each model gets learned and improves itself over time. Off-line building, on the other hand, does attribute extraction from data. This requires that the whole domain of attributes (not just those that are local to a single model at a time) be considered. As such, off-line learning is a more involved process and it requires iteratively reading the input data multiple times.

While Bayesian network models may be built automatically using the off-line learning process, they could alternatively be constructed using on a partial or completely manual process. As an example, partial building could mean manually ascribing a structure (e.g., for instance a naïve Bayesian structure), while automatically learning the probabilities. In this case, off-line building is not necessary, while on-line learning is needed in order to adapt the model over time. The on-line learning process can adapt models regardless of how they were originally learned, e.g., through the automatic off-line building/learning process or manually. The on-line learning process can also continue to learn models after modification, for instance, after a new attribute is added to the model, without loosing previously learned parametric and structural information.

The off-line learning process is comprised of the following stages (or steps):

1) Defining an AI entity domain based on a hierarchy of codes, which gives rise to a single file named *totalproductinfo.dat*.
2) Defining a *serial.dat* file that, as the name suggests, is a serial stream of contiguous blocks of data (referred to as serial cases).
3) Automatically building models based on the above files resulting in sets of attributes (referred to as "intrinsic attributes"), which are logically related (according to the hierarchy of codes defined in the *totalproductinfo.dat*) and a set of non-intrinsic attributes that have been determined to be statistically correlated with the intrinsic attributes; this process includes a "feature extraction" stage (as referred to in the art).
4) Converting the serial stream of data into a parallel stream of data by employing a re-sampling technique; parallel cases based on the serial data are produced for each model.
5) Generating a directed hyperlink topology that links statistically related models; these form the basis for the inter-agent communication network.
6) Producing a Bayesian network model with an optimal structure (based on statistical properties) and parameter values using the automatic off-line learning process, which runs an advanced machine learning algorithm, based on the cases.

Unlike prior systems, the MLS 510 of the AI module 220 provides an automatic way in which complex statistical models are learned, requiring very minor human intervention, e.g., only at the start of the process through the definition of the domain. This definition stage is carried out through a sequence of simple database operations that merge into a single file all of the products (or more generally AI entities) on which the AI system can learn and produce

intelligent predictions. The off-line learning process is configured for use by non-technical enterprise users and is the basis for a relatively quick on-site AI integration phase (i.e., integration of the intelligence system 200 in to an enterprise's system).

As discussed briefly above, the on-line learning process adapts (i.e., modifies over time) the Bayesian network models built by the off-line learning process with an aim of ensuring that the models' representations of their respective domains remains accurate as these domains change over time. This means that both the probability distributions of the attributes as well as interdependencies remain up to date. The on-line learning process has the following key aspects, in the preferred embodiment:

1) Input data stream is fully compatible with the data used in the off-line learning process, so that off-line and on-line learning can be interchanged or repeated at any time.
2) A simple input data interface, which continuously polls the existence of new data files fully, automates the on-line learning input-acquisition process.
3) A re-sampling technique for converting the serial stream into a parallel stream (same as in off-line building).
4) Includes a local statistical repository (LSR), per model, which holds the statistics needed to estimate the model's parameters and structure. The LSR holds, in a concise format, all statistics collected from the initial deployment of the system and preferably has a high degree of fault tolerance by performing regular saving of files through standard Java-serialization techniques.
5) Implements incremental learning, so there is no need to store and remember data cases. This feature is well suited for a dynamic, data-rich, real-world environment, such as with e-commerce sites on the Internet. Incremental learning of parameters using a standard known algorithm and incremental learning of the structure of each model.
6) Provides an ability to learn with incomplete data, based on the EM-algorithm. Inference is accomplished as a completion mechanism for learning, followed by a formation of multiple parallel probabilistic cases with a real-valued weight. These cases then update the LSR using the same process as regular cases coming from complete data.
7) A configurable automatic "window-of-focus" permits the adaptation of models according to a decreasing level of importance for older data. This is constantly in effect, thereby making the new incoming data stream always more important for

inference purposes.

8) Accommodates manual structural changes (i.e., editing) to a Bayesian model, adding/deleting attributes, and modifying links, all of which are automatically interpreted by the CSR, which immediately starts to collect new statistics based on these changes. A model update continuity property is employed, wherein structural changes are reflected in the learned model with proper weighing, i.e., the more recent statistics (after the change) having less weight than older ones.

9) Accepts automatic external data streams for adding/deleting attribute values to/from any model.

10) Hyperlink topology adaptation (HLA) is used, wherein the inter-agent communication network adaptively changes with time, tracking new statistical correlation between agents of different contexts.

11) Fully configurable and dynamically changeable learning parameters and learning-processes are efficiently monitored.

6.1.4 Machine Learning Process Description

As mentioned briefly above, the off-line learning process is the initial stage of the machine learning process. In the preferred embodiment, there are two files through which the off-line building process gets its necessary information. The first file encapsulates the domain for the AI module (e.g., totalproductinfo.dat file). The second file is a sequence of blocks (or "serial cases"), analogous to a series of consumer orders, each block listing the full path names of AI entities which are related to each other by the fact that they are in the same serial case (e.g., a serial.dat file).

An AI entity is a unique subject, e.g., a product, which can be represented such that the learning process can associate it with other various AI entities in a statistically manner. The IS 520 is then able to do predictive inference on the AI entity. According to the coding format of the preferred embodiment, each AI entity representation includes a comma-delimited sequence of codes. An example of an AI entity having 4 codes is: 'A1','BAK','BAKBAG','100343'. The sequence may have variable number of code names for different entities. An example of a code domain having a fixed-code of width equal to 4 that describes a product hierarchy is: Acode, Bcode, Ccode, productIDcode. Here, a product is an AI entity that has a full path code, e.g., A1, BAK, BAKBAG, 12911 (which indicates a bagel product). The full path code shows that the product is from the A1 category, BAK subcategory, and BAKBAG sub-

subcategory, with a product ID code of 12911.

Basically, any string name can be used as a code name. Generally speaking, in choosing the set of possible codes, one wishes to convey a certain logical relationship about the "real world" domain. Additionally, AI entities can be described at different levels of resolution. For instance, in the last example, the bagel product comes from a defined Fresh-Food-Category having a code "A1". The bagel product also comes from a defined Bakery-Category having a code "BAK", which appears as a subcategory under the Fresh-Food category.

Once the totalproductinfo.dat file has been generated, the data files, which convey associated items, are put in the format of a list of associated full path codes that are denoted as the serial.dat file, in the preferred embodiment. Together, the two files form an AI domain and input definition according to which many real-world domains, whether they represent products, user profile values, etc., can be well-represented. In the preferred embodiment, a serial stream, rather than a parallel set of cases (as, for instance, required by other learning methodologies, such as neural networks), permits a variable length contiguous blocks of associated items. Consequently, the present approach permits a variable degree of evidence per serial case, which is typically the case in a real-world input data file.

This general representation scheme inherently has logical hierarchical relationships. As a result, a given serial case can be easily split into multi-contexts simultaneously, at very little system cost. These multi-contexts projections of the same serial case are presented to various models simultaneously, each causing an update of its respective model. For instance, the serial case: {A1.BAK.BAKBAG.1202, A2.ICE.APPL.493, B2.COF.SANK.303} simultaneously effects the A1 model, the A1.BAK model, the A2 model, the A2.ICE model, the B2 model and the B2.COF model.

The totalproductinfo.dat file is a listing of all valid AI entities and their respective code names. Figure 6A shows an example of sections of a totalproductinfo.dat file, including its format. In the example of Figure 6A, each AI entity has a code name consisting of 4 codes (the first 4 values in each line). The remaining 4 values are description names of each of the codes respectively. In general, AI entity code names may have a variable number of codes.

The serial.dat file is a single stream of data used both for off-line building and on-line learning. It is defined as any stream of consecutive AI entity code names appended by an order number. An example of a portion of the serial.dat file is shown in Figure 6B.

Regarding the example of Figure 6B, several points may be appreciated. For example, the serial.dat file is a collection of serial cases sorted by the order number of each AI entity, which is the last code in each line, wherein a serial case is a collection of AI entities. The number of codes of an AI entity may vary from one AI entity to the next.

6.1.5 Attribute Extraction and Hyperlink Formation

Once the AI entity domain has been defined and incoming data has been represented according to it, the off-line building process starts. As part of this process, an attribute extraction technique according to the present invention includes the following key steps or aspects:

1) Collecting and forming sets of logically related attributes, i.e., according to their 'nearness' in the logical hierarchy, as defined in the totalproductinfo.dat. For instance, an attribute for each subcategory under A1 is added to a common set denoted as an intrinsic set of the model A1.
2) Iteratively scanning the serial.dat file, wherein each Bayesian model computes a representative set of attributes ("rep-sets"). Pairs of models whose rep-sets are highly mutually informative get linked via an importation of rep-set attributes from and to each other. As a result, a topological Bayesian network (different from the logical tree-like hierarchy) is built that links the models.
3) At the end of the process, there exists a collection of Bayesian models (not yet "learned") that are dependent both in a data-based sense (through the statistically formed topological links) and through logical relations.

This a fully automatic process of incorporating both prior domain knowledge (based on logical relationship) and empirical data information into a single unified collection of attribute sets which are later to become Bayesian models.

In the preferred embodiment, each hyperlink has a value that is proportional to the mutual information between the two respective representative sets. This coupling of models serves as a basis for a measure of information over the communication channel. An inference agent, when having the choice to receive information from multiple agents, decides from whom to receive, based on this coupling and also based on the evidence (or data) available in each of these agents.

Figures 7A and 7B show examples of logical and topological relations linking models, respectively. For example, in Figure 7A a topology of inference agents 700 is shown. In this

example, for instance, model A2 702 is connected to model A1.DAR 704, which suggests that there is a strong association (the strength of which is available to the two models) between patterns of product orders on the domain of model A1.DAR. In this example, model A1.DAR 704 happens to represent dairy products ("DAR") and model A2 702 represents the domain of frozen foods. These models are also related according to the logical hierarchy 750 shown Figure 7B. In this case, model A1.DAR 704 is represented as a folder "Dar" 708 that is subordinate to (or included in) folder "A1" 706, wherein folder A1 706 has a close relationship with A2 710, i.e., A1 706 is at the same level and proximate to A2 710 in the logical hierarchy.

6.1.6 Off-line Building Algorithms

The data serial.dat and totalproductinfo.dat files may be prepared using a stored-procedure database routine or some other external mechanism. Once the totalproductinfo.dat file exists, the corresponding model definition based thereon has a variety of characteristics. For example, given a corresponding Bayesian Network model by the name of "Top" (shown in Figure 7B), all of the first code names (i.e., the first code in each line of the totalproducitnfo.dat of Figure 6A) are defined as attributes of the model Top. Additionally, all 3 consecutive code names are recursively obtained. As an example, let the first code be the model name, the second code be an attribute name and the third code be an attribute value for the respective attribute.

For instance, based on Figure 6A, a Bayesian model by the name of A1 is created with the following attributes: A1.BAK and A2.VGP. The A1.BAK has the following values: 'BAKBAG', 'BAKBRD'. The attribute A2.VGP has the following values: 'FRZVGG', 'FRZVBE'. In a similar manner, a model by the name of A1.BAK is created with the following attributes: A1.BAK.BAKBAG and A1.BAK.BAKBRD taking the values: '100335' to '100357' and '100366' to '100376' respectively.

Before proceeding to the building algorithm, it should be understood that an "intrinsic attribute" of a Bayesian model is an attribute whose name consists of the model's name as a prefix followed by an additional single code name. For instance, A1.BAK.BAKBAG is an intrinsic attribute of model A1.BAK. Given the totalproductinfo.dat and serial.dat files, the model building proceeds according to the following steps:

1) Create a directory structure where each directory corresponds to a single Bayesian

model and the logical names of models are a function of their placement in the directory structure (see Figure 7B).

2) Let $d > 1$ be a given parameter (where d represents the number of hyperlinks).

3) For each model in a set of models:

   a) Collect all intrinsic attributes.

   b) Prepare a representative set (see rep-set discussion below).

4) For a specific model X from the set:

   a) For each model M ≠ X (where model M represents each model in the set that is not model X):

      i) Compute the score S(X, M) and store in list L1.

   b) Let there be d hyperlinks between model X and d other models M whose score S(X, M) is in the top d scores in the list L1 where a hyperlink is created using algorithm exchangeAttributes (see the exchange attributes discussion below).

   c) Store the d hyperlinks in list L2.

   d) Prepare a representative set for model X and for these d models.

5) Scan list L2 and remove duplicate hyperlinks ending up with a single hyperlink between any pair of models.

6) Print out into a *hyperlinks.xml* file, a list of all hyperlinks.

7) For each model X:

   a) Create and print into a file *cases.dat* in the respective directory of model X the cases for model X (see case created for the model discussion below).

   b) Print the attribute information for the model X into a file *varinfo.dat* in the respective directory of model X.

We now review the necessary algorithms used above. First, the algorithm used to prepare representative sets:

1) Given a model X and a given parameter t.

2) For each attribute A in model X:

   a) Compute the score S(A) as the total mutual information between A and any other attribute.

   b) List the t highest scored-attributes and place in list L.

3) Output list L as the representative set for model X.

Next, to exchange attributes:

1) Given two models X and M, let their representative lists be Lx and Lm respectively.
2) Let model X import the attributes listed in Lm and label them as non-intrinsic attributes for X. Let model M import the attributes in list Lx and label them as non-intrinsic attributes for M.
3) After importation, remove any possible duplicate attributes in model M and in model X.

To create cases for models:

1) Given a *serial.dat* file and given model X with its intrinsic and possibly non-intrinsic attributes.
2) Initialize a list LC of cases to empty.
3) Scan once the *serial.dat* file and <u>for each serial case (SC)</u> which contains at least one AI entity whose code name has as a prefix the name of model X do the following:
   a) Let Lx be a list of all AI entities in SC whose name has as a prefix any one of the attribute names of attributes in X. Let *c* be the cardinality of Lx.
   b) Obtain the order-dependent probability distribution for each such attribute, for example, given SC = {{A1,BAK,BAKBAG,1012,5},{A1,BAK,BAKBRD,1112,5},{A2,VGP,VGPFZZ,232,5}} and given model X name is A1 with intrinsic attributes = A1.BAK, A1.FRT and non-intrinsic attributes A2.VGP, A2.ICE then the order dependent distribution for the attributes is as follows:
      i) Attribute A1.BAK has {[NULL,0], [BAKBAG, 0.5], [BAKBRD, 0.5]},
      ii) Attribute A1.FRT has {[NULL, 1], [FRTA, 0], [FRTB, 0]},
      iii) Attribute A2.VGP has {[NULL, 0], [VGPFZZ, 1], [VGPA, 0]},
      iv) Attribute A2.ICE has {[NULL, 1], [ICEC, 0], [ICEYOG, 0]}.
   c) Create *c* cases as follows: for each attribute, in turn, choose its corresponding order-dependent distribution, draw a value *c* times from its domain using its order-dependent distribution. Together, these form *c* parallel cases, each case having a value for each attribute of model X.
   d) Append the *c* cases to the list LC.
4) Output LC to the file cases.dat for model X

An example of a hyperlinks.xml file is shown in Figure 8. As can be seen, a single

hyperlink defines a connection between MODEL1 and MODEL2, based on a shared set of attributes whose names is listed under SHVARS. The link has a coupling which is defined as the mutual information I(S1; S2), wherein S1 is a subset of shared variables originating from MODEL1 and S2 is a subset originating from MODEL2. For related hyperlinks description also refer to section 6.2 IS 220.

6.1.7 DRV Model Building

Derived models, referred to as DRV models (also see section 6.2 IS 520), are used for predicting the probability of a NULL and then distributing this prediction as derived evidence to the associated models. As far as building models is concerned, hyperlinks are not used for connecting the DRV models. The cases.dat files are formed based on a conversion process from a serial *serialsession.dat* file to parallel cases that reflect the time of entry into a subcategory, for each subcategory of a given category.

An example of a serialsession.dat file is shown in Figure 9, wherein its format is similar to the serial.dat file. The last code name in each line represents a session number. The serialsession.dat file lists AI entity code names in an incremental fashion. For instance, with respect to session No. 0, the file indicates that the consumer visited category TOP, then chose a link taking him to subcategory A1.BAK.BAKBAG (this counts as the first entry to category A1). The consumer then exited and re-entered category A1, chose subcategory DEL under A1 (counting as the second entry to category A1), then subcategory BRD (this counts as a third re-entry to category A1), and finally subcategory A1.DEL.DELMTS (which counts as a fourth re-entry into category A1). Based on this, the building process builds a cases.dat file for each DRV model, which contains cases that indicate for each subcategory under each category whether they have been entered and also at what entry time (where entry time of a subcategory is defined as the time that the immediate parent category was entered just prior to entering the subcategory). An example of such a case is displayed in Table 1, below.

DRV.A1.PKG.NULL	NULL
DRV.A1.PKG.TIME	5
DRV.A1.VEG.NULL	OTHER
DRV.A1.VEG.TIME	2

6.1.8 Off-line Learning

Off-line learning picks up where off-line building left off. This is a standalone routine that, from the parallel data cases produced by off-line building, learns a structure and estimates

the probability parameters of a Bayesian model. There are various modes according to which different implemented algorithms are used. The algorithms are primarily extensions of well known, but yet advanced, statistical principles, such as statistical model selection based on m_____ d_____ l_____ (MDL) criterion. For example, such algorithms and models include:

1) Parametric learning: standard Maximum Likelihood Estimation.
2) Structural learning: several pruning-based, as well as constructive, algorithms yield a final directed link structure. Representative Set based, MDL-based pruning.
3) Constrained learning of structure where a particular focus on a specific set of attributes is made as being the attributes of interest for prediction.

For illustrative purposes, the details of only one of the structural learning algorithms used in the off-line process is presented herein. Other algorithms are variations of that presented herein, as would be appreciated by those skilled in the art. The structured learning process constructively forms an initial structure. It picks out a representative set for a model, then links the rest of the attributes based on their coupling to this set. Special care is taken when a set of attributes is said to be of particular interest for prediction.

1) Given a cases.dat file for learning model X and a parameter c and a set O of output attributes that are of special interest for prediction (this set may be empty).
2) Determine the non-intrinsic attributes and intrinsic attributes of model X based on the names of the attributes.
3) Based on the cases.dat file, compute a table TJ of all joint probabilities P(Ai, Aj) of all possible pairs of attributes Ai and Aj in model X. Compute a table TS of single probability distributions of every attribute Ai in model X.
4) Based on TJ and TS compute a list ML of all pair-wise mutual information between all pairs of attributes Ai and Aj of model X using the formula:

   $I(Ai, Aj) =$

   $\sum_{k,m} P(Ai = k, Aj = m) \log P(Ai = k, Aj = m) / P(Ai = k)\, P(Aj = m).$
5) For each attribute A in model X, compute the score s(A) as being the total mutual information between A and other attributes.
6) In a set R put the set of c attributes having the highest c scores. If the set O is not empty, add to set R its elements.
7) For each attribute in the set R, create an outgoing link from it to every other attribute in

the model X while ensuring that no cycles appear.

8) Compute the initial MDL-score(X) of model X (see the discussion on MDL Scores Of A Bayesian Network Model below).

9) Repeat the following iterative process until a local minimum of MDL-score(X) is reached:

   a) For each link LK in the current version of X:

      i) Determine the change in MDL-score(X) for the following possible candidate structures: the same structure but without the link LK (provided the link is not one which connects an element in the set *O*); and the same structure with the link LK in a reversed direction, where the change is with respect to the current structure which includes the link LK. Note that the possible candidates must be valid, i.e., contain no cycles, otherwise continue and consider the next link in LK.

   b) Let the new structure of model X be the one which results in the maximum decrease in MDL-score(X) amongst all possible candidate structures.

10) For the obtained optimal structure of model X, for each attribute compute the conditional probability tables based upon the *cases.dat*.

11) Output a *bayesnet.xml* containing the graphical structure and the conditional probability tables.

Computing the MDL-score(X) for a Bayesian Network model X is a well-known computation. However, for completeness, the MDL Score of a Bayesian Network model we determined as follows:

1) Let *n* be the number of attributes in model X.

2) Let #Pa_i be the number of parents of the i-th attribute.

3) Let N be the number of cases in *cases.dat*.

4) Let |Ai| be the number of values of the *i*-th attribute.

5) Let *Comb(n,k)* = n! / (k! (n-k)!).

6) Let H(Ai | Pa_i) be the conditional entropy of Ai given the set of parents Pa_i (for the definition of conditional entropy see T. Cover & T. Thomas, *An Introduction To Information Theory*, (John Wiley & Sons 1991).

7) Then:

$$MDL\text{-}score(X) = \sum i \; (\log(n) + \log Comb(n, \#Pa_i)\,) +$$

$$\frac{1}{2} \log(N) \sum_i (|Pa_i| \ (|Ai| - 1)) +$$
$$N \sum_i H(Ai \mid Pa_i).$$

6.1.9 DRV Model Learning

A DRV model has two types of attributes, NULL and TIME with an outgoing link from NULL to TIME. Thus it has a fixed structure, as shown in Figure 10. The NULL attribute takes two values {NULL, OTHER} while the TIME attribute takes values {0,1,...,M}, for some constant M > 0. The NULL attribute is presented with a value NULL if in a given session the consumer did not enter a subcategory. For instance, DRV.A1.PKG.NULL = NULL indicates that the consumer did not enter the A1.PKG subcategory under the category A1. A value of "OTHER" indicates that the consumer did enter the subcategory and the associated time of entry is presented as a value to the corresponding TIME attribute. The time of entry is defined as the number of times the consumer entered the category (in this example it is A1) prior to entering the subcategory. Therefore, for instance, DRV.A1.PKG.TIME = 2 indicates that the consumer had already entered category A1 once.

As depicted in Figure 10, the reason for having NULL feeding an arrow to TIME is for viewing the NULL as the class variable to be predicted. The model presents two possible class conditional distributions over the domain of TIME. The first is P(TIME | NULL = *NULL*) and the second is P(TIME| NULL = *OTHER*). Learning processes take care of estimating these two distributions. The inference agent of this DRV model predicts the distribution of NULL, namely, P(NULL) = [P(NULL = *NULL*), P(NULL = *OTHER*)] based on observed evidence of TIME only.

Based on the cases.dat file obtained from the building process, off-line learning simply learns the probability estimates for each of the conditional probability tables of the DRV model as it learns any other "regular" model. An example of a derived case for learning the model DRV.A1 in Figure 10 is shown in Table 1 above, which corresponds to a serialsession.dat case that indicates that a consumer entered subcategory VEG under category A1 after having visited A1 twice during the session, while not having visited subcategory PKG under category A1 (we have used M = 5 as the max time limit in this example, so having 5 means that the user session terminated without having entered the PKG subcategory).

6.1.10 MLS On-line Learning

As discussed in the sections above, the off-line learning process generates a file in an

XML format that represents a fully learned Bayesian Network Model. The task of the on-line learning process is to modify this Bayesian Network Model over time with newly acquired data. In accordance with the present invention, MLS 510 includes the following aspects:

1) Local and incremental learning of parameters and structure, including advanced structural learning which involves state-space search in the space of model structures.
2) Various flexible forms of updates, wherein Bayesian models can be changed structurally, i.e., links and attributes alike, with minor loss of parametric information that has been previously computed based on past data.
3) Learning with incomplete data, wherein multiple probabilistic parallel data cases are formed for models that have at least a single missing value. These (weighted) cases are used to modify the LSR similar to regular deterministic cases.
4) An incremental inference-assisted learning of the hyperlink network that links all models takes place. Preferably, a single model, named HLA, captures mutual interdependencies between the rest of models and automatically instructs learning agents to cooperate by exchanging attributes.

6.1.11 On-line Learning Modules (or Components)

An ORB (ORB) (such as Voyager™ by Objectspace, Inc. of Dallas, TX) provides, among other things, remote-Java-method invocation between Java objects on separate Java virtual machines (VMs or JVMs) to facilitate the MLS on-line learning process. During this process a variety of components are used, and described in table 1100 of Figure 11. With respect to table 1100:

1) A component is a modular sub-part of the on-line MLS learning system or of the AI system as whole.
2) QRC™ agent is an XML-based object-oriented communication messaging protocol, a table 1400 describing various QRC™ agents are shown in Figure 14. Different AI components communicate using this protocol.
3) Client stands for any external event-generating server, for instance, an Internet server that passes "product-order" events made by on-line users (e.g., client 154 and consumers 110 of Figure 2).

The interaction between these components can be appreciated with respect to the on-line MLS block diagrams 1200 and 1250 of Figures 12A and 12B, respectively, and with respect to components communication table 1300 of Figures 13A and 13B. Figure 12B is a partially

decomposed portion of Figure 12A and the interactions between the MLS on-line learning system components are described in table 1300. For instance, table 1300 indicates, in row 1310, that the learning manager (LM) 1202 sends a QRC™ agent to the CMR 1208. Interaction Type of each component is with respect to another component (not a component's partner). Additionally, with respect to table 1300:

- *Admin* stands for an external plug-in used to administrate/configure the on-line MLS
- *Accept/Send* means a component does this action.
- *Invoke Method* means component invokes method locally (not through ORB)
- *Remote Method* means a component invokes method through ORB
- *Method Invoked* means a partner invokes method on the component

Concerning the Number column of table 1300 of Figures 13A and 13B:

- *1* means a single unique component
- *K* means there can be a flexible number, K>1
- *M* means the number of models

6.1.12 MLS On-line Learning Algorithms

The input provider of each LU (e.g., LU 1206 and LU 1206') accesses the same serial.dat file stream (e.g., input data files 1212 of Figure 12B). After completion of processing the current serial.dat file, each LU polls to check if a new serial.dat file has been placed. The format of the serial.dat is identical to that used for off-line building (see Figure 6A). The LU's input provider forms parallel cases from each serial case using the same mechanism as described in section 6.1.6 above for creating cases for models.

Each LU 1206 contains a tree-like data structure that holds statistics necessary and sufficient to estimate the probabilities of any of the conditional probability distributions of the relevant model. There are many possible data structures that can be used. In the preferred embodiment, a structure known by the name of *AD-tree* 1214 (see A. Moore & M. S. Lee, *Cached Sufficient Statistics For Efficient Machine Learning With Large Data sets*, 8 Journal of Artificial Intelligence Research 67, 91 (1998)) which has several memory as well as computationally efficient properties. Using this structure, frequency counts for all variable combinations are taken with respect to the data and are used as the maximum likelihood estimates of the probabilities in the conditional probability tables.

Based on the available statistics, each learning agent uses a standard state-space search strategy to find an optimal (in the local sense) structure for the model which has an optimal

MDL score (see, for instance, D. Heckerman, *A Tutorial On Learning With Bayesian Networks*, Microsoft™ Technical Report MSR-TR-95-06 (1996)). The search is similar to that described with respect to Figure 9, except instead of basing the estimate for the conditional probability distributions on the cases.dat file, the learning agent uses the statistics in its LSR.

For a given current structure, the probability estimates of each of the conditional probability tables in a model are computed based on the contents of the same LSR.

A learning agent writes its updated model to the CMR via the communication between its LU and the CMR (see Figures 12A, 12B, 13A and 13B). The frequency of updates is configurable via a parameter in the MLS configuration setup on the server 162 of Figures 1 and 2. After each model update, a learning agent computes its model's representative set using the same mechanism as described with respect to the algorithm for preparing rep-sets in section 6.1.6 above. All probabilistic knowledge for computing the scores of each attribute is available to the learning agent 1216 based on the content of its LSR.

A Bayesian model that has been updated and written is saved in an XML format which can be read off using the JavaBayes editor (JavaBayes is a public domain Java object editor provided by the Free Software Foundation, Inc., Cambridge, MA and useful for creating and maintaining object oriented Bayesian Models). Using the JavaBayes editor, manual changes to Bayesian models may be accomplished, such as adding new attributes or deleting existing ones, or adding/deleting values of attributes. Once these attribute-related changes are saved from the editor, the MLS on-line learning process is capable of reading (via the CMR) the XML file, building the Bayesnet model (or Bayesian Network Model) based on its newly changed structure.

Consequently, the LU 1206 of the particular model ensures that any new serial cases will get translated into parallel cases (as described above in this section) that conform to the new structure. Moreover, the LSR does not loose any statistical information that was learned for the prior structure of the model and which is still needed by the new structure. The only statistics deleted are that are not necessary, based on the new structure. The algorithm that determines which statistics to hold and which to delete from the LSR is described above in this section with respect to the structured learning process.

6.1.13 Learning with Incomplete Data

The MLS on-line learning system 1250 can learn from partially-complete cases. The corresponding algorithm, which is an implementation of the well-known EM algorithm, is as

follows:

1) An external configuration file identifies some models as "models with incomplete cases". For an attribute of such a model, a NULL value is never inserted by the learning unit's input provider. This is true both when this attribute appears as an intrinsic attribute in the model itself or as a non-intrinsic attribute in some other model.

2) Given a serial case, the same mechanism that is described for creating cases for models in section 6.1.6 above is used, except for the following: for those attributes which do not have at least one AI entity in the serial case which has them as a prefix, instead of inserting a NULL value, insert the following special value "NULL?". This forms the incomplete case *c*.

3) Once the cases are read by the learning agent, given a case which has a special value "NULL?" the learning agent employs an object called the Learning Assistant 1218 which utilizes the latest version of the learned model in conjunction with an inference engine described in section 6.2 below.

4) Given a case, the learning assistant 1218 first sets respective values for all attributes having a non-NULL? value. The learning assistant 1218 then computes a joint probability distribution *P* for the subset of attributes that have a NULL? in the current case.

5) Suppose there are *k* values in the domain for *P*. The learning agent 1216 transforms the original case *c* into *k* complete cases, each having a different combination *comb* of values over the attributes that originally had a NULL? value.

6) With each such complete case an associated weight *w* is defined which equals the probability of the associated combination *comb* according to the joint probability function *P*.

7) Once all *k* cases and their weights are ready, the learning agent 1216 increments its local statistics repository (LSR) due to each case, but with an increment not being 1, but rather being the weight (which is a number between 0 and 1).

6.1.14 Hyperlink Adaptation

Next a Hyperlink Adaptation (HLA) algorithm enables the adaptation of the hyperlinks used to connect the inference agents described in section 6.1, and is as follows:

1) Let there be an existing collection of Bayesian models, M1, M2, ..., M*k* fully learned and deployed as part of the on-line MLS.

2) Let a model named HLA be defined as follows: define attribute HLA.M*i* in the HLA model to correspond to a model M*i* in the above collection. The attribute HLA.M*i* takes two possible values {T, F} (i.e., true or false).

3) A structure for the model HLA is manually defined so that it is sufficient to represent every possible joint probability distribution over any possible pair of models M*i* and M*j*.

4) The serial.dat file is augmented using the following routine,
for each serial case:
   a) determine the list of models everyone of which has at least one AI entity code that has the model as a prefix. Let this list be denoted as L.
   b) For each model X in L do:
      i) Form the AI entity code HLA, X, T.
      ii) Append the AI entity code to the end of the current serial case.

5) Finish processing all the serial cases in this manner, which forms the augmented serial.dat file.

6) The MLS learns from the serial.dat as described previously. In particular, model HLA learns and the corresponding inference agent (IA) for model HLA can compute any pair-wise joint probability distribution P(HLA.A*i*, HLA.A*j*).

7) A module called HLAM uses the HLA IA for computing the mutual information I(HLA.A*i*, HLA.A*j*) between all pairs of attributes.

8) Between every pair of models M*i*, M*j* in the collection, a coupling *c*(M*i*, M*j*) is defined as I(HLA.A*i*, HLA.A*j*).

9) After every periodic update of its model the learning agent of HLA informs the HLAM to start adapting the hyperlinks.

10) The HLAM creates a new list of hyperlinks L as follows: for each model X, connect the *d* highest coupled models to it.

11) The HLAM delete duplicate hyperlinks from L.

12) In a model building process similar to that described in section 6.1.6 above, the HLAM considers once in turn each of the learning units 1206 and repeats for each one the following:

a) The HLAM sends the list L via QRC™ agent message to the learning manager 1202 containing the currently considered learning unit 1206.

b) Once having received L, a learning unit 1206 instructs its learning agent 1216 to:

i) Delete non-intrinsic attributes that came from currently linked neighboring models.

ii) Import the new attributes from the representative set of the new model to which it is to be connected based on the list L.

c) The LA 1216 imports these attributes from each of the *d* to-be-linked neighbors in a similar manner to the model building algorithm described in section 6.1.6. It then re-computes the representative set using the same preparing rep-sets algorithm also described in section 6.1.6.

6.2 Inference System (IS) 520

The inference system (IS) 520 (see Figure 5) parallelizes the task of inductive inference. Inductive inference is the process of making intelligent decisions/predictions given observations (also referred to as evidences) which have not necessarily been part of an original data set used to learn a Bayesian model. As such, there are two information channels that are associated with the IS 520. The first is the evidence channel, which is an event-based, asynchronous channel by which observations are provided to IS 520. The observations are provided, in parallel, for any subset of attributes of any of the Bayesian models. The second channel provides the means of obtaining the prediction/decision output from any of the Bayesian models. This second channel is based on an interface that defines several types of outputs, which are called *recommendations*. These are described in further detail below.

6.2.1 IS Architecture

The inference system 520 merges two independent technologies, parallel distributed processing and artificial intelligence, in implementing an efficient, fault-tolerant and accurate prediction/decision inference process. As mentioned earlier, the IS has the ability to provide a potentially unlimited amount of AI resources (i.e., limited only by the enterprise's number of computers), which are scalable and configurable, while simultaneously keeping a dynamic and efficient representation of the ever-changing state of the domain (e.g., simultaneous on-line user actions over multiple sessions) such that accurate predictions (i.e., recommendations to the users) can be promptly and reliably made.

The IS 520 is a main part of a single unified artificial intelligent "brain" from which AI inference is performed. IS 520 comprises multiple autonomous inference agents (IAs) that utilize the Bayesian models created by the MSL 510 to make the recommendations, wherein inference is performed using an intricate collection of localized algorithms (local at each agent). Inference agents cooperate, that is, they share "beliefs" on the current state of the "world". The inference agents listen to each other, taking various measures of confidence and information values into account before deciding on their recommendations.

6.2.2 Inference with Incomplete Evidence

IS 520 uses Bayesian model networks to accomplish inference with partial evidence. The use of Bayesian model networks to represent dynamic, complex (e.g., e-commerce) applications suits well with the dynamic nature in which information appears and changes in a typical on-line product-recommendation application, for example. The IS is configured such that it can give recommendations from the very start of a consumer 's session, even if, for instance, personal information about the user is missing and all that is known about the consumer is learned from current actions made on the site.

One of the primary advantages of having a distributed AI system 220 is in its ability to learn concepts on a multi-resolution level. This provides a multi-degree of inference about concepts. For instance, consider inferring about the concept of "Milk". This concept is, on a broad level, related to product families such as "Frozen Foods", "Packaged Foods", “Drinks” and so on. In the preferred embodiment, the representation of milk takes place over several Bayesian models, each model having a context in a different inference level. As discussed in section 6.1, these models are automatically built using the MLS 510. These simultaneous multiple context levels for the same concept, for all concepts in the domain, permit a rich variety of inference capabilities that suit different states of evidences and thereby increase the chances that even with partial evidence highly accurate predictions are possible.

Most e-commerce applications involve dynamically changing information. For instance, consider an on-line session of a consumer on an e-commerce Web-site. The consumer goes from one link to another, continuously changing the location on the site, adding and deleting products to a shopping cart. In general, not only the static information about the consumer, such as his gender, age and demographic information, is important for predicting his product liking, but also dynamically changing information.

However, in the preferred embodiment, *dynamic inference state representation* (DISR)

allows all types of evidence, whether they originate from static database tables or from rapidly changing user click data, are taken into account prior to product recommendation or, more generally, prior to any AI prediction. The key aspect of this is referred to as *Inference State Representation* (ISR), wherein, per each user session, at any given time instant, a user's state is the aggregation of evidence and internal attribute predictions across *all* Bayesian models up to that point of time.

Listed below are aspects of the IS 520 which ensure the dynamic updating of state (or *Passive Dynamic State Update*):

1) Continuous evidence collection and distribution across the wide Bayesian model network in the background in a passive manner, i.e., without needing the user (e.g., consumer) input, which alleviates time-consuming form-filling and completion of multiple choice questions to build a requisite amount of user data.
2) Simple and direct event-to-evidence conversion based on the hierarchical code format (e.g. a product order event for a product ID 12212 is converted to a full path name A1.BAK.BAKBAG.12212).
3) The above conversion permits a continuous distributed propagation of evidence so that a single event quickly influences the current state to reflect a change made by the user across the whole model hierarchy.
4) Accumulation of evidence using deterministic to probabilistic conversion of evidence.
5) In the same model, attributes can be of different types, accumulating new evidence or overriding new evidence. This gives a high degree of control over how new evidence gets updated.
6) Internal attribute predictions used as part of the communicated messages amongst the agents are cached, which greatly reduces redundant computations.

6.2.3 Derived Evidence

Often, deterministic evidence alone is not the only information available about the consumer's session. For instance in an e-commerce Web-site for selling products, the Bayesian models learn that the buying of certain product categories A and B go together while when buying from category C most likely one does not also buy from category D. Therefore, the notion of *not* buying from some category could be as important as actual buying from a category, for obtaining a high accuracy of prediction. However, the difficulty is in determining (if not deterministically at least to a certain degree of confidence) that an on-line

user is one which *does not* buy from category A. The possibility of asking the consumer is out of the question, since, as a general rule, it is considered undesirable to bombard the consumer with questions. Generally, any user behavior, for instance expressing a variable degree of interest in a product, which can be formulated either based on direct consumer-related events or indirect effects (e.g., time of entry to a category, pattern of category entries), can provide an important extension of the Bayesian models.

For the above reasons, the present invention includes a capability for *deriving* evidence that is based on predictions made by a separate set of models. These models are denoted as *Derive Models* (DMs or DRV models), and are not used for predictions geared for recommendation to the outside, but rather, only for AI-internal purposes. The DMs can be defined to monitor various events, such as category-change events, product-order events, events that reflect a user's level of interest (for instance, the amount of time a user spends at a particular category), and are used to predict various internal attributes of the recommending Bayesian models (i.e., the models that are used for external recommendation to the on-line user or to the business marketers). Even so, the DMs are in many respects similar to the recommending models; they are off-line as well as on-line learned, thus can be updated over time.

Accordingly, unlike other systems, the IS 520 generates evidence which does *not* directly come from the external world, e.g., does not reflect the purchase of a product or the change to another category on the site, but yet is usable by the recommending models. This evidence is probabilistic with varying degrees of confidence and is available for the same recommending models that usually get their evidence deterministically directly based on external events. Thus, a recommending Bayesian model, which has been learned and is being used for inference based on direct evidence, can also utilize this other source of information, the DMs, which derive evidence based on more subtle combinations of events or the absence thereof.

6.2.4 Inference Agents (IAs)

The purpose of inference agents is to deliver intelligent predictions and decisions concerning any subset of a fixed set of variables in a fixed and defined Bayesian model, while automatically and continuously collecting as much evidence as possible from different information sources which include, but are not limited to, dynamically changing information from inference agents, static information residing in a database, evidence due to real actions

taken by the current on-line consumers via server 152.

An inference agent is, in the preferred form, a complex self-autonomous Java code utilizing an inference engine (IE) for computing probabilistic inferences that are used for delivering predictions on variables in the Bayesian model network. The IA has various interfaces which connect it to the various sources of information, based on which the IA receives (and also sends) probabilistic information as evidence. Also, the IA employs several algorithms for information communication, all of which aim at improving the value of information that it receives. These algorithms include probability-distribution emphasis mode (which may be viewed as a filtering action) and absorption from more informative peers.

6.2.4.1 Classification and Prediction Modes

The most common prediction mode used with Bayesian model networks simply provides the probability distribution of the targeted variable. For doing classification (as is known in the pattern recognition field) the well-known Maximum Aposteriori Probability (MAP) decision process can then be applied. The inference agents of the present invention can predict and classify in this manner. Moreover, the IS 520 also implements a score-based classification mode that provides, for such Bayesian model networks, a higher rate of accuracy than the classical MAP decision process. In the preferred embodiment, the score-based classification mode is referred to as the DELTA classification mode.

In the preferred embodiment, there are 3 types of scoring functions used as part of the DELTA mode. Each is a different formula that assigns a real-valued number to every of the queried-attribute's values. The decision is then to choose the value with the largest score. In the DELTA mode an inference agent first computes the no-evidence probability distribution for each variable in the Bayesian model, which is denoted as P_no-evid. During a session, once evidence begins to be observed or propagated, the probability distribution for an attribute X is denoted as P_evid(X). Let x denote any value of the attribute X, and the 3 different scoring formulas are as follows:

1) Relative-score:

$score(x)$ = [P_evid(x) - P_no-evid(x)] / P_no-evid(x),

if P_no-evid(x) > beta = 0,

where beta is a fixed constant

2) Confidence-score:

$score(x)$ = Norm(0, sigma, x), where

Norm(m, sigma, x) = $(1/\sqrt{(2\pi s)})\int^{x}_{-\infty} \exp(-(t - m)^2/2s)\, dt$,

and sigma is a fixed constant

3) Omega-score:

$score(x)$ = P_evid(x) log(1/P_no-evid(x))

6.2.4.2 Involuntary Dispersal of Evidence

Each inference agent constantly distributes probabilistic evidence to a fixed set of peers, as soon as it receives deterministic evidence that comes from the inference manager (IM) or evidence from a peer agent. This is conducted passively, independent of recommendation requests that may arrive at any time to any of the agents. The purpose of this process is to make the IS agents (or inference agents) as aware as possible of the dynamically changing evidence simultaneously for all on-line sessions on the server 162, so that when a prediction request for any particular agent arrives, the chances are that the corresponding agent has evidence, even if the user did not necessarily act in a direct context of the agent's model, e.g., agent Dairy will still have evidence even if a user did not buy a Dairy product.

As an implementation, a communication network links all inference agents to their peers and to the source of evidence. As an example, the inference manager serves as an interface to server 162, which is a source of evidence. This is not a communication network as used in the field of data communication, but is rather a logical software-based graphical representation of the relationships (i.e., links) between agents. Over this network, probabilistic information in the form of Java objects representing discrete probability distributions are passed. One IA passes its 'belief' to another IA. For instance, evidence due to a placement of a Dairy product in the session "basket" can become evidence that the user most likely is not interested in Meat products, but is interested in Fresh Cheese products. This evidence is dispersed by the Dairy Agent passing its belief to the Meat Agent and to the Fresh-Food Agent. The present invention includes various features related to making information pass efficiently amongst the agent network, such as cycle avoidance to avoid redundant message passing and caching of previous probabilistic evidence distributions.

In addition to involuntary evidence dispersal, there is a need for evidence transfer that is dependent on the event of a prediction request. In such situations, each inference agent determines when and from whom to request evidence. This is dynamically determined based

on several considerations, such as the amount and value of evidence that the agent peer has obtained. The value relates to how many stages (i.e., agents) the evidence propagated through before it influenced the peer agent and also to the statistical coupling between the two inference agents. Also, it is possible for inference agents to operate in a synchronous mode that guarantees the arrival of evidence before an agent answers a prediction request. In this manner, it is guaranteed that the prediction took place when the maximum amount of evidence was at hand.

6.2.4.3 Inference Engine (Efficient Computations)

Each inference agent in the intelligence system has a pool of inference engines (IEs) on which it runs all inference computations. Having multiple identical service providers, such as these engines, permits concurrent (or parallel) processing, which can reduce significantly the performance times. In the preferred embodiment, each inference engine is a Java implementation of a well-known sum-product algorithm, specifically known as the *bucket elimination* algorithm. This algorithm is extended by several ways in the present invention, as described below.

1) It uses probabilistic as well as deterministic evidence.
2) It simplifies the complexity of computations using a dynamic marginalization technique, referred to herein as *dynamic attribute killing*. This technique allows for a tradeoff between prediction accuracy and time of computation, which is extremely important since it permits dynamically tuning the prediction accuracy based on the allowed time limit for returning an inference prediction. As a result, regardless of how busy the server 162 is (i.e., the number of on-line users), the AI recommendation response time will not significantly deteriorate.
3) It predicts joint probability distribution over more than a single variable at a time, which necessitates finding the affecting variables of the variables to be predicted. This is used for inter-agent communication.
4) It optimizes the use of a model's variables so that every attribute, regardless of whether it is observed with certain evidence, can serve as output for prediction. The intelligence system uses probability distribution marginalization to reduce attributes, thereby dynamically (temporarily) changing the state of evidence, but without significant bias in order to predict attributes values even if their states is known.

6.2.4.4 Inference Agents Cooperation

One of the most important features of the AI inference system 520 is to be able to predict in the domain of one context when evidence is present in other contexts. For example, in the e-commerce field this may be referred to as doing cross-category recommendation. As mentioned above, each inference agent uses a single Bayesian model for prediction. This model has a particular context and its attributes are features that describe this context in various ways. For instance, an agent for dairy foods has attributes that characterize dairy products. Therefore, there is a certain context attributed to every agent. Additionally, the agents in a pool of inference agents *cooperate* by passing probabilistic messages to each other in order to give a cross-context prediction capability. It relies heavily on the communication network mentioned above for allowing different inference agents to influence, i.e., spread their beliefs or predictions, to other agents, even for agents that do not have any deterministic evidence (i.e., observations about the actions of current on-line users).

6.2.4.5 Agent Hopping

"Agent Hopping" pertains to the formation of a broad-recommendation to the on-line user based on the collective inferences of many inference agents, while also adhering to the main objective (as always) that the recommendation is based on prediction, which is based on the evidence at hand and thus fits the particular user. The communication of these inferences occurs as a function of the relationships between inference agents. Like the learning agents of MLS 510, there are two ways in which inference agents are related. The first is a *logical relationship*, which is taken directly from the underlying logical hierarchy according to which the context categories of all agents are interrelated. The second is a *topological relationship*, which is based on a link structure that is statistically formed during the operation of the off-line machine learning system.

For example, as with the learning agents, Figure 7A can be used to depict a topology of inference agents. In this example, we see for instance, agent A2 702 is related to agent A1.DAR 704, which suggests that there is a strong association (the strength of which is available to the two agents) between patterns of product orders on the domain of A1.DAR (which represents dairy products) and on the domain of agent A2 (which represents frozen foods). Additionally, again as with the learning agents, Figure 7B can be used to depict the logical hierarchy of the inference agents.

The *Agent Hopping* algorithm carries out a hopping sequence through the topology of

inference agents. According to the algorithm, a sequence of agents is automatically selected, for instance, agent A2.BRK, A1.MTS, A2.ICE, etc., and eventually a recommendation is made based on the available evidence of the inference agents selected (i.e., not hopped over). The selection process is based on a dynamically controlled series of tests that are made once the hopping reaches a certain agent.

The tests include:

1) checking the amount of evidence at the linked peers;
2) determining the information value of this evidence (which may be probabilistic);
3) performing voluntary evidence transfer from the source agent to the hop-destination agent;
4) ensuring that no agent is visited more than once; and
5) traversing the logical hierarchy upward and downward according to logical relationships.

These operations are parameterized so that a variety of recommendation types can be achieved. For instance, an agent does a recommendation for a product from its context domain when it recursively digs downward, always choosing the most likely path to the next relevant agent, and only makes a recommendation when reaching the lowest level agent on this dynamically selected path. The fact that the path is downwards guarantees an increasing level of specificity, which eventually leads to particular products in the hierarchy. The same is true for the opposite direction. Namely, going upward in logical links, hopping at each step leads to a broader view. When the required level of broadness has been reached, the hopping starts going downwards as mentioned above, digging deeper inside and then only recommending at the last agent. This time the recommendation comes from a totally different branch in the hierarchy, which is still related (via a path) to the original agent. Therefore, the hopping path is state or evidence dependent, i.e., it is *not* hard-wired but rather each session may lead to different hopping paths since each session may have a different state of evidence.

Agent hopping is used to form a broad recommendation that spans the recommendations of multiple agents and aggregates them into one recommendation. Beyond this, the intelligence system 200 may make a prediction or recommendation on the same context by more than one agent by using "agent voting". For instance, suppose there is a variable named *abandoned* which may take a value of "yes" or "no" and which indicates whether an on-line user will abandon his/her electronic shopping cart and not buy at all. Let there be a network of *n* agents

each with a different context, while all having the attribute *abandoned* in addition to their specific context attributes. An "agent voting" algorithm combines the weighted predictions of all *n* agents on the variable *abandoned* and outputs a single yes or no prediction. The algorithm takes as a weight of an agent the confidence that it has in its prediction. This confidence is defined as the L1-norm of the *score* vector (see Section 6.2.4 regarding classification and prediction modes). The algorithm then uses one of the following possible decision modes:

1) Majority Vote: outputs the most common decision amongst all *n* agents.
2) Maximum Confidence Vote: outputs the decision of the most confident agent.
3) Weighted Confidence Vote: computes the weighted average of the predicted probability distributions over all *n* agents and then uses the MAP-decision on the resulting probability distribution.

The second decision mode above is preferred.

6.2.5 IS 520 in Details

An ORB (again, such as Voyager™ by Objectspace) provides, among other things, remote-Java-method invocation between Java objects on separate Java virtual machines (VMs) to facilitate the IS inference process. During this process a variety of components are used, and described in table 1500 of Figures 15A-C and shown in IS 520 block diagram of Figure 17. With respect to table 1500:

1) A component is a modular sub-part of the IS 520 or of the AI system as whole.
2) A QRC™ agent is an XML-based object-oriented communication messaging protocol, a table 1600 describing various QRC™ agents is shown in Figure 16. Different AI components communicate using this protocol.
3) Client stands for any external event-generating server, for instance, an Internet server that passes "product-order" events made by on-line users (e.g., client 154 and user 110 of Figure 2).

The interaction between components can be appreciated with respect to the IS 520 block diagram of Figure 17. For instance, table 1500 indicates, in row 1510, that the inference manager (IM) 1710 sends a QRC™ agent to CMR 1708 to request a net structure object. When reading table 1500, the "Interaction Type" of each component is with respect to another component (not the component's partner). Additionally, with respect to table 1500:

- *Accept/Send* means component does this action.

- *Invoke Method* means component invokes method locally (not through Voyager™)
- *Remote Method* means component invokes method through Voyager™
- *Method Invoked* means a partner invokes method on the component

Concerning the Number column:

- *1* means there is a single unique component in the AI system
- *k* means there can be $k > 1$ components
- *m* is the number of Bayesian models
- *s* is the number of on-line sessions

As previously discussed in section 6.1, the CMR 1708 (denoted as 1208 in Figures 12A and 12B) is responsible for maintaining all the AI Bayesian models and serves as the librarian of the AI system 220. Both MLS 510 and the IS 520 need to access the Bayesian models and they do so via CMR 1708. The inference agents' primary interaction with CMR 1708 is in ensuring that they have the most recently updated Bayesian models so that their recommendations will be more accurate and up to date. The IS 520 interaction with CMR 1708 is in reading models. As described in section 6.1 above, the MLS 510 includes multiple learning agents and is concerned with updating models by writing them to CMR 1708.

In addition, the CMR maintains an XML representation of an *agent-network topology* (also referred to as the *net*) of the inference agents, e.g., 1730, 1740, and 1750. This topology dictates the interaction and communication direction between pairs of inference agents 1730 and 1740, for example, and is based on the findings of an "automatic building" process, which is the initial process of MLS off-line learning. Upon initialization, the CMR reads the net hyperlinks XML file, produces a net flow file, and builds a net structure object. During IM 1710 initialization, CMR 1708 receives a request (via QRC™ agent) to supply the net structure to IM. This process is discussed in more detail below with respect to the scenarios of the IS. During the operation of the AI system 220, the Bayesian models are updated by MLS 510, wherein CMR 1708 is in charge of sending messages of model updates to the IM 1710 (or IMs). The IMs 1710, in turn, let the relevant inference agents, via remote method invocation on the CMR, obtain the actual updated models, which is also discussed in more detail below with respect to the scenarios of the IS.

Files containing the Bayesian models are organized within a directory hierarchy. The net structure object (e.g., a hash table) lists all hyperlinks, shared attributes of each hyperlink and the models coupling. Semaphores arbitrate the multiple Java threads that cause the CMR

to receive/send models. A flow-handler computes the direction of evidence flow over each hyperlink. This is used for involuntary evidence propagation. An example of a flow network is depicted in Figure 7A. In this example, the agent A2 has a *downstream* neighbor A2.EN2 and an *upstream* neighbor being A1.DAR.

6.2.6 Inference Manager (IM) 1710

IM 1710 is the gateway for its set of agents to the rest of the server. As indicated in table 1500 of Figures 15A-15C, there may be several IM's. Sessions are split over these IMs so that each session obtains its intelligent resources from a single IM 1710. Each IM has its own set of inference agents 1730 and each inference agent serves all possible sessions that are served by its respective IM 1710. Each IM receives QRC™ agents from other AI sub-parts, such as the CMR 1708, learning managers 1202 (see Figures 12A and 12B), AI business objects, which interface IS 520 to other non-AI sub-systems. As shown in table 1500, the IM receives QRC™ agents of the following types:

1) user driven events, that originate from the clients;
2) recommendation requests from AI business objects; and
3) model update notifications initiating from the CMR.

In order to handle the above requests asynchronously and concurrently, an IM 1710 delegates the responsibility for the actual handling of the events to its inference handlers (IH) 1720 and 1722, and each IH runs in its dedicated Java thread. The his 1720 and 1722, for example, are maintained in a blocked size pool that permits a balanced parallel distribution of all requests handled by a single IM 1710. The IM keeps a representation of its net topology of agents, wherein different agents may be placed on different Java Virtual Machines. The IHs need to know about the location of these agents in order to perform global operations which involve more than a single agent, for instance, to cause voluntary absorption between pairs of agents, to query multiple agents in a sequence based on the Agent Hopping module, and so on.

The structure of the net topology of agents is represented as a list of URL proxy addresses of all inference agents of the relevant IM 1710. Every IM uses its respective hyperlink file as the source of knowing which inference agents to create and deploy on line. Each IH, in a pool of IHs, is in charge of activating the inference agents for answering a recommendation request. A hyperlink is defined between two agents, or more precisely between their respective models, for which there needs to be a link of communication.

Hyperlinks are represented in the AI system 220 both in memory inside the data

structure of the IMs and also in an XML file format is shown in Figure 8. The file of Figure 8 is read by the CMR and can either be formed manually or be generated via the MLS off-line model building process discussed in section 6.1.8, given the proper data and domain definition files. As seen in Figure 8, a single hyperlink defines a connection between MODEL1 and MODEL2 based on a shared set of attributes whose names is listed under SHVARS. The link has a coupling that is defined as the mutual information I(S1; S2) where S1 is the subset of shared variables originating from MODEL1 and S2 is the subset originating from MODEL2.

6.2.7 Inference Handler (IH) 1722

The IH 1722 is responsible for performing operations on the set of inference agents (e.g., IA 1730, IA 1740, IA 1750) dedicated to its IM 1710. The IH handles operations such as business object recommendation requests , which may involve a series of operations on different inference agents. The result returned from each inference agent could determine the selection of the next inference agent upon which to operate. This form of activity requires the IH 1722 to run asynchronously from IM 1710, keeping IM 1710 free to process further incoming requests. To do so IH 1710 operates in its own dedicated thread, using IM 1710 services (i.e., methods) to access and traverse the agent network. When handling QRC™ agents that require a reply (such as recommendation requests arriving from AI Business Objects), an IH is responsible for sending the reply to the intended listeners. Since some of the high level operations of IH 1722 could be application dependent, it is possible to create specialized IH Java objects that inherit all characteristics from the general IH, but that also contain additional application or client specific capabilities.

6.2.8 Inference Agent (IA) 1730

Each inference agent (e.g., 1730, 1740, and 1750) is responsible to deliver predictive inference capabilities based on a single Bayesian model. Furthermore, each inference agent serves all on-line sessions to which its IM is in charge of delivering intelligent services. Each inference agent 1730 remembers the state of evidence for each such session separately, so that when doing predictive inference it gives out special recommendations which are personalized to the profile of the particular user (e.g., user 110) on each session.

The main tasks of an IA are:

1) Loading its model, ensures prompt replacement of model when the CMR has updated it.

2) Answering predictive queries about any attribute in its model, understands the various

recommendation requests through an inference API, and handles these requests for every session.

3) Maintaining the state of evidence, deterministic and probabilistic, concerning the model, for every session.

In order to handle multiple sessions, an IA keeps a list of inference session objects that record and maintain the state of evidence for each session, discussed above in section 6.2.7.

We now describe in finer details the functionality of the IA:

6.2.8.1 Involuntary Evidence Propagation

With regard to the discussion of IS components in section 6.2.5 and table 1500, the CMR was mentioned to create the network flow file and object. This flow object governs the direction of flow of evidence between pairs of agents. This transfer of evidence is done in a passive way and hence is denoted as *involuntary* evidence propagation. It does not take a query request to cause this flow, but rather any evidence event as intercepted by an IM 1710 gets placed on one or more inference agents, which in turn changes their state of belief, thereby initiating a wave of involuntary evidence propagation.

Evidence is propagated by first converting the evidence to a probability distribution over one or more attributes. Both the sender and receiver of this evidence must have a set of shared attributes over which this distribution is well-defined. Each inference agent keeps a list of neighbors, including the direction of evidence flow from/to each of them. Session evidence (which is client driven) enters the server 162 and gets sent to the appropriate inference manager (e.g., IM 1710). The IM sends the evidence (via remote method invocation) to one or more inference agent as "Real evidence", namely, evidence which directly reflects a real event that the on-line user 110 has generated, e.g., product order, category change, etc..

The real evidence is immediately translated into deterministic or probabilistic evidence, in the case where there needs to be multiple values placed on a single attribute. The inference agent then uses its IE (e.g., IE 1734) to re-compute the probability distribution over all shared sets with all its neighboring inference agents. The inference agent passes (i.e., propagates) evidence only to those neighboring inference agents which are "downstream " with respect to it, based on the flow structure.

Each IA can handle three forms of evidence input:

1) real deterministic evidence that represents a name of an attribute in a Bayesian model

and a single value in the domain of this attribute;

2) real probabilistic evidence that is a probability distribution with uniform weights over a subset of values in the domain on which evidence is to be placed; and

3) probabilistic evidence that is passed from one agent to another transferring beliefs concerning the probabilistic distribution of their shared variable(s).

Probabilistic evidence is transferred as a list of discrete functions, whose product is the probability of the shared variable(s). This multi-function form is done to avoid the creation and transformation of high-dimensionality probability functions.

When receiving new evidence the inference agent performs the following steps:

1) It identifies the inference session object for which new evidence has arrived.

2) It requests the session object to update itself, which includes the following steps:

   a) Update the state of evidence. In case the evidence is probabilistic, it is added to the list of probabilistic evidence. If a less recent probabilistic evidence for the same variable(s) exists it is replaced by the new evidence. In case of real evidence, the evidence is translated into probabilistic or deterministic evidence used to update the existing evidence state.

   b) Update the timestamp of last arriving evidence, to the current time stamp. (Time stamp is a local counter independent of the hardware clock).

   c) Empty the calculation cache, since it is no longer valid.

   d) Recalculate Gamma (see section the section on Gamma below).

3) Propagate probabilistic evidence to each "downstream" neighbor inference agent, concerning the probability distribution of its shared variable(s) with this agent. This propagation is performed as follows:

   a) Get a snapshot of the session's current state.

   b) For each downstream neighbor, calculate the distribution of its shared variable(s), and represent it as a list of discrete functions that are sent as probabilistic evidence. The evidence will normally be sent asynchronies, but a synchronous mode is available as well and is used in instances where it is important to guarantee that evidence arrives prior to answering a query.

6.2.8.2 Voluntary Evidence Absorption

The involuntary evidence propagation mechanism described in the last section aims at

spreading information-rich evidence to as many inference agents as possible. The mechanism operates in a background manner and information follows in the direction of the flow structure. This flow direction ensures that an agent which is more coupled, i.e., has neighbors with a high mutual information coupling over its shared attribute sets, gets to be the sender of information to its downstream neighbors. As indicated before, this operation is independent of the query activity that agents undergo.

In the event of a query request to an inference agent, it is best (for accurate predictions) that the agent has as much evidence as possible. As far as real evidence, this is limited to whatever the agent obtained from its IM 1710, which is basically all relevant information seen so far through client-triggered events. Thus, the agent relies heavily on the real evidence it has received. However, probabilistic evidence from its neighbors ads another dimension of information. The involuntary evidence flow could have delivered evidence to it, but not necessarily. For instance, this inference agent could have been an upstream neighbor to all of its neighboring inference agents, in which case it never received evidence involuntarily, even if these neighbors had supporting real evidence.

Moreover, the quality of the evidence plays an important role here, since evidence is probabilistic and thus has a certain information-content, i.e., parts of which are noisy. For this reason, there is an extended ability by which an inference agent can voluntarily absorb evidence from "trustworthy" neighboring agents. The meaning of trustworthy is based on a combination of the coupling of the hyperlink between the agent and its neighboring agent and also on the confidence in the neighbor at the time of absorption. It is also a function of the strength of evidence that the neighbor has. The latter being inversely proportional to the distance that it propagated since the time it was first placed as real evidence at some terminal point in the agent network.

Voluntary evidence absorption is performed by an inference agent prior to performing one or several possible inference computations. The idea is to absorb evidence from a set of agents that does not include the agent's upstream neighbors, since they already involuntary propagated evidence to the agent. The absorption is performed recursively so that an agent X receiving an absorption request from agent Y performs an absorption itself from its own neighbors (not including Y) prior to computing the probabilistic evidence that it returns to Y. In order to keep the computation efficient, a timestamp for each probabilistic evidence is kept, which is the value of a time counter of the agent that originally sent it (note that an agent

maintains a time counter per each session).

When the absorbing agent *Y* enters a request to *X*, it sends to it the timestamp of the last evidence that it got from *X*. Upon receiving this absorption request, *X* performs the following:

1) **If** the received timestamp is not older than the timestamp of the last evidence received on this session from this particular neighbor, then the agent returns with an indication that no absorption is necessary. Again, this is because the requesting agent already has the most recent belief (or evidence) based on its timestamp.
2) **Else** the agent sends an absorption requests to all of its remaining neighbors.
3) If all the neighbors returned with an indication that no absorption is necessary, the agent checks its cache for the requested calculation result, and returns it if it is found. Otherwise, after receiving all the requested probabilistic evidence, the session object is updated, i.e., the agent *X* changes its internal state due to the absorbed evidence.
4) The requested calculation (i.e., the prediction of the shared variables with the requesting agent) is performed, and the result is returned to the requesting agent *Y*.

6.2.9 Answering Simple Queries (Prediction, Map, Best-Explanation)

When receiving a simple query request the IA will perform the following:

1) Get a snapshot of the session's state, which includes all of the session's information. The idea here is to capture the state of the session, making the calculation that is about to be performed insensitive to incoming evidence or ongoing concurrent queries.
2) If a result for the requested calculation is found in the snapshot cache, then it is immediately returned as the result of the computation, which saves the need to reactivate the inference engine and re-compute the same prediction again.
3) Remove probabilistic/deterministic evidence concerning the queried variables. Here one uses the ability of a Bayesian model to have both input and output on any attribute. Removing all evidence on the queried variable ensures that the returned prediction is not simply the observed input evidence, but rather a predicted value based on what the model learned. Such value is clearly of higher interest to the on-line user than what it already has or knows.
4) If any evidence has been removed, recalculate Gamma given the altered set of evidence. (See below for Gamma explanation).
5) Perform the actual computation using the JBayes Handler (JBH).
    a) Allocate a free inference engine.

b) Create an Inference Graph, based on the engines Bayesian model.

c) Set the Deterministic Evidence in the Inference Graph.

d) Calculate Gamma (if it is null).

e) Perform the actual computation on the inference graph using the Bucket-Elimination algorithm.

6) Cache the result of the calculation, only if:

a) no evidence has arrived since the time in which the snapshot, upon which this calculation is based, was taken; and

b) a more recent result for the same calculation has not been placed in the cache.

6.2.10 Answering Recommendation Queries

The returned objects supplied to the inference manager are lists of values obtained from a predictive query operation, and are referred to as *recommendations*. In the preferred embodiment, a *recommendation* has both a value and a score, for example, {Coffee, 0.56}. A recommendation can be given on any value of a Bayesian model's attribute. The recommendation object contains the following information:

1) Subject of recommendation: the actual recommended value (e.g., A1.BAK.BAKMUF.10499)

2) Context of recommendation: the attribute whose value was recommended (e.g., A1.BAK.BAKMUF)

3) Local score: the score of the recommendation indicating the strength of belief in the specific value of the attribute. (e.g., 0.343).

4) Context score: this is a vector of scores indicating the strength of belief in the context in which the recommendation is given. If the recommendation was performed after a certain traversal of the agent network, the context score represents the strength of belief in each traversal step. (e.g., < 0.33 , 0.56, 0.77 >).

5) An indication of whether any real evidence existed in the model where the recommendation was performed.

A group of recommendations is kept in a modular object called a *RecommendList*. Such a list can be collected at a given Bayesian model or during the execution of a high level recommendation algorithm such as the Agent Hopping algorithm. Currently, the recommendation list is sorted by context score as a primary key and local score as a secondary

key. Other forms of sorting are also possible.

The inference currently handles five types of recommendation requests:

1) Var-Recommend: A recommendation on the values of a specific variable.

2) Shallow-Recommend: A recommendation on the attributes of a model. This recommendation is produced by performing a Var-Recommend on the Class variable of the model.

3) Deep Recommend: A recommendation on a selected number of intrinsic attributes of a model. First the selected attributes are chosen by performing Shallow-recommend on the model, then a Var-Recommend is performed on all of the chosen attributes.

4) Agent_Hop_recommend: A higher level recommendation that is produced by an algorithm that traverses the agent network.

5) Agent Vote recommend: A group of agents that can predict a given attribute are activated. The final recommendation result is taken as a function of the confidence of each of these agents and the number of agreeing agents.

All recommendation results are returned in the form of a recommendation list.

There are two modes of recommendation:

1) PROBABILITY mode: The local score of recommended subject values is determined by their probabilities.

2) DELTA mode: The local score of recommended subject values is determined by the change in their probability values given the evidence in the model, relative to their original probabilities without the evidence. This mode enables to concentrate on directions of change caused by the session evidence. In the case of DELTA mode the local score of a subject S is calculated as either:

1) Relative:

(Pevidence(S) – Pno-evidence(S)) / Pno-evidence(S)

or by the formula

2) Omega:

Pevidence(S) * *log* (1/ Pno-evidence(S))

In the case of relative mode, filtering is performed to disregard a subject having a no evidence probability that is below than a certain threshold. This threshold is inversely proportional to the number of possible values of the queried attribute. Both modes aim at giving a higher score to attribute values for which the evidence-distribution gives a high

probability while at the same time the no-evidence distribution gives a low probability. The omega mode is similar to an information retrieval measure of importance of word-based keys.

6.2.11 Gamma

Gamma represents the total probabilistic evidence for a given Bayesian model, corresponding to a specific session state. Gamma is represented as a list of discrete functions. It is calculated from a set of probabilistic evidence on different variables, and is effected by existing deterministic evidence in the Bayesian model. The calculation of Gamma is performed as follows:

1) Set the available deterministic evidence.
2) Extract the union, Vu, of variables participating in the available probabilistic evidence.
3) Calculate the joint probability of Vu, and represent it as a set of discrete functions, to avoid the creation of a single function of many variables.
4) Invert the latter discrete functions, thus producing the denominator of Gamma.
5) Add the discrete functions of the probabilistic evidence, thus adding the numerator of Gamma.
6) The product of the final set of discrete functions is Gamma.

Gamma is used in every inference computation by inserting its functions into the "buckets" during the Bucket-Elimination (BE) algorithm, as described in the next section.

Data structures for GAMMA include:

1) A list of neighbors (stating the propagation flow direction).
2) A list of Inference Session states.
3) A JBayes Handler (responsible of performing the calculations).

As mentioned in several sections above, a primary mechanism of the inference system 520 is the ability of its distributed resources, namely the Inference Agents, to communicate and pass on their beliefs to their neighbors. Having this ability is paramount since it enables completing patterns of values over attributes across the domain of all models as a whole. For instance, having a partial observation in the domain of model A1 is still sufficient to produce a prediction over the domain of model A2. This mechanism is what ties all models together into a single domain over which inference can be achieved.

The primary mechanism of communication is the transfer of probabilistic evidence. The mathematical notion of transfer of probabilistic evidence is well known. Given two models, X and Y, with a shared attribute set {S1, S2} and suppose model Y has attributes Y1, Y2 and

Y3, then evidence that flows from X to Y has the following effect on the probability of any attribute, say Y1, of model Y:

$$P^*(Y1) = \sum\{S1,S2,Y2, Y3\}\ P(Y1, Y2, Y3 \mid S1, S2)\ P^*(S1, S2),$$

where P*(Y1) is the new distribution after absorption of the evidence, P*(S1, S2) is the propagated evidence distribution based on model X and P(Y1, Y2, Y3 | S1, S2) is the conditional probability distribution based on model Y.

Note that using Baye's Rule P(Y1, Y2, Y3 | S1, S2) may be expanded as P(Y1, Y2, Y3, S1, S2) / P(S1, S2), where the numerator is called the *joint* probability distribution of model Y.

<u>Gamma Formal Definition:</u> Let Gamma(S1,S2) = P*(S1, S2) / P(S1, S2)

Thus we now have the general separated form (for the above example):

$$P^*(Y1) = \sum\{S1,S2,Y2, Y3\}\ P(Y1, Y2, Y3, S1, S2)\ Gamma(S1, S2),$$

which is expressed in terms of the receiving model's joint probability and the receiving model's Gamma function.

Gamma is composed of a set of numerator functions, in the example these are functions whose product is P*(S1, S2), and denominator functions whose product is P(S1, S2). The joint probability distribution P(Y1, Y2, Y3, S1, S2) is the core of model Y. Based on it, any sets of attributes in Y can be predicted by a simple operation of marginalization.

In the preferred embodiment, the IS 520 uses the BE algorithm to do this marginalization. In this algorithm, probability functions whose product forms the joint probability distribution are initially placed in "buckets". Since the effect of evidence is localized to the Gamma(S1, S2) term, it is possible to place Gamma's functions, i.e., both the numerator and then the inverted denominator functions, into the buckets using the same rule. The BE algorithm's computation then proceeds in a regular manner yielding the probability distribution of the variable Y1.

Note that the additional steps mentioned below with respect to the inference agents' computations will need to take place prior to forming the buckets. These steps rely on being able to treat Gamma's variables as deterministic evidence variables, thereby making use of the important "d-separation" step that makes the computation much more efficient (d-separation is

a well-known concept in the field of Bayesian Networks).

Generally, in the case of more than one neighbor to Y, the numerator functions of Gamma will come from various models. For instance, suppose there are 3 shared attributes, model Y shares S1, S2 with model X, and shares attribute S3 with model Z, then Gamma(S1, S2, S3) = P*(S1, S2) P(S3) / P(S1, S2, S3). If the same attribute S is shared with more than one model, the numerator will consist of a product of functions including probability functions P(S) from all relevant models.

6.2.11.1 Gamma Object Definition

The Gamma object encapsulates the influence of probabilistic evidence on calculations within a given Bayesian model:

1) Gamma = { Numerator , Denominator }
2) Gamma.Numerator = A set of probability functions (probabilistic evidence functions), received by the agent from neighboring agents.
3) Gamma.Denominator = The joint distribution of the union of variables that appear in the Numerator (given the existing deterministic evidence).

6.2.11.2 Gamma Initialization

Gamma is initialized to be Gamma = {null, null}

6.2.11.3 Update of Gamma by Evidence

Upon arrival of new evidence the numerator or denominator of Gamma may be set to null in order to force their recalculation. Arrival of deterministic evidence sets both numerator and denominator to null. Arrival of probabilistic evidence sets only the numerator to null, while the need for denominator recalculation is determined at the time of Gamma recalculation.

6.2.11.4 Using Gamma

The actual calculation of Gamma varies according to the type of performed operation. However, all forms of operations use the Gamma recalculation method described below, wherein the different forms of Gamma calculations are:

1) Calculating Gamma for Prediction purpose.
2) Calculating Gamma for Recommendation purpose.
3) Calculating Gamma for Propagation purpose.

The core recalculation function performed by the engine is presented in pseudo code format:

recalculate-Gamma (current-Gamma, Probabilistic-Evidence, Deterministic-Evidence)

1) Obtain current-Gamma.Numerator from Probabilistic-Evidence.

2) Let Vn be the union of all variables appearing in current-Gamma.Numerator.

3) If (current-Gamma.Denominator != null), then
Let Vd be the union of all variables appearing in current-Gamma.Numerator.

4) If (current-Gamma.Denominator == null or Vd != Vn),
then current-Gamma.Denominator = JointProb (Vn | Deterministic-Evidence)

Gamma recalculation is performed whenever a new snapshot of an inference session is taken, and either the numerator or denominator of a session's Gamma equals null. This policy enables the inference engine to postpone Gamma recalculation to the point in time where it is actually needed, thus avoiding unneeded recalculations.

6.2.11.5 Gamma Calculation For Recommendation

Recommendation for an attribute, whether it comes from Var-Recommend, Shallow-Recommend, etc., means outputting a predicted probability distribution for the attribute. A Bayesian model network enables any attribute to serve as an input or an output, but not both simultaneously. Thus, prior to answering such a prediction one must be sure to remove any evidence at hand for the queried variable. This way, the recommendation for the variable will not simply be the evidence that the inference agent has for it. This is done as follows:

1) If probabilistic evidence exists on the recommended variable, produce a marginalized version of the snapshot's probabilistic evidence, with respect to the variable. This gets rid of the variable from the joint probability function.

2) If deterministic evidence exists on the recommended variable, then produce a version of the deterministic evidence without the evidence on the variable.

3) If one of the above conditions has occurred, then recalculate Gamma given the adjusted probabilistic & deterministic evidence.

Thus for a recommendation on variable *X*, given a snapshot containing Deterministic & Probabilistic evidence, the following pseudo code (i.e., process) is performed:

recalculate-For-Recommend (X, Snapshot)

(Note: This method calls the previously mentioned method recalculate-Gamma method.)

1) X-Gamma = current-Gamma

2) If (Snapshot contains deterministic evidence on X), then
Remove deterministic evidence on X from snapshot.

3) If (Snapshot contains Probabilistic evidence on X), then

marginalize all snapshot probabilistic functions with respect to X.

4) If (one of the above conditions occurred), then

recalculate-Gamma (X-Gamma, Snapshot.Prob-Evidence, Snapshot.Det-Evidence)

6.2.11.6 Gamma Calculation For Propagation

Consider a scenario in which Model 1 propagated to Model 2 on attribute A, and at later point in time attempts to absorb evidence from Model 2 on A. Without care, the result might be that Model 1 absorbs the exact same evidence that it has propagated earlier. Thus, a preferred algorithm dictates that in such case Model 2 must lift evidence P*(A) that it has received from Model 1 prior to computing the probability distribution over A. Therefore, when propagating probabilistic evidence to a neighboring model Ni on a set of shared variables Vs, if probabilistic evidence from Ni on Vs already exists, then it is removed from the snapshot and Gamma is recalculated. This is done to avoid the situation in which one inference agent passes to the another the probabilistic evidence received from that other agent.

For propagating evidence to neighbor Ni, on shared variables Vs, given a snapshot of Deterministic & Probabilistic evidence, the following pseudo code (i.e., process) is performed:

recalculate-For-Propagation (Ni, Vs, Snapshot)

(Note: This method calls the previously mentioned method to recalculate-Gamma.)

1) X-Gamma = current-Gamma

2) If (Snapshot contains Probabilistic evidence on Vs that was originated from Ni), then, remove it from snapshot .

3) **recalculate-Gamma** (X-Gamma, Snapshot.Prob-Evidence,Snapshot.Det-Evidence)

6.2.12 Inference Session

An inference session object (ISO) maintains the state of a session, including its evidence, Gamma function, and calculations cache. The ISO is also responsible for keeping its local time counters (since sessions are independent), so each ISO keeps the time stamp of its last arriving evidence. The data structures used for ISO include:

1) the contents of the real evidence;

2) a list of deterministic evidence;

3) a list of probabilistic evidence;

4) a cache of previous calculations (including Gamma); and

5) a list of previously handled events.

6.2.13 JBayes Handler

The JBayes Handler (JBH) is responsible for performing all calculations concerning a single Bayesian model. JBH maintains a pool of IEs (e.g., IE 17340) which perform the actual calculations - predictions, MAP decisions, Maximum Expected Utility and Gamma calculations. The JBH also performs other special purpose tasks that involve direct interaction with the IE. In the preferred embodiment, a public domain distributed software named *JavaBayes* is extended and used as the basis for the inference engine. For instance, such extensions include the ability to use probabilistic evidence, to predict multiple attributes simultaneously and to perform dynamic simplification of computations as part of a tradeoff with the available memory resources. The data structures used for JBH include a pool of engines of a specific model.

6.2.14 Inference Engine (IE)

An inference engine (e.g., IE 1734) is the primitive module (or component) for performing various process involved in making an inference. Given a Bayesian model, a state of evidence (or observed values) for any subset of attributes of the model, and given a query request which may be a request to predict the probability distribution of a single or multiple attributes, a Maximum Aposteriori Probability (MAP) classification request, or a Maximum Expected Utility (MEU) decision, the engine proceeds with the requested computation.

The basis of the IE, computations are based on the Bucket Elimination Algorithm, previously mentioned. Extended use of standard ideas from probability theory as well concepts of "*d-separation*" are used to make computations more efficient. The low level computations of the inference system 520 are:

1) the marginal posterior distribution of a single variable;
2) the marginal joint distribution of a set of variables;
3) the MAP of a set of variables; and
4) the MPE (Most Probable Explanation) of the entire model.

These low level computations are performed using the Bucket Elimination Algorithm, with the extension of using Gamma functions, and the "Killing Algorithm", which optimizes the tradeoff between accuracy and computational resources such as memory and CPU time. All of the listed computations are performed similarly using variations of the extended Bucket Elimination algorithm.

6.2.14.1 Extended Bucket Elimination (BE) Algorithm

As mentioned above, IS 520 adds extensions to the standard package JavaBayes to enable additional computational modes. The standard JavaBayes software implements the BE algorithm. In order to reduce the calculation time when doing inference (e.g. prediction) the standard JavaBayes uses the interdependencies information inherent in the graphical structure of the network in order to determine which variables are relevant, i.e., have influence on the queried variable. This is done in the constructor of a class called "Ordering", through a call to a method *all_affecting (objective_index),* which is in the Java class DSeparation. The method is based on the standard concept of d-separation, see J. Pearl, *Probabilistic Reasoning in Intelligent Systems*, (Morgan Kaufmann Publishers 1988).

The extensions to the BE algorithm arise from the following needed capabilities, which were missing in the JavaBayes package:

1) predicting joint probability distributions for multiple attributes, which necessitates finding the affecting variables to all of the variables to predicted, and
2) handling probabilistic evidence in addition to deterministic evidence.

Inputs to the extended BE algorithm include:

1) a set of queried variables (possibly containing a single variable);
2) the Bayesian model's deterministic evidence;
3) a Gamma object containing the Gamma functions, only if probabilistic evidence exists in the model; and
4) input modes indicating the type of prediction (e.g., posterior distribution, MAP or MPE).

Outputs of the extended BE algorithm include:

1) When computing the marginal posterior distribution of a single variable, a single discrete function is returned.
2) When computing a marginal joint distribution of a set of variables, a set of discrete functions is returned whose product equals the requested joint probability.
3) When computing the MAP of a set of functions, the most probable combination of values is returned (A Vector of {Variable, Value} pairs).
4) When computing the MPE of a model, the most probable combination of variable values is returned (A Vector of {Variable, Value} pairs).

The extended BE algorithm includes the following steps:

1) Step 1: Determine the set of "affecting variables" with regard to the queried variables, based on the deterministic evidence (observations) and Gamma. The set of "affecting variables" consists of the union of variables that are not d-separated from at list one queried variable. Variables for which probabilistic evidence exists are treated as observed as far as d-separation is concerned.

   a) The standard d-separation algorithm can be used as is, in the prediction of joint probability and the existence of probabilistic evidence. The only required modifications are in its form of activation, as explained above. The "participating variable" set is the set of variables that will participate in the bucket elimination phase. This set is defined as the union of the "affecting variable" set and the set of variables participating in Gamma.

2) Step 2: Filter Gamma functions as a result of d-separation. Functions appearing in the denominator that do not involve any of the "affecting variables" are removed from Gamma.

3) Step 3: Produce an ordered set of variables, and a set of functions that will participate in the bucket elimination phase. The set of variables is initialized to the "participating variable" set, induced in the previous step. The function set is initialized to include the conditional probability tables (CPTs) of the "affecting variables" and the set of filtered Gamma functions. The above sets are induced in an iterative manner, wherein each iteration the following is performed:

   a) The algorithm attempts to create an ordering of the current variable set based on a "graph moralization" algorithm (see R. Dechter, *Bucket Elimination: A Unifying Framework For Probabilistic Inference*, Uncertainty in Artificial Intelligence UA196, 211,219 (1996). In the preferred embodiment, the moralization algorithm has been altered to receive a set of functions, replacing the model CPTs. As a result, Gamma functions are taken into account when inducing the order of variables. The altered moralization algorithm receives a set of variables and a set of functions, from which it builds a "moralization graph". In the graph, each variable is represented by a node. Two nodes are linked if their variables appear in a common function. After initialization, the altered moralization algorithm chooses, in an iterative manner, the node having the minimal number of links. It eliminates it and links all of its neighbors to

one another. The returned order of variables is the order of elimination. Since the process of node elimination corresponds exactly to the BE algorithm, which is executed at some later time on the same ordered set of variables, it is possible to identify the maximum bucket dimensionality at this early stage of moralization. Our "kill" extension, explained below, uses this to identify when a certain bucket is about to exceed a certain determined threshold and then kills the variable.

b) If during the creation of the moralization graph, one of the graph nodes exceeds a certain threshold known as the "maximum bucket dimensionality", the ordering attempt fails. The maximum bucket dimensionality represents the maximal allowed size (number of elements) in the "Lambda" function, which every bucket produces. This size is equal to the product of the cardinality of all variables of all functions in the bucket. The dimensionality of a variable's node in the moralization graph is equal to the dimensionality of its bucket in the bucket elimination phase.

c) If the computation of ordering succeeded, the iterative process stops, and the current ordered variable set and function set are chosen to be used in the bucket elimination phase.

d) Otherwise a "kill" is performed on one of the graph-nodes whose dimensionality exceeded the allowed threshold. The killed variable is chosen, in the preferred embodiment, to be that of the highest dimensionality, but other heuristics may be considered as well.

e) Once a variable is "killed", it is removed from the current variable set. All the CPTs and Gamma functions containing the killed variable are marginalized with respect to it while assuming it has a uniform distribution (marginalization is the process of taking the conditional expected value).

4) Step 4: When creating the buckets for the BE process, a bucket is created for each member of the ordered variable set, produced in the previous step. All members of the produced function set (including the Gamma functions) are inserted into the buckets according to the standard BE algorithm.

5) Step 5: The buckets are also reduced according to the standard BE algorithm, wherein in each step, the current last bucket is reduced into a Lambda function that is inserted

into one of the remaining buckets. The only exception concerns the case of a joint probability computation. In this case, the reduction of buckets is terminated at the stage where the only remaining buckets are those of the queried variables. At this stage, the reduction stops and the returned result is the union of functions found in the buckets of the queried variables. Although the joint probability is equal to the product of the function set, it is kept in the form of a function set to avoid the creation of a function of high dimension. When computing the distribution of a single variable, the result function is returned after normalization.

6) Step 6: In case of a MAP or MPE calculation, a backward maximization is performed to induce the most probable value of each variable.

For a detailed example of an inference agent computation, consider the Bayesian network model 1800 of Figure 18A, where the value ranges of the model variables are:

1) Range of A = {a1, a2, a3}
2) Range of B = {b1, b2, b3, b4}
3) Range of C = {c1, c2, c3, c4, c5}
4) Range of D = {d1, d2, d3}
5) Range of E = {e1, e2, e3, e4}
6) Range of G = {g1, g2, g3, g4, g5}
7) Range of H = {h1, h2, h3}
8) Range of I = {i1, i2, i3}
9) Range of Q = {q1, q2, q3}

This example focuses on a case where a prediction on variable Q is performed given the following state of the model:

Deterministic Evidence: H = h1
Probabilistic Evidence: P*(B), P*(D)
Gamma.numerator: P*(B), P*(D)
Gamma.denominator: P1(B), P2(B,D)
Max Dimensionality Threshold = 15

Notice that the product of functions in the denominator equals the joint probability of B, D in model 1800; also notice that the existence of P2(B,D) indicates that B and D are not independent, as can easily be seen by viewing model 1800.

Working the example through the extended BE algorithm:

1) **Phase 1:** Determining the set of "affecting variables":

a) Before activating the d-separation algorithm, set the variables having probabilistic evidence, namely B and D, as observed just for the sake of d-separation. (The observation settings are lifted right after D-separation).

b) After activating the d-separation algorithm the "affecting variable" set is determined to be: **{ A, D, G, E, Q }**. The deterministic evidence on H caused the elimination of H and I while the probabilistic evidence P*(B) on B caused the elimination of B and C.

The "participating variable" set is **{A, D, G, E, Q, B }**.

2) **Phase 2:** Filter Gamma functions.

After filtering Gamma functions that involve only variables in the set {A, D, G, E, Q}, Gamma is then set to:

<u>Gamma.numerator</u>: P*(B), P*(D)

<u>Gamma.denominator</u>: P2(B,D)

Notice that P1(B) has been filtered out, since it involves only a variable which is not an "affecting variable", namely B.

3) **Phase 3:** Producing an ordered variable set.

a) Initialization: The ordered variable set is initialized to the "participating variable" set, namely { A, D, G, E, Q, B }; the function set becomes { P(A|B,D,G), P(D|B,E), P(G|E), P(E), P(Q|A,H), P*(B), P*(D), P2(B,D) }.

b) An ordering of the variables is attempted using the graph-moralization algorithm. During moralization, the node of variable E contains variables {E, D, B, G}, so that its dimensionality (240) exceeds the maximum Bucket Dimensionality, which is set to 15. As a result, variable G is chosen to be killed (since it has the highest cardinality 5). The current sets after killing are:

i) participating variable set = {A, D, E, Q, B}.

ii) Function set =
{P(A| B,D), P(D|B,E), P(E), P(Q|A,H), P*(B), P*(D), P2(B,D)}.

Note the marginalization of functions P(A|B, D, G) and P(G|E) with respect to G are:

i) $P(A|B,D,G) \Rightarrow P(A|B, D) = \Sigma g\ P(A|B,D,G)$

ii) $P(G|E) \Rightarrow \Sigma g\ P(G \mid E) = 1$

Where Σg stands for a summation over all possible values of G.

c) An ordering based on the new sets is attempted. During moralization, the node of variable E contains variables {E, D, B}, although smaller then in the previous step, its dimensionality (48) still exceeds the threshold. As a result, variable E of highest cardinality 4 is killed. The current sets after killing are:

i) participating variable set = {A, D, Q, B}

ii) Function set =

{P(A| B,D), P(D|B), P(Q|A,H), P*(B), P*(D), P2(B,D)}

Notice that P(E) has been completely eliminated.

d) An ordering based on the new sets is attempted. During moralization the node of variable B contains variables {B, A, D}, although smaller then in the previous step its cardinality (36) still exceeds the threshold. As a result variable B of highest cardinality 4 is killed. The current sets after the killing are:

i) participating variable set = {A, D, Q}

ii) Function set = {P(A| D), P(D), P(Q|A,H), P*(D), P2'(D)}.

At this stage the moralization algorithm has completed successfully, so the final ordering is = { D, A, Q }.

4) **Phase 4:** Creating the buckets for the bucket elimination phase:

The produced buckets are:

a) Bucket D: P(A|D), P(D), P*(D), P2'(D)

b) Bucket A: P(Q|A)

c) Bucket Q: empty

Notice that in bucket A, P(Q|A,H) has been reduced to P(Q|A) by substituting the deterministic value of H.

5) **Phase 5:** Performing the bucket elimination phase:

a) Step1: Bucket A: P(Q|A), λ(A),

Bucket Q: empty

b) Step2: Bucket Q: λ(Q)

c) Step3: The product of functions in bucket Q, that is λ(Q) is normalized, such that Norm(λ(Q)) is returned as the result of the algorithm.

6) **Phase 6:** This phase is not run in this example since it is a prediction calculation and not a MAP or MPE.

6.2.15 DRV Inference Agents or Models

As described with respect to the learning agents of the MLS 510 in section 6.1, DRV models are used in the AI system 220 for the purpose of predicting the probability of NULL value for each attribute in any model. This is because a NULL value has an important and strong influence over the prediction accuracy. However, a NULL value can never be entered as a deterministic value, since as long as a session is active it is not possible to state that a user entered NULL for an attribute (i.e. has not selected that category of product). The present invention includes a process for updating update any model's belief concerning any of its attributes taking the value NULL.

The basis of this prediction is information that contains the *entry time* to categories in a Web site, as an example. For instance, consider attribute A1.VEG in model A1 1900 of Figure 19. The process predicts the probability that a person will not select A1.VEG based on the number of times that the person has visited the A1 category and still has not selected A1.VEG. The premise here is that a user is less and less likely to select A1.VEG as the number of times that the user enters A1 and does not choose A1.VEG increases.

A DRV agent handles predicting P(A1.VEG = NULL) based on evidence which reflects the number of times the user entered the parent category A1. An example of a DRV agent DRV.A1 associated with regular Agent A1 is shown Figure 10, but also applies in this case. As seen, there are two types of attributes in a DRV model: the TIME attribute 1002 and the NULL attribute 1004. The NULL attribute 1004 takes the value {NULL ,OTHER} where an OTHER indicates that the user has selected something from the related category while a NULL signifies it has not selected. The TIME attribute 1002 takes the values {0, 1, ..., M}, where M is some fixed number. The time attribute's value x refers to the time that the associated category was entered relative to the entry time of its parent category. For example, placing deterministic evidence of the form: DRV.A1.VEG.TIME = 3 means that the user has entered the A1.VEG category after he entered the A1 category 3 times. An example of such a sequence of activity is: A3, A1, A1.PKG, A1, A1.FRT, A1, A1.SEA. At this point in time, the user entered A1 3 times. Therefore, the likelihood of the user entering A1.VEG will be low.

The inference system distributes such time evidence to all relevant DRV models based on activity events called <CATEGORY_CHANGE> events. These events contain time sequence of category entries, as shown and described above. The format 2000 of this event is

shown in Figure 20.

The IS 520 uses the DRV models 1800 to predict the NULL attribute. The IM 1710 distributes evidence based on category change events to its DRV IAs. In the preferred embodiment, evidence is always placed on the TIME attribute and never on the NULL attribute. For instance, if a current session enters subcategory PKG under model A1 after having entered category A1 twice, then the following evidence is placed: DRV.A1.PKG.TIME = 2 (1002 of Figure 10). The DRV.A1 agent 1000 then computes for each of its DRV.NULL attributes one of the following, depending on where the user is currently (as indicated by the last CATEGORY_CHANGE event received by the IM), without loss of generality we display it for DRV.A1.PKG:

1) Given evidence DRV.A1.PKG.TIME = *Tcurrent*,
   a) if based on the last CATEGORY_CHANGE event it is determined that the user is in category A1, then compute the conditional distribution:
      P^* = P(DRV.A1.PKG.NULL = *NULL* | TIME = *Tcurrent*), but
   b) if the user is outside category A1, then compute the joint distribution:
      P^* = P(DRV.A1.PKG.NULL = *NULL*, TIME ≥ *Tcurrent*)
2) Propagate as evidence this P^* to all "regular" models (non-DRV) having attribute A1.PKG.
3) Upon receipt, a regular model sets P(A1.PKG = NULL) = P^* , and normalizes the probabilities for the remaining values of A1.PKG with a total of 1 – P^*.

Based upon the above algorithm, probabilistic evidence on NULL is placed based on the DRV model's 1800 confidence, which is directly related to the category changes exhibited so far in the current on-line session.

6.2.15.1 Learning DRV models

The DRV models are learned using the same Machine Learning System that learns the entity Bayesian models (see also section 6.1 on MLS 510). However, instead of using the *serial.dat* files, the DRV models utilize category-change information, which is still represented in a serial manner. A module converts this serial stream of category-change information into cases for attributes NULL and TIME by scanning the number of times each category was visited, as seen in the serial stream. When the end of the session activity stream is reached, letting the TIME attribute take the last entry time relative to the parent category (discussed

more detail in MLS section 6.1.9). From these cases, the DRV model 1800 is learned using the same off-line process as the regular, entity Bayesian models. The learned DRV model can then be used by a DRV Inference Agent.

6.2.16 IS 520 API

IS 520 can be viewed as an AI system 200 resource that receives query requests and returns sets of induced recommendations. The available query forms are represented by a set of APIs, which are activated by sending QRC™ agents, to the IS. The result of each API is returned in the form of a ReplyObject, containing a set of recommendation descriptions. IS 520 supports the following five basic APIs:

1) Var-Agent-Recommend:

Parameters:

a) agentName (String) = name of queried agent.

b) attributeName (String) = name of queried attribute within the agent.

c) maxItemsNum (int) = maximum number of returned recommendations.

d) evaluationCriterion (String) = name of the evaluation criterion applied by IS.

Return:

e) A RecommendationList reply object.

2) Shallow-Agent-Recommend:

Parameters:

a) agentName (String) = name of queried agent.

b) maxItemsNum (int) = maximum number of returned recommendations.

c) evaluationCriterion (String) = name of the evaluation criterion applied by IS.

Return:

d) A RecommendationList reply object.

3) Deep-Agent-Recommend:

Parameters:

a) agentName (String) = name of queried agent.

b) maxItemsNum (int) = maximum number of returned recommendations.

c) evaluationCriterion (String) = name of the evaluation criterion applied by IS.

d) maxAttributes (int) = maximum number of recommended attributes within the queried agent.

e) maxAttributeValues (int) = maximum number of recommended values of each

attribute.

Return:

f) A RecommendationList reply object.

4) Agent Hop Recommend:

Parameters:

a) triggerCode (String) = AI code name of a lowest level entity, e.g. a product.

b) maxItemsNum (int) = maximum number of returned recommendations.

c) HoppingStrategy = Topological or Logical hopping

d) evaluationCriterion (String) = name of the evaluation criterion applied by IS.

e) MaxNumberOfHops (int) = upper bound on number of hops allowed

f) BranchingFactor = the number of neighbor agents to consider when at an agent point.

g) maxAttributes (int) = maximum number of recommended attributes within the queried agent.

h) maxAttributeValues (int) = maximum number of recommended values of each attribute.

i) AllocatedTime (int) = max permitted time before a return result.

Return:

j) A RecommendationList reply object.

5) Agent-Vote-Recommend

Parameters:

a) attributeName (String) = name of queried attribute.

b) maxItemsNum (int) = maximum number of returned recommendations.

c) evaluationCriterion (String) = name of the evaluation criterion applied by IS.

Return:

d) A RecommendationList reply object. This is the decision as obtained from a cooperative decision of several agents.

6.2.16.1 QRC™ Agent Format of APIs

All APIs are activated by sending a QRC™ agent of tag "INF_RECOMMEND_EVENT" to the IS. The sent QRC™ agent must contain the following elements:

1) MODEL – indicating the name of the queried agent;

2) SUBJECT – indicating what is being queried, which should either one of the following:

   a) The name of an attribute of the queried agent, indicating that this is a VarAgentRecommend.

   b) The string "Shallow-Agent-Recommend", indicating that this is a ShallowAgentRecommend.

   c) The string "Deep-Agent-Recommend", indicating that this is a DeepAgentRecommend.

3) MAX_ITEMS_NUM – indicating the maximum number of returned recommendations.

4) EVAL_CRIT - indicating the evaluation criterion used by the IS, which should be one of:

   a) The String "PROB_RECOMMEND_TYPE".

   b) The String "DELTA_RECOMMEND_TYPE".

When activating the Deep-Agent-Recommend API the QRC™ agent must also contain the following fields:

1) MAX_ATTRIBS – indicating the maximum number of recommended attributes within the queried agent.

2) MAX_ATTRIB_VALS – indicating the maximum number of recommended values in each recommended attribute.

Examples of QRC™ agents demonstrating the QRC™ agent structure of each API shown as follows:

1) **Var-Agent-Recommend**: Structure 2110 shown in Figure 21A.

2) **Shallow-Agent-Recommend**: Structure 2120 shown in Figure 21B.

3) **Deep-Agent-Recommend**: Structure 2130 shown in Figure 21C.

4) **Shallow-Neighbors-Agent-Recommend**: Structure 2140 shown in Figure 21D.

6.2.16.2 Result of Recommendations

The recommendations are returned in the form of a reply object of class **RecommendList** (see Figure 22), which represents a collection of recommendations, induced by IS 520. Each recommendation element is an object of class **Recommend** containing the following fields:

1) **Subject**: indicating the subject of recommendation (e.g., "A1.B1.C1", "A2.B2", etc.).

2) **SubjectType**: indicating the type of the recommended subject (e.g., "product", "B level category", etc.).

3) **Context**: the context in which this recommendation was done (e.g., the category in which this subject was recommended).

4) **LocalScore**: the relative score (e.g., probability) of this subject within its context.

5) **ContextScore**: the score of the context of the recommended subject.

The contents of the RecommendationsList object are translated to outer "users" (such as Business Objects) via a set of defined interface methods. An example 2200 is given in Figure 22, which presents the returned RecommendList information 2200 when activating a "Shallow-Neighbors-Agent-Recommend" API on agent A1 (see Figure 19), using the QRC™ agent structure 2150 of Figure 21E

The returned RecommendList 2200 of Figure 22 contains a variety of information. First, note that the total number of items returned in the table is not more than 10, as specified by <MAX_ITEMS_NUM VALUE = 10> in the structure of Figure 21E. Next, note that there are only two types of Contexts, namely, A1.BRD and A1.FRT. This adheres to the specification of <MAX_NEIGHBORS VALUE = 2 /> in Figure 21E. Also, for the first context, there are 2 items whose subjects are A1.BRD.BRDPIZ and A1.BRD.BRDCRM. The size being "2" adheres to the max value as specified by <MAX_NEIGHBOR_ATTRIBS VALUE = 3 />. This is also the case for the second context that has 3 items whose subjects are A1.FRT.PROJAR, A1.FRT.FROPA and A1.FRT.PROFGR.

6.2.17 IS Configuration Modes

Functional: The configuration modes control the following functionality:

1) Operational modes:

   a) Asynchronous/Synchronous propagation of evidence

   b) Asynchronous/Synchronous handling of QRC™ agents by the IM 1710. Asynchronous handling is achieved by a pool of Inference Handlers (IH) each operating in its dedicated thread.

   c) The default number of IEs dedicated to each IA. Each engine requires memory for its model. Engines are preferably not ORB objects, so it is possible for them to share the same model, thereby saving memory.

2) Compression modes:

a) Minimum length for compression

b) Minimum compression ratio

c) Minimum length of series to compress

3) Technical fields concerning location of input and log files

4) Printing modes that determine which information will be spooled to the inference log file.

The configuration is implemented as static Java variables in the AI Configuration object. This AI configuration is accessible to all objects that run in the current Java VM.

6.2.18 Scenarios of the Inference System

For illustrative purposes, several scenarios of IS 520 are discussed below.

6.2.18.1 Inference Manager Initiation

1) The initiator is the intelligence system 200 server 162.

2) A QRC™ agent "Start New Inference Manager" is sent to IM, and no reply is needed.

3) A new IM object is constructed.

4) IM sends CMR a QRC™ agent "Supply Model Net", and waits for reply.

5) CMR sends a reply object containing, a list of model objects & neighbors.

6) IM creates an internal agent graph.

7) IM creates the real agents (i.e., ORB objects).

6.2.18.2 Inference Manger Update

1) The initiator is CMR.

2) A QRC™ agent "Update a model" is sent to IM, no reply needed.

3) IM receives an IP reference to the new model.

4) IM sends the new model to the corresponding agent IA.

5) IA switches to its new model while previous session object still use the older model version.

6.2.18.3 Propagation of user driven evidence

1) The initiator is, for example, client 154, as result of user action .

2) A QRC™ agent "Product Order Event" is sent to IM, no reply needed.

3) IM finds the affected agent/s and propagates to them the "real" evidence.

4) The affected agent/s IA, receives the "Real" evidence and updates its corresponding session object IS.

5) IS performs the following:

a) Updates its evidence state (Real, Deterministic, Probabilistic evidence).

b) Update the timestamp of last arriving evidence, to the current time stamp. (time stamp is a local counter independent of the clock).

c) Have the calculations cache point to an empty list (Hash-Table).

d) Recalculate Gamma, and store it in the cache.

6) Get the Session's current Snapshot.

7) Propagate probabilistic evidence to each downstream of IA, as follows:

a) Calculate the distribution of its shared variable/s with IA, (The distribution is kept as a list of functions, to avoid functions of large magnitude).

b) Propagate the probabilistic evidence to the downstream agent, in an unsynchronized/synchronized manner.

8) The downstream agent IA receives the probabilistic evidence, and performs steps (4 to 7), excluding the updating of real evidence.

9) The propagation process terminates when all agents have propagated their evidence to all their downstream neighbors.

6.2.18.4 Query Request from Rule Evaluation System

1) The initiator is a AI rule evaluation system.

2) A QRC™ agent "Perform Inference Query" message is sent to IM, and a reply is sent after calculation.

3) IM 1710 allocates a free inference handler (e.g., IH 1710) and sends it the contents of the QRC™ agent, including the requester's addresses.

4) The IH, operating in a dedicated thread, interprets the query, and selects the initial agent with which to start the query.

5) IH performs a series of steps, wherein in each step the currently selected agent (e.g., IA 1730) is requested to perform a single or series of operations, such as:

a) Evidence absorption from neighbors.

b) Simple computations such as: Posterior Probability, Classification (e.g., MAP), and Most Probable Explanation (MPE).

c) More complex computations such as: "Highest Peek" and "Delta Classification".

6) Based on its computations at each step, the IH can shift its attention to a different IA,

requesting it to perform further calculations.

7) The query result is calculated by a target agent or agents that are reached through this series of steps.

8) Upon achieving the result, IH returns it in a reply object to the initiating business rule component, and becomes ready for further incoming requests.

6.2.18.5 Simple Computation (Prediction, MAP, MEU) by Inference Agent

1) The initiator is an IH, as part of performing the "Query Request" scenario.

2) The Inference Agent(e.g., IA 1730) receives the requested computation request.

3) IA attains a snapshot of the corresponding session.

4) If a result for the requested calculation is found in the snapshot's cache, then it is returned as the result of the computation.

5) The probabilistic/deterministic evidence corresponding to the queried variables is removed, and the snapshot's Gamma is recalculated.

6) Perform the actual computation using the Java Bayes Handler (JBH):

   a) Allocate a free inference engine (e.g., IE 1734).

   b) Create an inference graph, based on the engines Bayesian model.

   c) Set the deterministic evidence in the inference graph.

   d) Calculate Gamma (if it is null).

   e) Perform actual computation using the Bucket-Elimination algorithm.

7) Cache in the result of the calculation, only if :

   a) No evidence has arrived since the time in which the snapshot, upon which this calculation is based, has been taken.

   b) A more recent result for the same calculation has not been placed in the cache.

   8) Return the result to the initiating IH.

6.2.18.6 Evidence Absorption

1) The initiator is IH (it is possible that IA will do it implicitly before each query).

2) The IA receives a request to absorb all evidence from its non-downstream neighbors.

3) IA requests each of its non-downstream neighbors to supply it with probabilistic evidence on their mutually shared variables, by sending it an absorb request.

4) Each such neighbor then absorbs evidence from its non-downstream neighbors, calculates the requested probabilistic evidence and returns it to the requesting agent.

5) This recursive process terminates when the initial IA has completed absorption of

evidence from all of its neighbors.

6.2.18.7 Session Termination

1) The initiator is AI system 200, (external, client 154 driven).

2) A QRC™ agent "Session Terminated" is sent to the IM, and no reply is necessary.

3) IM broadcasts the session termination message to all its agents.

4) Each IA, removes the session object.

6.2.19 Examples Of Evidence Propagation

For illustrative purposes, several examples of evidence propagation are included below.

6.2.19.1 Overlapping Probabilistic Evidence.

Referring to the example of Figure 23, Model 1 2310 gets (and has) probabilistic evidence from Model 2 2320 and Model 3 2330. As a result, Model 1 has probabilistic evidence P*(A,B) and P*(B,C), with overlapping on variable B. Both functions will be used in the calculations so that Gamma = (P*(A,B) P*(B,C)) / (P(A,B,C).

6.2.19.2 Overriding Probabilistic evidence

An example of overriding evidence is shown in Figure 24, wherein Model 1 2410 shares variable set A with Model 2 2420 and Model 3 2430. Suppose that both Model 2 and Model 3 propagate probabilistic evidence on variable A. There are various ways to form this single P*(A). For instance, let P*(A) be the linear combination of both functions with weights equaling the model-dependent confidence, where confidence is a function of the coupling (i.e., the mutual information of the set A with respect to the no-evidence distribution) and the distance between the evidence-distribution of A and the no-evidence distribution of A (any of the standard distance functions may be used, e.g., the L1-distance).

In the current implementation, the algorithm lets P*(A) be the most recently propagated probabilistic evidence, namely, the most recently obtained evidence over A overrides any prior probabilistic evidence on A.

6.2.19.3 Propagating Probabilistic Evidence On Observed Variables.

An Example of mixed deterministic and probabilistic evidence is shown in Figure 25, wherein Model 1 2510 propagates probabilistic evidence P*(A,B) to Model 2 2520. Since Model 1 2510 received real evidence on A 2530, the probabilistic evidence sent to Model 2 2520 contains only a function P*(B). Notice that the evidence on A is sent to Model 2 in the form of real evidence by the IM of Model 1, thus there is no need for Agent A to send it to Model 2.

6.2.19.4 Preferring Probabilistic Evidence

Referring to Figure 26, probabilistic evidence that emanates from real evidence 2630 is preferred over propagated probabilistic evidence. That it, real evidence on two values of A is translated into probabilistic evidence on A. In the preferred embodiment, in Model 1 2610, this evidence overrides probabilistic evidence propagated from Model 2 2620.

The invention may be embodied in other specific forms without departing from the spirit or central characteristics thereof. The present embodiments are therefore to be considered in all respects as illustrative and not restrictive, the scope of the invention being indicated by appending claims rather than by the foregoing description, and all changes that come within the meaning and range of equivalency of the claims are therefore intended to be embraced therein.

Claims

1. An intelligent computer system for making real-time recommendations and predictions within the context of a domain, said system comprising:
   - A. at least one application server linked to a first storage media, having a plurality of Bayesian models stored therein, wherein each Bayesian model represents an entity within said domain and includes one or more entity attributes;
   - B. at least one client device disposed between said at least one application server and a device of a user;
   - C. an intelligence application module hosted on said application server and responsive to system events received from said client, said intelligent program including:
     - i. a set of domain related rules stored in a rules database;
     - ii. an artificial intelligence module configured to dynamically update said Bayesian models as a function of said system events and domain related information;

2. An intelligent computer system as in claim 1, wherein said Bayesian models are logically related in a Bayesian model network.

3. An intelligent computer system as in claim 2, wherein said Bayesian models logical relationships are statistical relationships.

4. An intelligent computer system as in claim 2, wherein said Bayesian models logical relationships are in the form of hyperlinks.

5. An intelligent computer system as in claim 1 wherein said Bayesian models are manually generated.

6. An intelligent computer system as in claim 1 wherein said Bayesian models are represented in XML format.

7. An intelligent computer system as in claim 1 wherein said domain includes a plurality of concepts and wherein a concept is represented over one or more of said Bayesian models.

8. An intelligent computer system as in claim 7 wherein a concept represented over a plurality of Bayesian models is represented at different levels of resolution in different ones of said Bayesian models.

9. An intelligent computer system as in claim 7 wherein said domain is an e-commerce domain and one or more of said concepts represents a product or a service.

10. An intelligent computer system as in claim 1 wherein said artificial intelligence module includes an inference module hosted on said application server and configured to generate predictions and recommendations as a function of:

i) attribute value predictions across all of said Bayesian models over time; and

ii) evidence derived from said domain related information,

wherein said inference module is configured to make said predictions and recommendations using partial, incomplete evidence.

11. An intelligent computer system as in claim 10 wherein said domain related information includes substantially static information including historical data related to a set of past users' sessions and statistical information related to a context of said domain.

12. An intelligent computer system as in claim 10 wherein said domain related information includes dynamically changing information.

13. An intelligent computer system as in claim 10 wherein said inference module is configured to passively and continuously conduct collection of said evidence across said Bayesian model network.

14. An intelligent computer system as in claim 10 wherein said inference module is configured to convert information related to system events into new evidence in real-time.

15. An intelligent computer system as in claim 14 wherein said new evidence is distributed and propagated to relevant ones of said Bayesian models via a Bayesian model network, wherein relevant Bayesian models are those Bayesian models having attributes to which said new evidence pertains.

16. An intelligent computer system as in claim 10 wherein said inference module includes:

A. one or more inference agents configured to gather said evidence and deliver said predictions and recommendations, wherein said inference agents are inter-linked according to logical and statistically topological based links by an inference agent network and further linked to said domain for receiving real-time, event-based evidence from said domain; and

B. a different set inference engines associated with each of said inference agents, wherein each of said inference engines is configured to generate and classify said attribute value predictions by generating probabilistic inference evidence related to said attribute values.

17. An intelligent computer system as in claim 1 wherein said artificial intelligence module includes a machine learning system comprising:

A. an on-line learning module hosted on said application server and configured to selectively and dynamically update and maintain said Bayesian models in real-time, as a function of dynamically changing domain related information.

18. An intelligent computer system as in claim 17, said machine learning system further includes:

B. an off-line learning module configured to dynamically generate and update said Bayesian models using auto-data mining and as a function of:

a) a unique definition of each entity included in said domain.

19. An intelligent computer system as in claim 18, wherein said off-line learning module is configured to further dynamically generate and update said Bayesian models as a

function of:

b) a set of definitions of the relationships between said entities.

20. An intelligent computer system as in claim 1, wherein said system events are related to interaction of said user with said intelligent computer system.

21. An intelligent computer system as in claim 1, wherein said system events are generated by a second application linked to said intelligent computer system.

22. An intelligent computer system as in claim 1, wherein said system events are generated by a second computer system linked to said intelligent computer system.

23. An intelligent computer system as in claim 1, wherein said intelligence application module includes a rules generation module, configured to facilitate the generation and storage of said domain related rules.

24. An intelligent e-commerce computer system for making real-time recommendations and predictions within the context of an e-commerce domain, said system comprising:

A. at least one application server linked to a first storage media, having a plurality of Bayesian models stored therein, wherein each Bayesian model represents at least one product or service entity within said e-commerce domain and includes one or more entity attributes;

B. at least one Web server disposed between said at least one application server and a device of said user;

C. an intelligence application module hosted on said application server and responsive to system events received from said client, said intelligent program including:

i. a rules generation module, configured to facilitate the generation of domain related rules and storage of said rules in a rules database;

ii. an artificial intelligence module configured to dynamically update said Bayesian models as a function of said system events, including an

inference module hosted on said application server and configured to generate predictions and recommendations using partial, incomplete evidence, as a function of:

a) attribute value predictions across all of said Bayesian models over time; and

b) evidence derived from domain related information; and

iii. a core module configured to evaluate said system events in accordance with said domain related rules and said Bayesian models and, as a function thereof, provide predictions and recommendations.

25. An intelligent e-commerce computer system as in claim 24 wherein said artificial intelligence module includes a machine learning system comprising:

a. an on-line learning module hosted on said application server and configured to selectively and dynamically update and maintain said Bayesian models in real-time, as a function of dynamically changing domain related information.

26. An intelligent e-commerce computer system as in claim 25, wherein said machine learning system further includes:

b. an off-line learning module configured to dynamically generate and update said Bayesian models using auto-data mining and as a function of:

a) a unique definition of each entity included in said domain.

27. An intelligent e-commerce computer system as in claim 26, wherein said off-line learning module is configured to further dynamically generate and update said Bayesian models as a function of:

b) a set of definitions of the relationships between said entities.

28. An intelligent e-commerce computer system as in claim 24, wherein said system events are related to interaction of said user with said intelligent e-commerce computer system

29. An inference system, as part of a computer system defining an application domain, wherein said intelligence computer system is accessible by a plurality of computers via a network, said inference system comprising:

A. at least one server linked to a storage media;

B. a plurality of Bayesian models, stored in said storage media and representing specific entities within a context of said domain; and

C. an inference module hosted on said server and configured to generate predictions and recommendations as a function of:

i) attribute value predictions across all of said Bayesian models over time,

ii) evidence derived from domain related information; and

wherein said inference module is configured to make said predictions and recommendations using partial, incomplete evidence and said Bayesian models are logically related to form a Bayesian model network.

30. An inference system as in claim 1 wherein said inference module includes an event-based asynchronous first channel by which said attribute value observations are provided.

31. An inference system as in claim 1 wherein said inference module includes a second channel by which said attribute value predictions are provided.

32. An inference system as in claim 1 wherein said domain includes a plurality of concepts and wherein a concept is represented over one or more of said Bayesian models.

33. An inference system as in claim 4 wherein a concept represented over a plurality of Bayesian models is represented at different levels of resolution in different ones of said Bayesian models.

34. An inference system as in claim 5 wherein said domain is an e-commerce domain and one or more of said concepts represents a product or a service.

35. An inference system as in claim 1 wherein said domain related information includes

dynamically changing information related to an on-line user's sessions.

36. An inference system as in claim 1 wherein said domain related information includes dynamically changing information related to said user's interaction with said computer system.

37. An inference system as in claim 1 wherein said domain related information includes dynamically changing information related to all on-line users' sessions.

38. An inference system as in claim 1 wherein said domain related information includes substantially static information.

39. An inference system as in claim 10 wherein said substantially static information includes historical data related to a set of past users' sessions and statistical information related to a context of said domain.

40. An inference system as in claim 10 wherein said substantially static information includes user related information, said user related information including indicia of one or more of the following, said user's:

A. financial information;
B. personal information;
C. history of activity with said computer system;
D. demographic information; and
E. affiliations.

41. An inference system as in claim 11 wherein said personal information includes at least one of the following:

i. age;
ii. gender;
iii. number of children; and
iv. interests.

42. An inference system as in claim 1 wherein said inference module is configured to passively and continuously conduct collection of said evidence across said Bayesian model network.

43. An inference system as in claim 1 wherein said inference module is configured to convert inference system event information into new evidence in real-time.

44. An inference system as in claim 15 wherein said conversion and an accumulation of said new evidence is accomplished using deterministic and probablistic methods implemented by said inference module.

45. An inference system as in claim 15 wherein said new evidence is distributed and propagated to relevant ones of said Bayesian models via said Bayesian model network, wherein relevant Bayesian models are those Bayesian models having attributes to which said new evidence pertains.

46. An inference system as in claim 15 wherein said new evidence overrides older evidence for corresponding attiributes in each of said relevant Bayesian models.

47. An inference system as in claim 1 wherein said inference module includes derived models configured to monitor system events and derive said evidence from system events detected.

48. An inference system as in claim 19 wherein said inference module includes derived models configured to monitor system events and derive said evidence from a recognition of system events not detected.

49. An inference system as in claim 19 wherein the events monitored by said derived models are related to a user's on-line activity and include one or more of:

A. a category change;

B. a choice of a concept; and

C. a time spent relative to a chosen concept.

50. An inference system as in claim 1 wherein said inference module further includes

recommending models configured to generate said predictions and recommendations.

51. An inference system as in claim 1 wherein said inference module includes:

A. one or more inference agents configured to gather said evidence and deliver said predictions and recommendations, wherein said inference agents are inter-linked according to logical and statistically topological based links by an inference agent network and further linked to said domain for receiving real-time, event-based evidence from said domain; and

B. a different set inference engines associated with each of said inference agents, wherein each of said inference engines is configured to generate and classify said attribute value predictions by generating probabilistic inference evidence related to said attribute values.

52. An inference system as in claim 23 wherein said attribute value predicitons for a targeted attribute are made using probability distributions.

53. An inference system as in claim 23 wherein said attribute value predictions for a targeted attribute are classified using the Maximum Aposteriori Probability algortihm.

54. An inference system as in claim 23 wherein said attribute value predictions for a targeted attribute are classified using one or more scoring techiniques.

55. An inference system as in claim 23 wherein said scoring techiniques determine one or more of the following:

A. a relative score;

B. a confidence score; or

C. an omega score.

56. An inference system as in claim 23 wherein said probablistic evidence is involuntarily dispersed from each inference agent to its peer inference agents, as said probabilistic evidence is generated.

57. An inference system as in claim 28 wherein said probabilistic evidence is cached.

58. An inference system as in claim 23 wherein each of said inference agents is configured to selectively request evidence from other inference agents having evidence related to said prediction request in response to reciept of said prediction request.

59. An inference system as in claim 23 wherein said inference engines are configured to concurrently generate said probabilistic inference evidence.

60. An inference system as in claim 31 wherein each of said inference engines is configured to generate said probabilistic inference evidence on a first subset of relevant attributes by implementing an extended Bucket Elimination process.

61. An inference system as in claim 32 wherein said Bucket Elimination process is extended to dynamically eliminate a second subset of said relelvant attributes, as a function of the affect of said second subset of relevant attributes on the determination of the probablistic inference evidence.

62. An inference system as in claim 32, wherein said second subset of relevant attributes are further eliminated as a function of a tradeoff between a prediction accuracy and a timeframe available for generating said probablistic inference evidence.

63. An inference system as in claim 23 wherein said inference module further includes an inference manager configured to selectively associate each of said inference agents with a specific one of said Bayesian models and further configured to selectively associate at least one inference engine with each inference agent.

64. An inference system as in claim 23 wherein said inference agents are logically linked according to a link topology, and wherein links between said inference agents in said topology represent a statistical relationships between said inference agents.

65. An inference system as in claim 36 wherein said inference module is configured to perform

agent hopping, wherein a sequence of statistically related inference agents having a common context are selectively queried for evidence corresponding to a prediction request to form a sinlge prediction based on the aggregation of evidence from said statistically related inference agents.

66. An inference system as in claim 37 wherein said agent hopping is dynamically controlled as a function of:

A. the amount of relevant evidence in topology related peer inference agents of a source inference agent;

B. an information value associated with the relevant evidence of each of said peer inference agents;

C. a determination that said source inference agent has not previsously hopped to said destination agent; and

wherein said agent hopping includes a voluntary transfer of evidence from said source inference agent to a destination peer inference agent and said topology may be traversed upward and downward.

67. An inference system as in claim 23 wherein each of a plurality of inference agents relates to a different context, but all of said plurality of agents include a common attribute, and said inference module is configured to generate said prediction as a function of a vote associated with said common attribute by each of said plurality of agents.

68. An inference system as in claim 39 wherein said inference module is configured to select said prediction by determining the most common prediction among said plurality of inference agents.

69. An inference system as in claim 39 wherein said inference module is configured to select said prediction by determining the weighted average of a set of predicted probability distributions over all of said plurality of inference agents and applying the Maximum Aposteriori Probability algortihm to said weighted average of the predicted probability distributions.

70. An inference system as in claim 39 wherein said inference module is configured to select said prediction by determining the prediction of the most confident inference agent of the plurality of inference agents.

71. A machine learning system comprising:

A. a processing device linked to a data storage media;

B. an interface to an application system, wherein said interface is configured to receive domain and entity related information from said application system;

C. a plurality of Bayesian models, stored in said storage media and representing specific entities within a context of said domain, wherein each Bayesian model includes one or more attributes; and

D. an on-line learning module hosted on said server and configured to selectively and dynamically update and maintain said Bayesian models in real-time, as a function of dynamically changing domain related information,

wherein said Bayesian models are used by said computer system to make user-related recommendations and predictions within a context of said domain.

72. A machine learning system as in claim 1, wherein one or more of said Bayesian models are generated manually.

73. A machine learning system as in claim 1, wherein said on-line learning module is further configured to update relationships between said Bayesian models.

74. A machine learning system as in claim 1, wherein said Bayesian models are stored in XML standard format.

75. A machine learning system as in claim 1, further comprising:

D. an off-line learning module configured to dynamically generate and update said Bayesian models using auto-data mining and as a function of:

a) a unique definition of each entity included in said domain.

76. A machine learning system as in claim 5, wherein said off-line learning module is

configured to further dynamically generate and update said Bayesian models as a function of:

b) a set of definitions of the relationships between said entities.

77. A machine learning system as in claim 5, wherein said off-line learning module is further configured to generate links between said Bayesian models as a function of a statistical relationships between said Bayesian models.

78. A machine learning system as in claim 7, wherein said links are hyperlinks.

79. A machine learning system as in claim 5, wherein each of said Bayesian models includes a group of attributes, and said attributes are grouped according to domain related criteria.

80. A machine learning system as in claim 5 wherein said off-line module is configured to generate said Bayesian models from a serial stream of data.

81. A machine learning system comprising:

A. a processing device linked to a data storage media;

B. an interface to an application system, wherein said interface is configured to receive domain and entity related information from said application system;

C. an off-line learning module executed on said processing device and configured to dynamically generate and update Bayesian models using auto-data mining as a function of:

a) a unique definition of each entity included in said domain; and

wherein said Bayesian models are stored in said storage media and represent specific entities within the context of said domain.

82. A machine learning system as in claim 11 wherein said off-line learning module is configured to further dynamically generate and update Bayesian models as a function of:

b) a set of definitions of the relationships between said entities.

83. A machine learning system as in claim 11, wherein said off-line learning module is further configured to generate links between said Bayesian models as a function of a statistical relationships between said Bayesian models.

84. A machine learning system as in claim 11, wherein each of said Bayesian models includes a group of attributes, and said attributes are grouped according to domain related criteria.

85. A machine learning system as in claim 11 wherein said off-line module is configured to generate said Bayesian models from a serial stream of data.

86. A machine learning system as in claim 11, further comprising:

D. an on-line learning module hosted on said server and configured to selectively and dynamically update and maintain said Bayesian models in real-time, as a function of dynamically changing domain related information.

87. A machine learning system as in claim 16, wherein said on-line learning module is further configured to update relationships between said Bayesian models.

AMENDED CLAIMS

[received by the International Bureau on 16 October 2000 (16.10.00);
original claims 1-87 replaced by new claims 1-158 (25 pages)]

1. An intelligent computer system comprising:
   A. an application system including an application domain and a plurality of Bayesian models stored therein, wherein each Bayesian model relates to an entity within said application domain and includes a set of attributes;
   B. at least one application device coupled to said application system;
   C. an intelligence system linked to said application system, said intelligence system including:
      1) an artificial intelligence module configured to selectively and dynamically update said Bayesian models as a function of dynamically changing domain related information; and
      2) a core module configured to provide substantially real-time predictions and recommendations related to said application domain as a function of said dynamically changing domain related information.

2. An intelligent computer system as in claim 1, wherein said Bayesian models are logically related in a Bayesian model network.

3. An intelligent computer system as in claim 2, wherein said Bayesian models logical relationships are statistical relationships.

4. An intelligent computer system as in claim 2, wherein said Bayesian models logical relationships are in the form of hyperlinks.

5. An intelligent computer system as in claim 1 wherein said Bayesian models are manually generated.

6. An intelligent computer system as in claim 1 wherein said Bayesian models are represented in XML format.

7. An intelligent computer system as in claim 1 wherein one or more of said entities is represented over a plurality of said Bayesian models.

8. An intelligent computer system as in claim 7 wherein an entity represented over a plurality of Bayesian models is represented at different resolution-based context levels.

9. An intelligent computer system as in claim 7 wherein said application domain is an e-commerce domain and one or more of said entities represents a product, a product category, a profile, or a service.

10. An intelligent computer system as in claim 1 wherein said artificial intelligence module includes an inference module configured to generate predictions and recommendations as a function of:

1) one or more predicted attribute values of one or more of said Bayesian models; and
2) evidence derived from said application domain.

11. An intelligent computer system as in claim 10 wherein said evidence derived from said application domain includes substantially static information.

12. An intelligent computer system as in claim 11 wherein said substantially static information includes one or more of the following:

A. financial information;
B. personal information;
C. history of activity with said intelligent computer system;
D. demographic information;
E. affiliations; and
F. externally provided user profile information.

13. An intelligent computer system as in claim 12 wherein said personal information includes one or more of the following:

1) age;
2) gender;
3) number of children;
4) interests or hobbies;
5) marital status;
6) financial income; and

7) previous purchases.

14. An intelligent computer system as in claim 10 wherein said Bayesian models are related to form a Bayesian model network and said inference module is configured to passively and continuously conduct collection of said evidence across said Bayesian model network.

15. An intelligent computer system as in claim 10 wherein said inference module is configured to convert information related to system events into new evidence in real-time.

16. An intelligent computer system as in claim 15 wherein said Bayesian models are related to form a Bayesian model network and one or more of said Bayesian models are configured to propagate new evidence to relevant other ones of said Bayesian models via said Bayesian model network, wherein relevant Bayesian models are those Bayesian models having attributes to which said new evidence pertains.

17. An intelligent computer system as in claim 10 wherein said inference module includes:

1) one or more inference agents configured to gather said derived evidence and deliver said predictions and recommendations, wherein said inference agents are inter-linked according to logical and statistically topological based relationships to form an inference agent network and further linked to said application domain for receiving real-time, event-based derived evidence from said application domain; and

2) a different set of inference engines associated with each of said inference agents, wherein each of said inference engines is configured to generate and classify said predicted attribute values by generating probabilistic inference evidence related to said attribute values.

18. An intelligent computer system as in claim 1 wherein said artificial intelligence module includes a machine learning system comprising:

A. an on-line learning subsystem configured to selectively and dynamically update and maintain said Bayesian models in real-time, as a function of said dynamically changing domain related information.

19. An intelligent computer system as in claim 18, wherein said machine learning system further includes:

B. an off-line learning subsystem, configured to build one or more of said Bayesian models.

20. An intelligent computer system as in claim 19, wherein said off-line learning subsystem includes an auto-data mining module configured to build and update one or more of said Bayesian models as a function of:

1) a unique definition of each entity included in said domain; and

2) a set of definitions of the relationships between said entities.

21. An intelligent computer system as in claim 1, wherein said dynamically changing domain related information is related to interaction of a user with said intelligent computer system.

22. An intelligent computer system as in claim 1, wherein said application device includes a second application and said dynamically changing domain related information is generated by said second application.

23. An intelligent computer system as in claim 1, wherein said application device is a second computer system and said dynamically changing domain related information is generated by said second computer system.

24. An intelligent computer system as in claim 1, wherein said intelligence system further includes:

3) a set of domain related rules, wherein said core module is further configured to provide said substantially real-time predictions and recommendations as a further function of said rules.

25. An intelligent computer system as in claim 24, wherein said intelligence system includes a rules generation module, configured to facilitate the generation and storage of said rules.

26. An intelligent computer system as in claim 1, wherein said application device is a wired or wireless Web-enabled device.

27. An intelligent e-commerce computer system comprising:

A. at least one application system including an e-commerce domain and a plurality of Bayesian models stored therein, wherein one or more of said Bayesian model represents at least one product, product category, profile or service entity within said e-commerce domain and includes one or more attributes;

B. at least one Web server coupled between said at least one application system and at least one user device;

C. an intelligence system coupled to said application system and configured to make predictions and recommendations related to said e-commerce domain as a function of a set of domain related rules and system events related to said e-commerce domain.

28. An intelligent e-commerce computer system as in claim 27 wherein said intelligence system includes:

1) an inference system module configured to generate said predictions and recommendations using partial, incomplete evidence, as a function of:

a) predicted values of attributes made across a plurality of said Bayesian models over time; and

b) evidence derived from said e-commerce domain.

29. An intelligent e-commerce computer system as in claim 24, wherein said intelligence system includes:

1) a machine learning system including:

a) an on-line learning subsystem configured to selectively and dynamically update said Bayesian models in real-time, as a function of dynamically changing domain related information; and

b) an off-line learning subsystem configured to build one or more of said Bayesian models as a function of at least one of:

i. a unique definition of each entity included in said e-commerce domain; and

ii. a set of definitions of the relationships between said entities.

30. An intelligent e-commerce computer system as in claim 27, wherein said intelligence system includes a plurality of computer systems, and wherein said intelligence system includes:

1) a load distribution module, configured to selectively distribute e-commerce domain related processes across said plurality of computer systems.

31. An intelligent e-commerce computer system as in claim 27, wherein said system events are related to interaction of said at least one user device with said e-commerce domain.

32. An intelligent e-commerce computer system as in claim 27, wherein said intelligence system includes:

1) a rules generation module, configured to facilitate the generation and maintenance of said domain related rules.

33. An intelligent e-commerce computer system as in claim 27, wherein each of said at least one user device is a wired or wireless device configured to access the e-commerce domain via the Web.

34. An inference system, interfaced with an application system defining an application domain having a plurality of entities represented therein, and accessible by a plurality of external devices via a network, said inference system comprising:

A. at least one processing device linked to a storage media;

B. a bidirectional interface module coupled to said application system;

C. a plurality of Bayesian models having a set of attributes, stored in said storage media and relating to entities within said application domain; and

D. an inference module hosted on said processing device and configured to generate predictions and recommendations related to said entities as a function of:

1) one or more predicted attribute values of one or more of said Bayesian models, and

2) evidence derived from said application domain.

35. An inference system as in claim 34 wherein said inference module includes an event-based asynchronous first channel, by which said evidence is provided for one or more

attributes of said Bayesian models.

36. An inference system as in claim 34 wherein said inference module includes a second channel, by which said predictions and recommendations are provided.

37. An inference system as in claim 34 wherein one or more of said entities is represented over a plurality of said Bayesian models.

38. An inference system as in claim 37 wherein an entity represented over a plurality of Bayesian models is represented at different resolution-based context levels.

39. An inference system as in claim 38 wherein said application domain is an e-commerce domain and one or more of said entities represents a product or a service.

40. An inference system as in claim 34 wherein said evidence derived from said application domain includes dynamically changing information related to one or more on-line users' operation of said external devices.

41. An inference system as in claim 34 wherein said inference module is configured to make said predictions using partial, incomplete evidence.

42. An inference system as in claim 34 wherein said Bayesian models are logically related to form a Bayesian model network.

43. An inference system as in claim 34 wherein said evidence derived from said application domain includes substantially static information.

44. An inference system as in claim 43 wherein said substantially static information includes historical data related to a set of past users' sessions and statistical information related to said application domain.

45. An inference system as in claim 43 wherein said substantially static information includes one or more of the following types of user related information:

A. financial information;

B. personal information;
C. history of activity with said computer system;
D. demographic information;
E. affiliations; and
F. externally provided user profile information.

46. An inference system as in claim 45 wherein said personal information includes at least one of the following:

1) age;
2) gender;
3) number of children;
4) interests or hobbies;
5) marital status;
6) financial income; and
7) previous purchases.

47. An inference system as in claim 34 wherein said Bayesian models are logically related to form a Bayesian model network and said inference module is configured to passively and continuously conduct collection of said evidence across said Bayesian model network.

48. An inference system as in claim 34 wherein said inference module is configured to convert inference system event information into new evidence in real-time.

49. An inference system as in claim 48 wherein said conversion and an accumulation of said new evidence is accomplished using deterministic and probabilistic methods implemented by said inference module.

50. An inference system as in claim 48 wherein said Bayesian models are logically related to form a Bayesian model network and one or more of said Bayesian models are configured to propogate said new evidence to relevant other ones of said Bayesian models via said Bayesian model network, wherein relevant Bayesian models are those Bayesian models having attributes to which said new evidence pertains.

51. An inference system as in claim 48 wherein said new evidence overrides older evidence

for corresponding attributes in relevant ones of said Bayesian models.

52. An inference system as in claim 34 wherein said inference module includes derived models configured to monitor system events and derive said evidence from system events detected.

53. An inference system as in claim 52 wherein said inference module includes derived models configured to monitor system events and derive said evidence from a recognition of system events not detected.

54. An inference system as in claim 52 wherein the events monitored by said derived models are related to a user's on-line activity via one of said external devices, and said events include one or more of:

A. a category change;
B. a choice of an entity; and
C. a time spent relative to a chosen entity;
D. a user's session start or end time;
E. a page view change;
F. adding or removing a product from a shopping cart; or
G. a content and a size of the shopping cart;
H. a product purchase.

55. An inference system as in claim 34 wherein said inference module is configured to generate said predictions and recommendations using at least one recommendation query technique chosen from a group of techniques including:

A. deep recommend;
B. var recommend;
C. shallow recommend; and
D. active evidence recommend.

56. An inference system as in claim 34 wherein said inference module includes:

1) one or more inference agents configured to gather said derived evidence and deliver said predictions and recommendations, wherein said inference agents are inter-linked according to logical and statistically topological

based relationships to form an inference agent network and further linked to said domain for receiving real-time, event-based derived evidence from said domain; and

2) a different set inference engines associated with each of said inference agents, wherein each of said inference engines is configured to generate and classify said predicted attributes value by generating probabilistic inference evidence related to said attribute values.

57. An inference system as in claim 56 wherein said attribute value predictions for a targeted attribute are made using probability distributions.

58. An inference system as in claim 56 wherein said attribute value predictions for a targeted attribute are classified using the Maximum Aposteriori Probability algorithm.

59. An inference system as in claim 56 wherein said attribute value predictions for a targeted attribute are classified using one or more scoring techniques.

60. An inference system as in claim 59 wherein said scoring techniques determine one or more of the following:

A. a relative score;

B. a confidence score; or

C. an omega score.

61. An inference system as in claim 56 wherein said probabilistic evidence is involuntarily dispersed from each inference agent to its peer inference agents, as said probabilistic evidence is generated.

62. An inference system as in claim 61 wherein said probabilistic evidence is cached.

63. An inference system as in claim 56 wherein each of said inference agents is configured to selectively request evidence from other inference agents having evidence related to said prediction request in response to receipt of said prediction request.

64. An inference system as in claim 56 wherein said inference engines are configured to

concurrently generate said probabilistic inference evidence.

65. An inference system as in claim 64 wherein each of said inference engines is configured to generate said probabilistic inference evidence on a first subset of relevant attributes by implementing an extended Bucket Elimination process, wherein said relevant attributes are those attributes useful in the generation of said probabilistic inference evidence.

66. An inference system as in claim 65 wherein said Bucket Elimination process is extended to dynamically eliminate a second subset of said relevant attributes, as a function of the affect of said second subset of relevant attributes on the determination of the probabilistic inference evidence.

67. An inference system as in claim 66, wherein said Bucket Elimination process is extended to determine said first subset of relevant attributes as a function of a tradeoff between a desired prediction accuracy and a timeframe available for generating said probabilistic inference evidence, wherein said second subset of relevant attributes includes those relevant attributes not included in said first set of relevant attributes.

68. An inference system as in claim 56 wherein said inference module further includes an inference manager configured to selectively associate each of said inference agents with a specific one of said Bayesian models and further configured to selectively associate at least one inference engine with each inference agent.

69. An inference system as in claim 56 wherein said inference agents are logically linked according to a link topology, and wherein links between said inference agents in said topology represent a statistical relationships between said inference agents.

70. An inference system as in claim 69 wherein said inference module is configured to perform agent hopping, wherein a sequence of statistically related inference agents having a common context are selectively queried for evidence corresponding to a prediction request to form a single prediction based on the aggregation of evidence from said statistically related inference agents.

71. An inference system as in claim 70 wherein said agent hopping is dynamically controlled

as a function of:

A. the amount of relevant evidence in topologically related peer inference agents of a source inference agent;

B. an information value associated with the relevant evidence of each of said peer inference agents; and

C. a determination that said source inference agent has not previously hopped to said destination agent,

wherein said agent hopping includes a voluntary transfer of evidence from said source inference agent to a destination peer inference agent and said topology may be traversed upward and downward.

72. An inference system as in claim 56 wherein each of a plurality of inference agents relates to a different context and share a common attribute, and said inference module is configured to generate said prediction as a function of a vote associated with said common attribute by each of said plurality of agents.

73. An inference system as in claim 72 wherein said inference module is configured to select said prediction by determining the most common prediction among said plurality of inference agents.

74. An inference system as in claim 72 wherein said inference module is configured to select said prediction by determining the weighted average of a set of predicted probability distributions over all of said plurality of inference agents and applying the Maximum Aposteriori Probability algorithm to said weighted average of the set of predicted probability distributions.

75. An inference system as in claim 72 wherein said inference module is configured to select said prediction by determining the prediction of the most confident inference agent of the plurality of inference agents.

76. An inference system as in claim 56 wherein said inference module is configured to generate said predictions and recommendations using at least one recommendation query technique, chosen from a group of techniques including:

A. deep recommend;

B. var recommend;
C. shallow recommend;
D. active evidence recommend
E. agent hop; and
F. agent vote.

77. A method of providing predictions and recommendations, related to entities represented within an application domain of an application system that is accessible by a plurality of external devices via a network, by an inference system coupled to said application system and including a processing device and storage media having a plurality of Bayesian models related to said entities stored therein, said method comprising:
A. accessing relevant Bayesian models from said storage media;
B. predicting attribute values for said Bayesian models;
C. deriving evidence from said application domain; and
D. generating a prediction or recommendation as a function of said Bayesian models and said derived evidence.

78. The method of claim 77 wherein one or more of said entities is represented over a plurality of said Bayesian models.

79. The method of claim 77 wherein said application domain is an e-commerce domain and each of said entities represents a product or a service.

80. The method of claim 77 wherein step C includes monitoring system events and deriving evidence from events observed.

81. The method of claim 80 wherein said events monitored are related to a user's on-line activity via one of said external devices, and said events include one or more of:
1) a category change;
2) a choice of a context;
3) a time spent relative to a chosen entity,
4) a user's session start or end time;
5) a page view change;
6) adding or removing a product from a shopping cart;

7) a content and a size of the shopping cart; and

8) a product purchase.

82. The method of claim 77 wherein step C includes monitoring system events and deriving evidence from a recognition of events not observed.

83. The method of claim 77 wherein said evidence derived from said application domain in step C includes substantially static information.

84. The method of claim 83 wherein said substantially static information includes historical data related to a plurality of past users' interaction with said application system.

85. The method of claim 83 wherein said substantially static information includes information related to a current user.

86. The method of claim 77 wherein steps B and C are carried out passively and continuously.

87. The method of claim 77 wherein step B includes the step of:

1) generating probablistic inference evidence on a first subset of relevant attributes by implementing an extended Bucket Elimination process,

wherein said relevant attributes are those attributes useful in the generation of said probabilistic inference evidence.

88. The method of claim 87 wherein step B further includes, as an extension to said Bucket Elimination process:

2) eliminating a second subset of attributes as a function of the affect of said second set of attributes on the generation of the probablistic inference evidence on said first set of attributes.

89. The method of claim 88 wherein said extension to said Bucket Elimination process includes performing a tradeoff analysis between a desired prediction accuracy and a timeframe to generate said probabilistic inference to determine said first set of relevant attributes necessary to achieve said prediction accuracy in said timeframe, wherein said

second subset of relevant attributes includes those relevant attributes not included in said first set of relevant attributes.

90. The method of claim 77, wherein said inference system includes a plurality of inference agents under the control of one or more inference managers, and steps A through D are carried out by said inference agents, wherein at least one inference agent is associated with each of said Bayesian models.

91. The method of claim 90 wherein said inference agents facilitate the sharing of said evidence among said Bayesian models.

92. The method of claim 77 wherein said in Step D includes generating said predictions and recommendations using at least one recommendation query techniques from a group of techniques including:

1) deep recommend;
2) var recommend;
3) shallow recommend;
4) active evidence recommend
5) agent hop; and
6) agent vote.

93. A machine learning system comprising:

A. at least one processing device linked to a data storage media;
B. an interface to an application system defining an application domain, wherein said interface is configured to receive dynamically changing domain related information from said application system;
C. a plurality of Bayesian models having a set of attributes, stored in said storage media and relating to entities within said application domain; and
D. an on-line learning subsystem hosted on said at least one processing device and configured to selectively and dynamically update said Bayesian models in real-time, as a function of said dynamically changing domain related information.

94. A machine learning system as in claim 93, wherein said on-line learning system is further configured to update said Bayesian models as a function of substantially static

information, including indicia of one or more of the following:

A. historical application domain related data;

B. financial information;

C. personal information;

D. user history of activity with said application system;

E. demographic information;

F. affiliations; and

G. externally provided user profile information.

95. A machine learning system as in claim 93, wherein said at least one processing device includes a plurality of processing devices, and wherein said machine learning system further includes:

E. a load distribution module, configured to selectively distribute processes related to said updating of said Bayesian models across said plurality of processing devices as a function of a processing load of each of said processing devices.

96. A machine learning system as in claim 93, wherein said Bayesian models are stored in XML standard format.

97. A machine learning system as in claim 93, wherein said Bayesian models are part of a Bayesian model network and linked by inter-model links, and said on-line learning subsystem includes an on-line learning module configured to create, update and delete said inter-model links.

98. A machine learning system as in claim 97, wherein each of said inter-model links corresponds to a statistical relationship between two linked Bayesian models and said on-line learning module is configured to update said inter-model links as a function of said dynamically changing domain related information.

99. A machine learning system as in claim 97 wherein said Bayesian model network represents a joint probability distribution across all attributes of said plurality of Bayesian models.

100. A machine learning system as in claim 93, wherein one or more of said Bayesian models

includes a model structure defined by said attributes and a set of intra-model links, wherein for a given Bayesian model a set of intra-model links establishes relationships between two or more attributes of said given Bayesian model.

101. A machine learning system as in claim 100, wherein said on-line learning subsystem is configured to create, update and delete said intra-model links.

102. A machine learning system as in claim 100 wherein said intra-model links represent joint probabilities of at least two attributes, and wherein said updating includes updating said joint probabilities.

103. A machine learning system as in claim 93 wherein said on-line learning subsystem includes a plurality of learning agents and each of said learning agents is associated with and configured to update a corresponding one or more of said Bayesian models.

104. A machine learning system as in claim 93, wherein said plurality of Bayesian models are part of a Bayesian model network and said on-line learning subsystem includes a plurality of learning agents and each of said learning agents is configured to transfer information between two or more of said plurality of Bayesian models via the Bayesian model network.

105. A machine learning system as in claim 93 further comprising:

E. an interface to an inference system, wherein said inference system is configured to generate recommendations and predictions related to said Bayesian models.

106. A machine learning system as in claim 105 wherein said on-line learning subsystem is configured to query said inference system to obtain said recommendations and predictions.

107. A machine learning system as in claim 105 wherein one or more of said Bayesian models is updated in response to information received from said inference system.

108. A machine learning system as in claim 93 wherein said on-line learning subsystem is configured to update said Bayesian models using partial and incomplete sets of attribute

values, wherein one or more of said Bayesian models includes a subset of attributes having undefined attribute values.

109. A machine learning system as in claim 93 wherein said on-line subsystem includes one or more of:

1) a polling module, configured to poll said application domain for said dynamically changing domain related information; and

2) a notification receiver module, configured to receive notifications from said application domain related to said dynamically changing domain related information.

110. A machine learning system as in claim 93, further comprising:

E. an off-line subsystem including:

1) an off-line building module, configured to generate one or more of said Bayesian models, wherein each of said Bayesian models includes a set of attributes; and

2) an off-line learning module, configured to learn said one or more Bayesian models, wherein each learned Bayesian model includes determined attribute values for said set of attributes.

111. A machine learning system as in claim 110, wherein said off-line building module includes an auto-data mining module configured to build said one or more Bayesian models as a function of one or more of:

1) a unique definition of each of said entities;

2) a set of data cases, each of said data cases including information relating a group of said entities; and

3) a set of definitions of the relationships between said entities.

112. A machine learning system as in claim 93, wherein said dynamically changing domain related information is related to an event generated by an application hosted on or accessible by said application system.

113. A machine learning system as in claim 93, wherein said dynamically changing domain related information is related to a user's interaction with said application system via a

wired or wireless device linked to said application system.

114. A machine learning system as in claim 93, wherein said application system is a Web-based application system and said dynamically changing domain related information is a function of a user's interaction with a said Web-based application system.

115. A machine learning system as in claim 93, wherein said application domain is an e-commerce domain and one or more of said entities represents a product, service, or product category.

116. A machine learning system comprising:

A. at least one processing device linked to a data storage media; and

B. an off-line learning subsystem hosted on said at least one processing device and including:

1) an off-line building module, configured to generate and store in said storage media one or more Bayesian models, wherein each of said Bayesian models is configured to represent at least one entity related to an application domain of an application system and each of said Bayesian models includes a set of attributes.

117. A machine learning system as in claim 116, wherein said off-line building module further includes:

2) an off-line learning module, configured to learn said one or more Bayesian models, wherein each learned Bayesian model includes determined attribute values for said set of attributes.

118. A machine learning system as in claim 116, wherein said off-line building module includes an auto-data mining module configured to build said one or more Bayesian models as a function of one or more of:

1) a unique definition of each of said entities; and

2) a set of data cases, each of said data cases including information relating a group of said entities.

119. A machine learning system as in claim 118, wherein said off-line building module is

configured to further generate Bayesian models as a function of:

3) a set of definitions of the relationships between said entities.

120. A machine learning system as in claim 116, wherein said Bayesian models are structured to be used to make recommendations and predictions within said application domain.

121. A machine learning system as in claim 116, wherein said off-line learning subsystem is further configured to generate a Bayesian model network comprising said Bayesian models and a plurality of corresponding inter-model links.

122. A machine learning system as in claim 121, wherein each inter-model link relates to a pair of Bayesian models and is generated as a function of one or more statistical relationships between entities represented by said pair of Bayesian models.

123. A machine learning system as in claim 121 wherein said Bayesian model network represents a joint probability distribution across all attributes of said plurality of Bayesian models.

124. A machine learning system as in claim 116, wherein said attributes are grouped according to one or more application domain related criterion.

125. A machine learning system as in claim 116, wherein one or more of said Bayesian models includes a model structure defined by said attributes and intra-model links, wherein for a given Bayesian model a set of intra-model links establish relationships between two or more attributes of said given Bayesian model.

126. A machine learning system as in claim 125 wherein said intra-model links represent joint probabilities of at least two attributes.

127. A machine learning system as in claim 116, wherein said off-line subsystem further comprises:

2) a manual Bayesian model generator, configured to generate and store one or more of said Bayesian models in response to a user's instructions.

128. A machine learning system as in claim 116, further comprising:

C. an interface to said application system, wherein said interface is configured to receive dynamically changing domain related information from said application system; and

D. an on-line learning subsystem hosted on said at least one processing device and configured to selectively and dynamically update said Bayesian models in real-time, as a function of said dynamically changing domain related information.

129. A machine learning system as in claim 128, wherein said on-line learning subsystem is configured to update relationships between said Bayesian models.

130. A machine learning system as in claim 128, wherein said plurality of Bayesian models are part of a Bayesian model network and said on-line learning subsystem is configured to transfer information between two or more of said plurality of Bayesian models via the Bayesian model network.

131. A machine learning system as in claim 116, further configured to generate, learn, and update said Bayesian models as a function of substantially static information, including indicia of one or more of the following:

A. historical application domain related data;

B. financial information;

C. personal information;

D. user history of activity with said application system;

E. demographic information;

F. affiliations; and

G. externally provided user profile information.

132. A machine learning system as in claim 116, wherein said application domain is an e-commerce domain and one or more of said entities represents a product, service, or product category.

133. A machine learning system comprising:

A. at least one processing device linked to a data storage media;

B. an interface to an application system defining an application domain, wherein said

interface is configured to receive dynamically changing domain related information from said application system;

C. a plurality of Bayesian models having a set of attributes, stored in said storage media and relating to entities within said application domain;

D. an off-line subsystem configured to generate and learn one or more of said Bayesian models; and

E. an on-line learning subsystem hosted on said at least one processing device and configured to selectively and dynamically update said Bayesian models in real-time, as a function of said dynamically changing domain related information.

134. A machine learning system as in claim 133, further comprising:

F. an interface to an inference system, wherein said inference system is configured to generate recommendations and predictions related to said Bayesian models.

135. A machine learning system as in claim 133, wherein said application domain is an e-commerce domain, said application system is accessible by consumers via the Web, and one or more of said entities represents a product, service, or product category.

136. A machine learning system as in claim 133, wherein said dynamically changing domain related information is related to a user's interaction with said application system via a wired or wireless device linked to said application system.

137. A machine learning system as in claim 133, wherein each entity represents a product or service available via said application system and said dynamically changing domain related information includes information related to one or more consumers' interaction with said application system.

138. A machine learning system as in claim 133, further configured to generate, learn, and update said Bayesian models as a function of substantially static information, including indicia of one or more of the following:

A. historical application domain related data;

B. financial information;

C. personal information;

D. user history of activity with said application system;

E. demographic information:

F. affiliations; and

G. externally provided user profile information.

139. A machine learning system as in claim 133, wherein one or more of said Bayesian models includes an model structure defined by said attributes and intra-model links, wherein for a given Bayesian model a set of intra-model links establishes relationships between two or more attributes of said given Bayesian model.

140. A machine learning system as in claim 138, wherein said on-line learning subsystem is configured to create, update and delete said intra-model links.

141. A machine learning system as in claim 139 wherein said intra-model links represent joint probabilities of at least two attributes, and wherein said updating includes updating said joint probabilities.

142. A machine learning system as in claim 133, wherein said plurality of Bayesian models are linked by inter-model links as part of a Bayesian model network, and said on-line learning subsystem is configured to transfer information between two or more of said plurality of Bayesian models via the Bayesian model network.

143. A method of representing and maintaining entities in an application domain of an application system, wherein said application system is linked to a learning system having a processing device and a storage media, said method comprising:

A. generating a plurality of Bayesian models representing said entities, wherein each of said Bayesian models includes a set of attributes corresponding to at least one of said entities;

B. producing a Bayesian model network comprised of said Bayesian models and inter-model links, wherein one or more of said inter-model links are generated between a pair of said Bayesian models as a function of one or more statistical relationships between entities represented by said pair of Bayesian models; and

C. updating said Bayesian models and said Bayesian model network as a function of dynamically changing domain related information.

144. The method of claim 143, wherein said Bayesian models are further updated as a function of substantially static information.

145. The method of claim 144 wherein said substantially static information includes indicia of one or more of the following:

A. historical application domain related data;
B. financial information;
C. personal information;
D. user history of activity with said application system;
E. demographic information;
F. affiliations; and
G. externally provided user profile information.

146. The method of claim 143 further comprising:

D. interfacing with an inference system configured to generate predictions and recommendations related to said Bayesian models.

147. The method of claim 143 wherein, for at least one of said entities, step A is accomplished using a partial and incomplete set of attribute values, wherein a subset of attribute values are undefined.

148. The method of claim 147 wherein step A includes:

1) obtaining recommendations and estimations for said undefined attribute values from an inference system configured to generate predictions and recommendations.

149. The method of claim 143 wherein steps A and B are accomplished using auto-data mining and as a function of:

1) a unique definition of each entity included in said domain; and
2) a set of data cases, each of said data cases comprising a group of entities in said domain.

150. The method of claim 149 wherein steps A and B are accomplished as a further function of:

3) a set of definitions of the relationships between said entities.

151. The method of claim 143 wherein said updating in step C is carried out by learning agents, wherein each learning agent is associated with at least one of said Bayesian models.

152. The method of claim 143, wherein Step A includes defining a model structure by establishing a set of intra-model links between said attributes, wherein for a given Bayesian model a set of intra-model links establishes relationships between two or more attributes of said given Bayesian model.

153. The method of claim 151, wherein Step D includes creating, updating and deleting said intra-model links.

154. The method of claim 152, wherein said intra-model links represent joint probabilities of at least two attributes, and wherein said updating includes updating said joint probabilities.

155. The method of claim 143, wherein at least one of steps A, B, and C includes transferring information between two or more of said plurality of Bayesian models via the Bayesian model network.

156. The method of claim 143 wherein said context is an e-commerce context and each of said entities represents a product, service, or product category.

157. The method of claim 143 further comprising:

D. accessing said application domain via the Web.

158. The method of claim 143 further comprising:

D. accessing said application domain via a wired or wireless device.

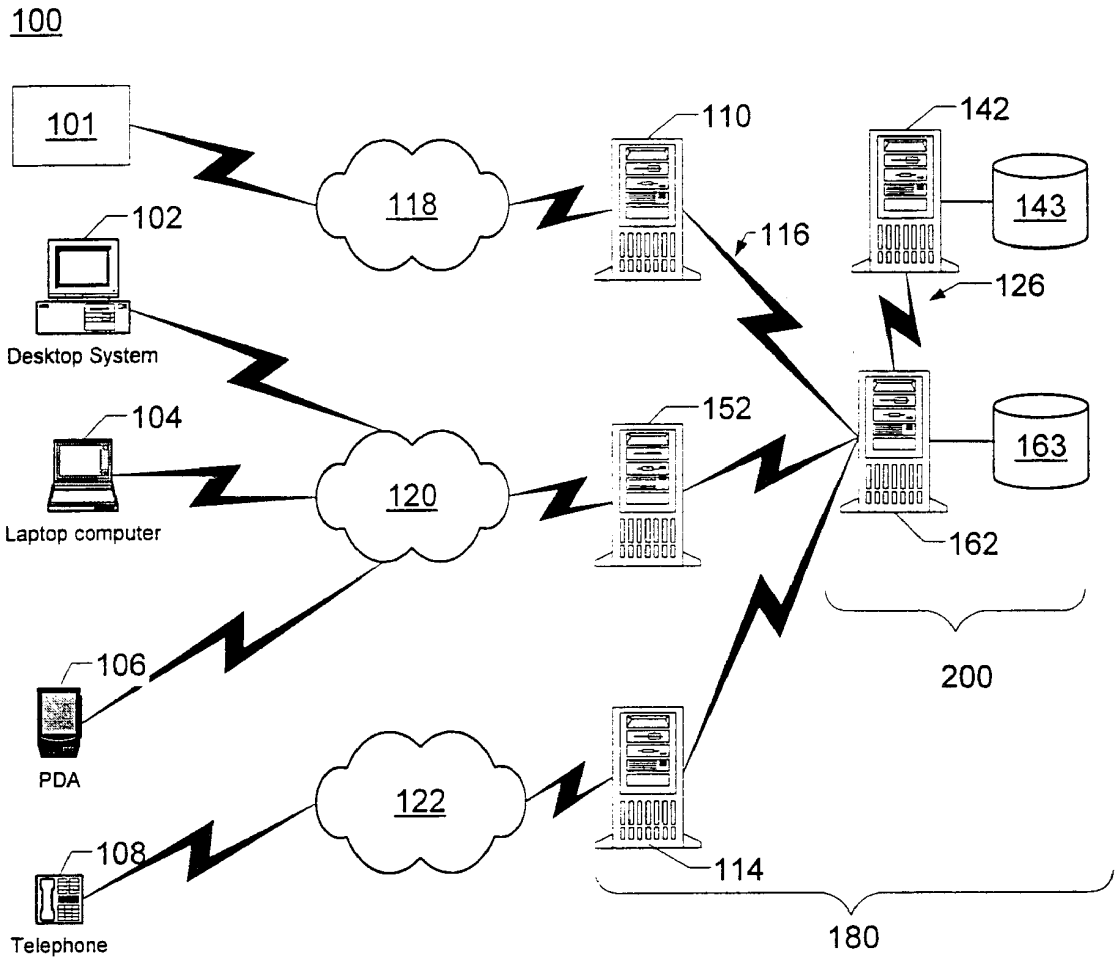

100
101
102
Desktop System
104
Laptop computer
106
PDA
108
Telephone
118
120
122
110
116
152
114
142
143
126
163
162
200
180

FIG. 1

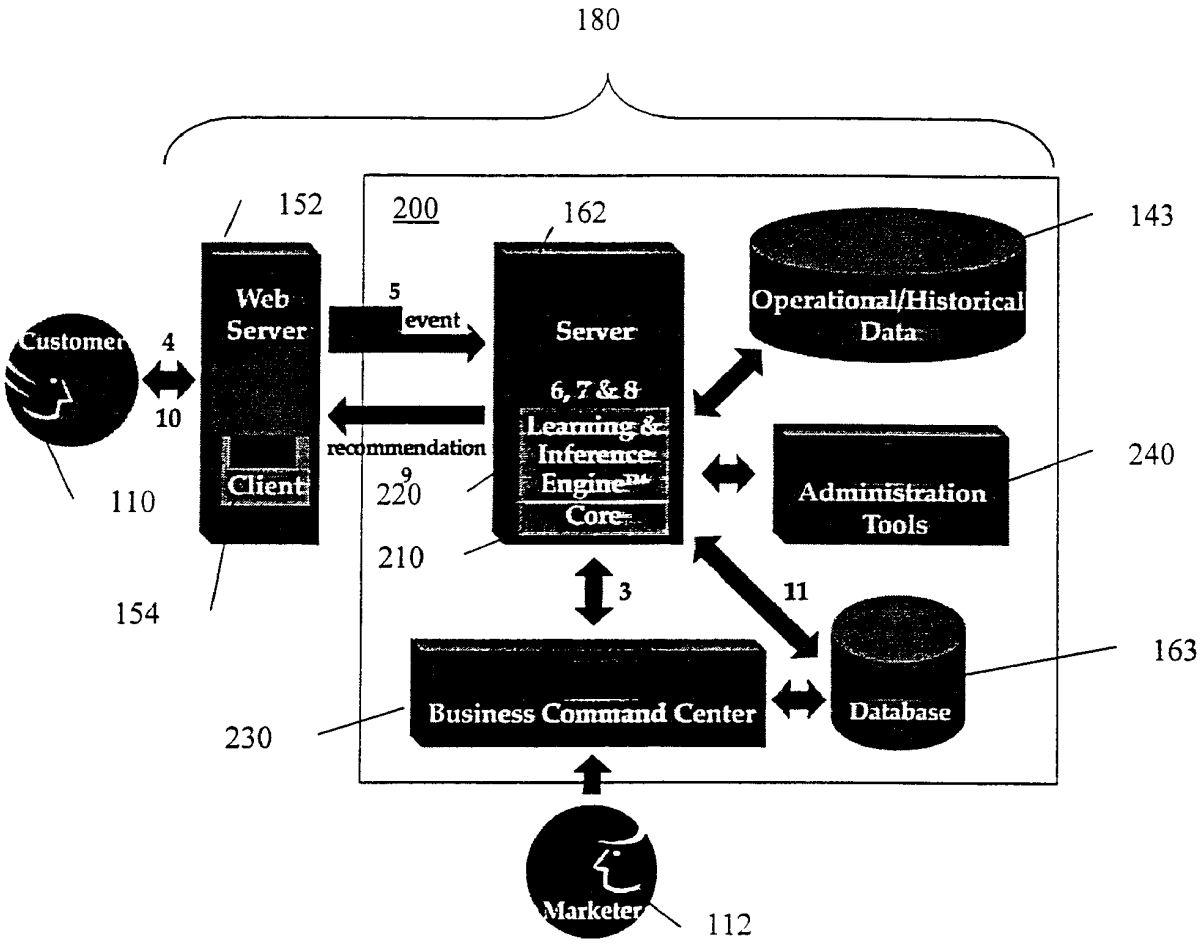

180
152
200
162
143
Web Server
5
event
Server
Operational/Historical Data
Customer
4
10
6, 7 & 8
Learning & Inference Engine™
Core
recommendation
9
220
Administration Tools
240
110
210
154
3
11
163
Business Command Center
Database
230
Marketer
112

FIG. 2

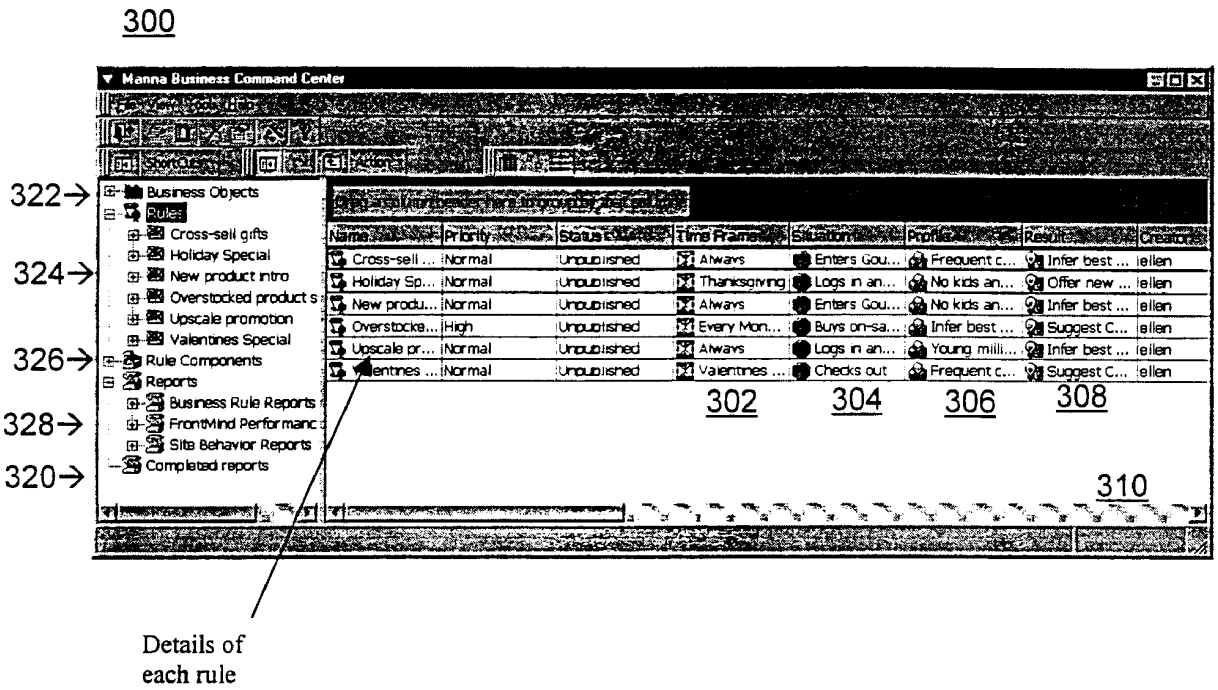

300
Manna Business Command Center
322
324
326
328
320
Business Objects
Rules
Cross-sell gifts
Holiday Special
New product intro
Overstocked product s
Upscale promotion
Valentines Special
Rule Components
Reports
Business Rule Reports
FrontMind Performanc
Site Behavior Reports
Completed reports
Name
Priority
Status
Time Frame
Situation
Profile
Result
Creator
Cross-sell ... Normal Unpublished Always Enters Gou... Frequent c... Infer best ... ellen
Holiday Sp... Normal Unpublished Thanksgiving Logs in an... No kids an... Offer new ... ellen
New produ... Normal Unpublished Always Enters Gou... No kids an... Infer best ... ellen
Overstocke... High Unpublished Every Mon... Buys on-sa... Infer best ... Suggest C... ellen
Upscale pr... Normal Unpublished Always Logs in an... Young milli... Infer best ... ellen
Valentines ... Normal Unpublished Valentines ... Checks out Frequent c... Suggest C... ellen
302
304
306
308
310
Details of
each rule

FIG. 3

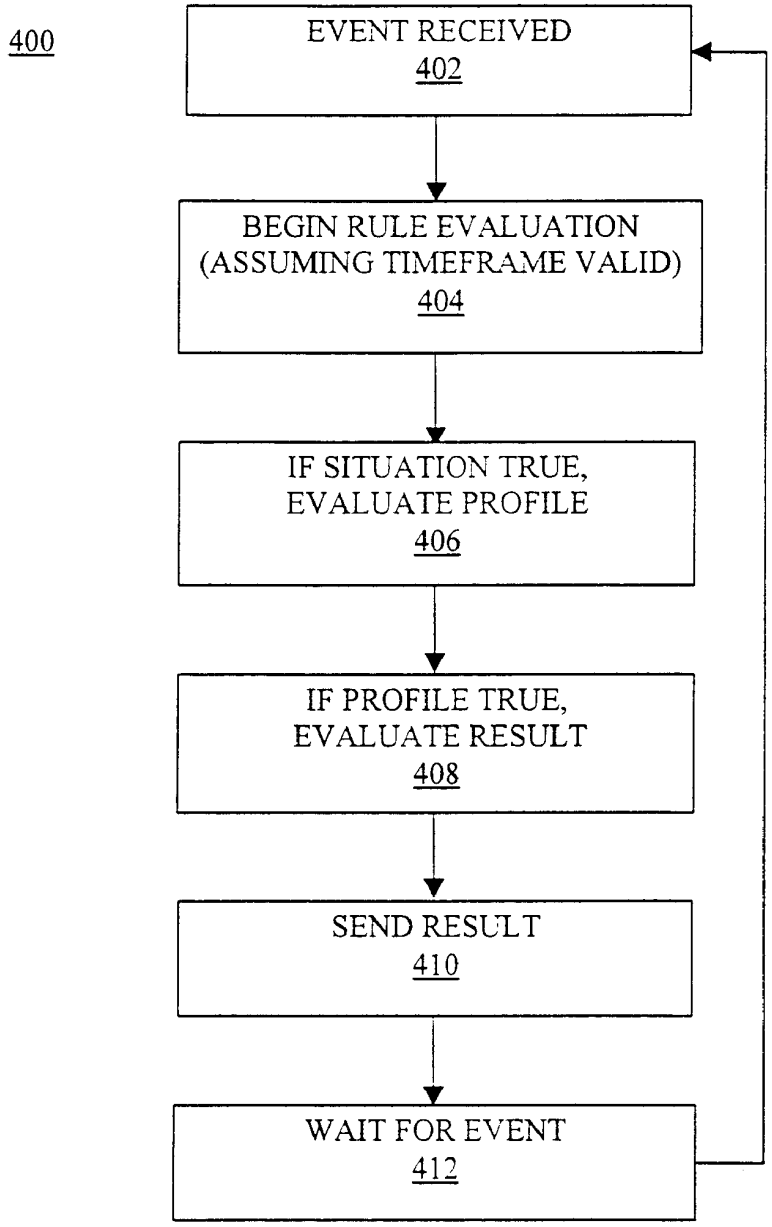

400
EVENT RECEIVED
402
BEGIN RULE EVALUATION
(ASSUMING TIMEFRAME VALID)
404
IF SITUATION TRUE,
EVALUATE PROFILE
406
IF PROFILE TRUE,
EVALUATE RESULT
408
SEND RESULT
410
WAIT FOR EVENT
412

FIG. 4

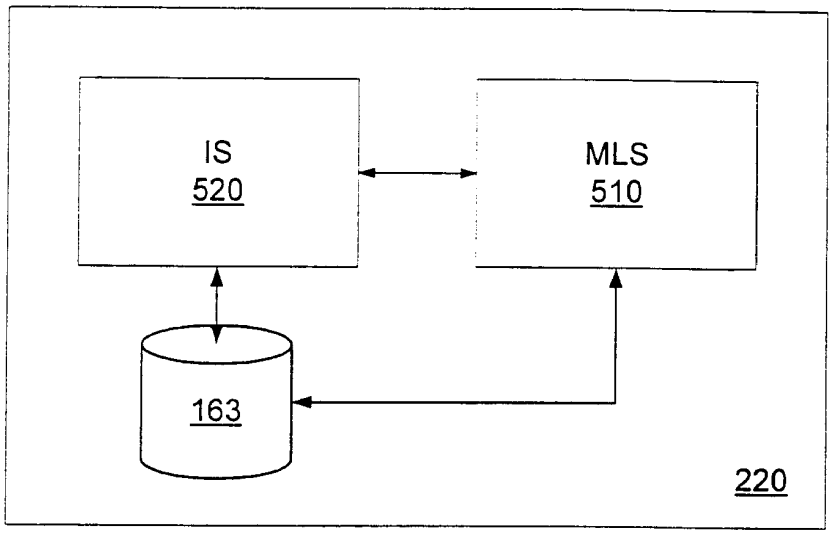


FIG. 5

. . 600 .
'A1','BAK','BAKBAG','100336','Fresh','Bakery','Fresh Baked Bagels','Back Bay Bagels - Garlic'
'A1','BAK','BAKBAG','100337','Fresh','Bakery','Fresh Baked Bagels','Back Bay Bagels - Honey Grain'
'A1','BAK','BAKBAG','100338','Fresh','Bakery','Fresh Baked Bagels','Back Bay Bagels - Salt'
'A1','BAK','BAKBAG','100339','Fresh','Bakery','Fresh Baked Bagels','Back Bay Bagels - Rye'
'A1','BAK','BAKBAG','100340','Fresh','Bakery','Fresh Baked Bagels','Back Bay Bagels - Marble'
'A1','BAK','BAKBAG','100341','Fresh','Bakery','Fresh Baked Bagels','Back Bay Bagels - Poppy'
'A1','BAK','BAKBAG','100342','Fresh','Bakery','Fresh Baked Bagels','Back Bay Bagels - Pumpernickel'
'A1','BAK','BAKBAG','100343','Fresh','Bakery','Fresh Baked Bagels','New York Bagel Chips Garlic'
'A1','BAK','BAKBAG','100344','Fresh','Bakery','Fresh Baked Bagels','New York Bagel Chips Plain Low Salt'
'A1','BAK','BAKBAG','100345','Fresh','Bakery','Fresh Baked Bagels','New York Bagel Chips Cinnamon Raisin'
'A1','BAK','BAKBAG','100346','Fresh','Bakery','Fresh Baked Bagels','New York Bagel Chips Garlic 98% Fat Free'
'A1','BAK','BAKBAG','100347','Fresh','Bakery','Fresh Baked Bagels','Back Bay Bagels - Everything'
'A1','BAK','BAKBAG','100348','Fresh','Bakery','Fresh Baked Bagels','Back Bay Bagels - Blueberry'
'A1','BAK','BAKBAG','100349','Fresh','Bakery','Fresh Baked Bagels','Boston Bagel Plain 20 oz'
'A1','BAK','BAKBAG','100350','Fresh','Bakery','Fresh Baked Bagels','Boston Bagel Onion 20 oz'
'A1','BAK','BAKBAG','100351','Fresh','Bakery','Fresh Baked Bagels','Boston Bagel Sesame 20 oz'
'A1','BAK','BAKBAG','100352','Fresh','Bakery','Fresh Baked Bagels','Boston Bagel Honey Wheat 20 oz'
'A1','BAK','BAKBAG','100353','Fresh','Bakery','Fresh Baked Bagels','Boston Bagel Marble 20 oz'
'A1','BAK','BAKBAG','100354','Fresh','Bakery','Fresh Baked Bagels','Boston Bagel Cinnamon Raisin 20 oz'
'A1','BAK','BAKBAG','100355','Fresh','Bakery','Fresh Baked Bagels','New York Pita Chips Plain'
'A1','BAK','BAKBAG','100356','Fresh','Bakery','Fresh Baked Bagels','New York Pita Chips Garlic'
'A1','BAK','BAKBAG','100357','Fresh','Bakery','Fresh Baked Bagels','Back Bay Bagels - Sourdough'
'A1','BAK','BAKBRD','100366','Fresh','Bakery','Fresh Baked Breads','Fresh Bread - Scala'
'A1','BAK','BAKBRD','100367','Fresh','Bakery','Fresh Baked Breads','Fresh Bread - Italian Sliced'
'A1','BAK','BAKBRD','100368','Fresh','Bakery','Fresh Baked Breads','Fresh Bread - Italian Unsliced'
'A1','BAK','BAKBRD','100369','Fresh','Bakery','Fresh Baked Breads','Fresh Bread - Pumpkin Raisin Sliced'
'A1','BAK','BAKBRD','100370','Fresh','Bakery','Fresh Baked Breads','Fresh Bread - French Baguette'
'A1','BAK','BAKBRD','100371','Fresh','Bakery','Fresh Baked Breads','Fresh Bread - Lucerne Multi Grain Sliced'
'A1','BAK','BAKBRD','100372','Fresh','Bakery','Fresh Baked Breads','Fresh Bread - Challah'
'A1','BAK','BAKBRD','100373','Fresh','Bakery','Fresh Baked Breads','Fresh Bread - Rye Crusty'
'A1','BAK','BAKBRD','100374','Fresh','Bakery','Fresh Baked Breads','Fresh Bread - Batard'
'A1','BAK','BAKBRD','100375','Fresh','Bakery','Fresh Baked Breads','Fresh Bread - Sour Dough'
'A1','BAK','BAKBRD','100376','Fresh','Bakery','Fresh Baked Breads','Fresh Bread - Mountain White'
. . .

Fig. 6A

650

'A1','FRT','PROFPN','109727','1'
'A1','FRT','PROFML','109776','1'
'A1','BAK','BAKCAK','100393','12'
'B1','BAF','BABFDG','100224','15'
'B1','BAF','BABFDG','100225','15'
'B1','BAF','BABFDG','100258','15'
. . .
'A1','BRD','BRDBAK','101344','16'
'A1','BRD','BRDBAK','101359','16'
'A1','BRD','BRDBUN','101365','16'
'A1','BRD','BRDLOV','101446','16'
'A1','BRD','BRDLOV','101472','16'
'B1','CER','BRKCOL','101654','16'

Fig. 6B

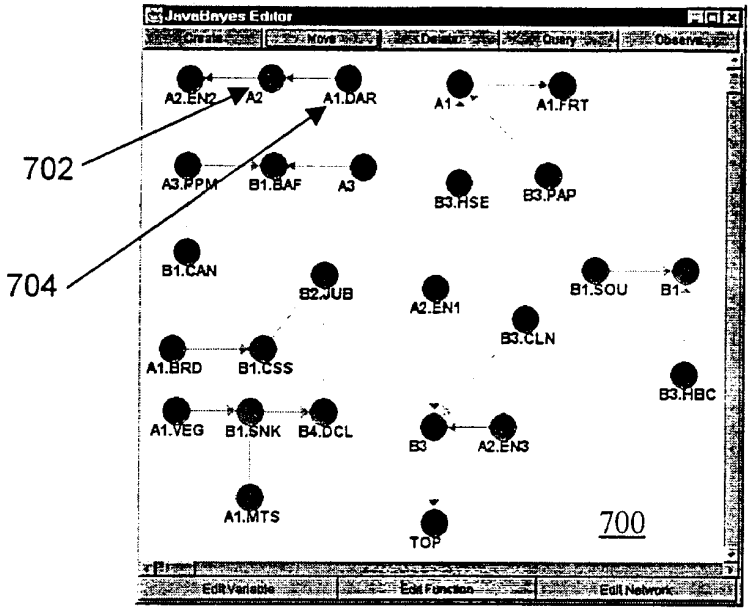

JavaBayes Editor
Create
Move
Delete
Query
Observe
A2.EN2
A2
A1.DAR
A1
A1.FRT
702
A3.PPM
B1.BAF
A3
B3.HSE
B3.PAP
704
B1.CAN
B2.JUB
A2.EN1
B1.SOU
B1
B3.CLN
A1.BRD
B1.CSS
B3.HBC
A1.VEG
B1.SNK
B4.DCL
B3
A2.EN3
A1.MTS
TOP
700
Edit Variable
Edit Function
Edit Network

FIG. 7A

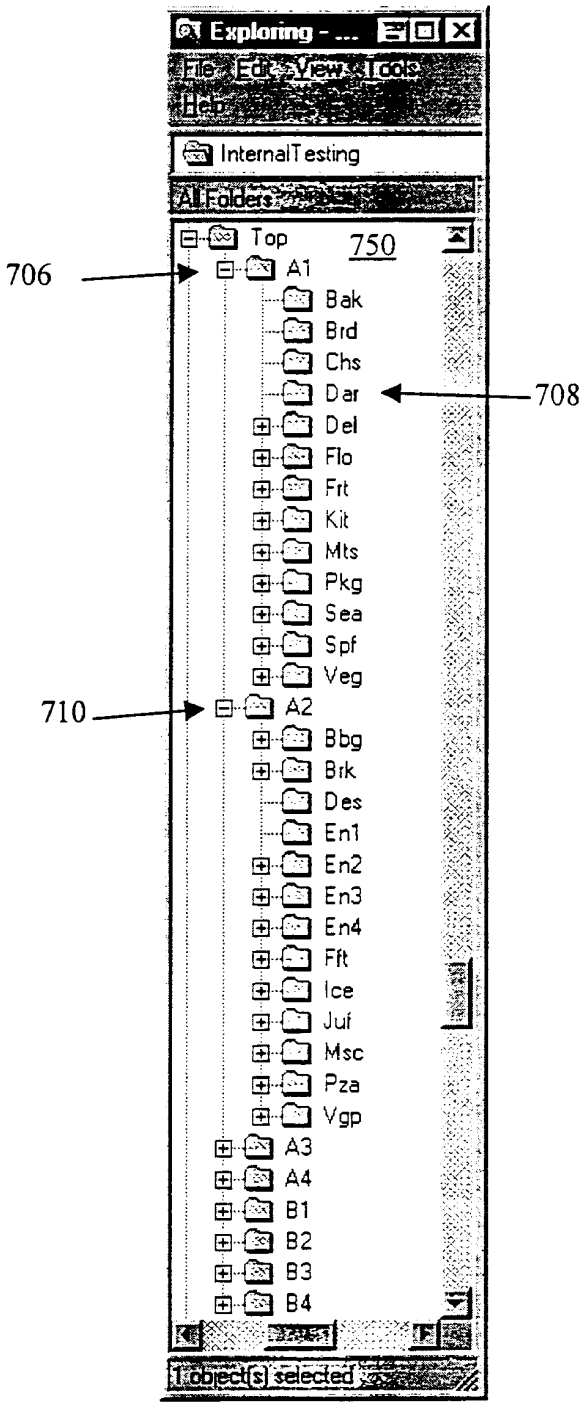

Exploring - ...
File Edit View Tools
Help
InternalTesting
All Folders
Top
750
706
A1
Bak
Brd
Chs
Dar
708
Del
Flo
Frt
Kit
Mts
Pkg
Sea
Spf
Veg
710
A2
Bbg
Brk
Des
En1
En2
En3
En4
Fft
Ice
Juf
Msc
Pza
Vgp
A3
A4
B1
B2
B3
B4
1 object(s) selected

FIG. 7B

```
<HYPER_LINKS>
<HYPER_LINK MODEL1="A0.A0B9" MODEL2="A7"
COUPLING="0.7334138956943544">
        <SHVARS VAR="A0.A0B9"/>
        <SHVARS VAR="A7.A7B5"/>
</HYPER_LINK>
<HYPER_LINK MODEL1="A5.A5B6" MODEL2="A7.A7B8"
COUPLING="0.7747616246161703">
        <SHVARS VAR="A5.A5B6"/>
        <SHVARS VAR="A7.A7B8"/>
</HYPER_LINK>
</HYPER_LINKS>
```

800

FIG. 8

900

'TOP','0'
'A1','BAK','BAKBAG','0'
'A1','0'
'A1','DEL','0'
'A1','BRD','0'
'A1','DEL','DELMTS','0'
'A1','0'
'A1','MTS','0'
'B2','BWS','BWSLIQ','1'
'A1','1'
'A1','BRD','2'
'A1','2'
'A1','CHS','2'
'A1','CHS','CHZGMT','2'
'A2','DES','FRZPIE','2'
'A2','2'
'A2','EN1','FRZE1D','2'
'A2','EN1','2'
'TOP','3'
'A1','3'
'A1','BAK','3'
'TOP','3'
'A2','3'

Fig. 9

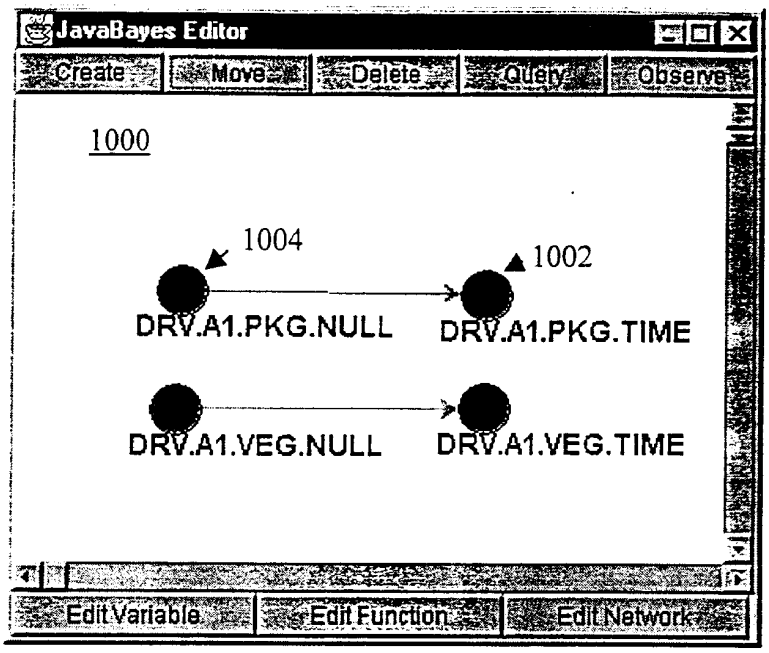

JavaBayes Editor
Create
Move
Delete
Query
Observe
1000
1004
1002
DRV.A1.PKG.NULL
DRV.A1.PKG.TIME
DRV.A1.VEG.NULL
DRV.A1.VEG.TIME
Edit Variable
Edit Function
Edit Network

FIG. 10

1100

Component	Full Name	Sends QRC Agent	Accepts QRC Agent	Works with ORB objects?	Is ORB Object?
CMR	Central Model Repository	To LM	From LM	With LU Shuttle	Yes
LM	Learning Manager	To CMR	From CMR		
LU	Learning Unit				
LU Shuttle	Learning Unit Shuttle			With CMR	Yes

Fig. 11

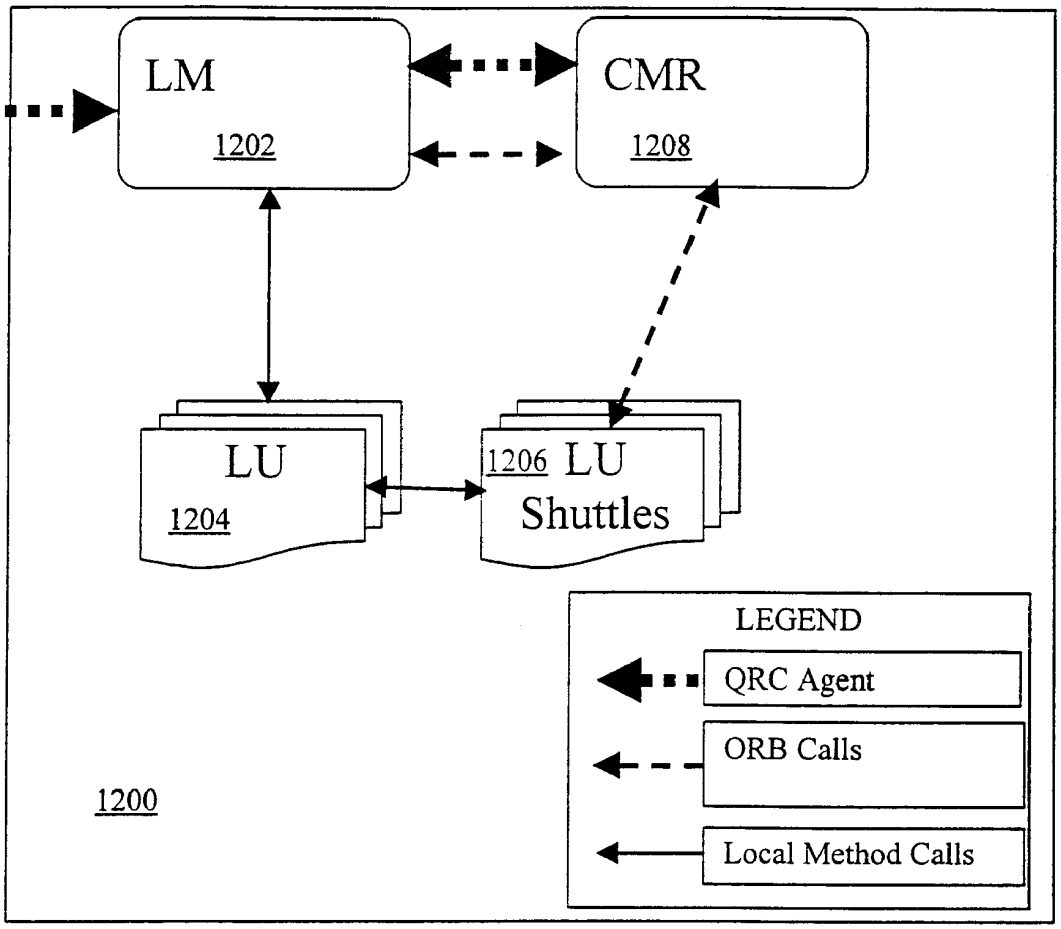

LM
1202
CMR
1208
LU
1204
1206 LU
Shuttles
LEGEND
QRC Agent
ORB Calls
Local Method Calls
1200

FIG. 12A

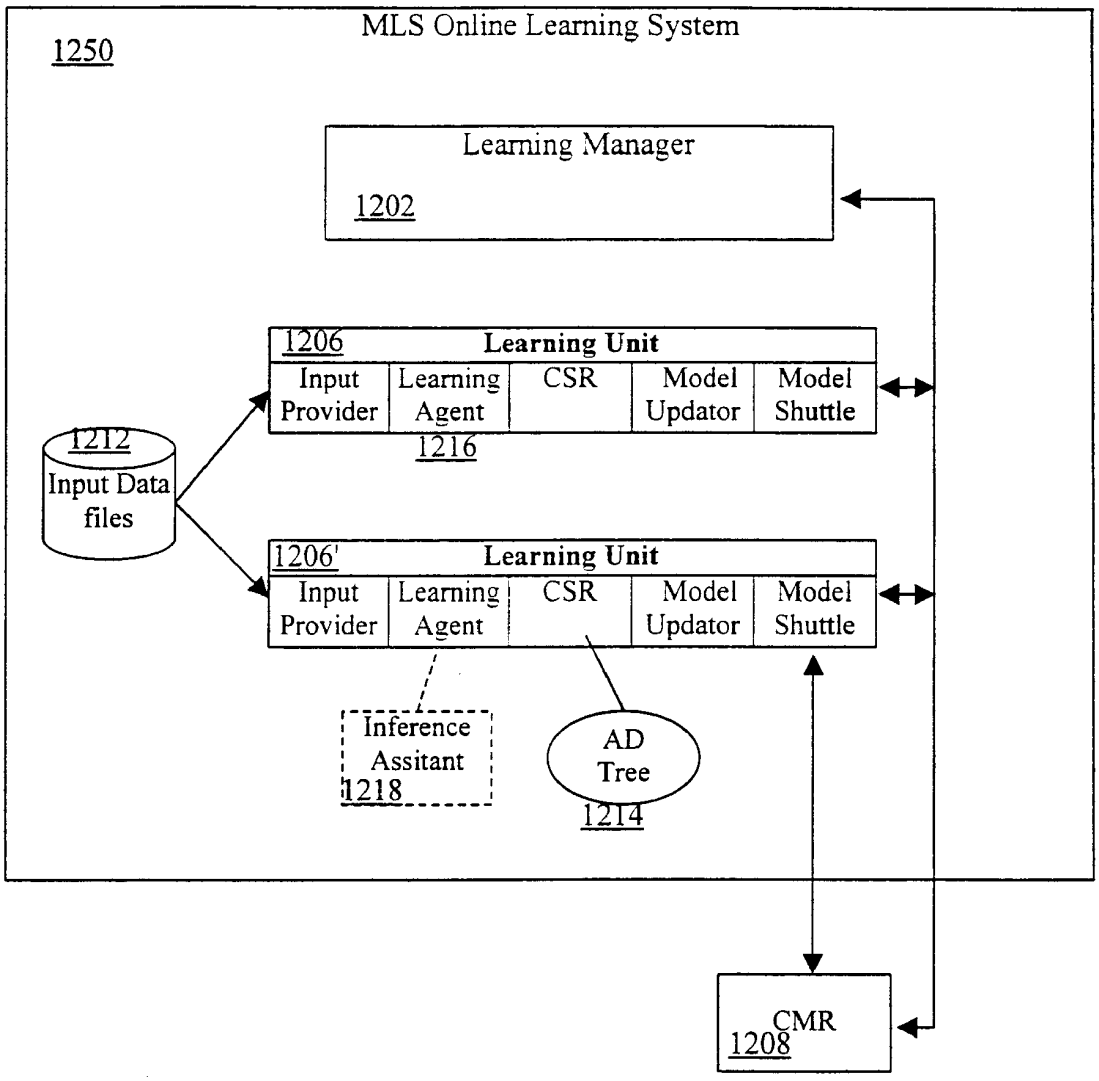

MLS Online Learning System
1250
Learning Manager
1202
1206
Learning Unit
Input Provider
Learning Agent
CSR
Model Updator
Model Shuttle
1216
1212
Input Data files
1206'
Learning Unit
Input Provider
Learning Agent
CSR
Model Updator
Model Shuttle
Inference Assitant
1218
AD Tree
1214
CMR
1208

FIG. 12B

1300

Com-ponent	Full name	No.	Interaction partner	Interaction type	Interaction
CMR	Central Model Repository	1	Learning Manager	Accept QRC Agent	Request for model names, & CMR proxy.
			"	Send Reply	Supply model names, proxy.
			Model Shuttle	Method Invoked	Receive learned model notification.
			"	Remote Method	Request to supply the latest learned Model.
LM	Learning Manager	K	CMR	Send QRC Agent	Request for model names, & CMR proxy.
			"	Accept Reply	Supplied model names, proxy.
			Admin	Accept QRC Agent	Load system, (don't start learning yet)
			"	"	Activation, start the learning.
			"	"	Configure, perform dynamic configuration.
			"	"	Shutdown (gracefully).
			"	"	Suspend, one or more LUs.
			"	"	Resume, one or more LUs.
			"	"	Category change
			LU	Construct	Create new LU .
			"	Invoke Method	Restart from checkpoint
				"	Dynamic Configuration
				"	Request Shutdown LU.
				"	Request LU Suspension
			"	"	Request LU resumption.
			"	"	Request for No. of cases learned by the LU.
			"	Method Invoked	Return total No. of cases processed by LUs of this LM.

Fig. 13A

LU	Learning Unit	M	Learning Manager	Constructed	Created by LM.
			"	Method invoked	Restart from checkpoint
			"	"	Dynamic Configuration
			"	"	Shutdown, (kill all dedicated threads).
			"	"	Suspend.
			"	"	Resume
			"	"	Return No. of cases learned by the LU.
			Model Shuttle	Construct	Create a dedicated Shuttle
			"	Method invoke	Restart from checkpoint
			"	"	Instruct Shuttle to performDynamic Configuration
			"	"	Instruct Shuttle to Shutdown
			"	"	Request Shuttle to retrieve model from CMR.
MSH	Model Shuttle	M	Learning Unit	Constructed	Created by owner LU.
			"	Method invoked	Restart from checkpoint.
			"	"	Perform Dynamic Configuration
			"	"	Shutdown
			"	"	Return model retrieved from CMR.
			"	"	Receive new model learned by the LU.
			CMR	Remote Method (oneway)	Notify CMR that new model is available.
			"	Remote Method	Return latest learned model.

Fig. 13B

1400

QRC Agent	From	To	Data	Reply	Scenario
Load System	Admin	LM			Loading
Request model names & CMR proxy	LM	CMR		List of names & proxy.	Loading
Activate System	Admin	LM			Activation
Dynamic Configuration	Admin	LM		MLS config & IS config	Dynamic Configuration
Shutdown	Admin	LM			Shutdown
Suspend LU	Admin	LM		LU name	LU suspension
Resume LU	Admin	LM		LU name	LU resumption

Fig. 14

1500

Comp.	Full Name	No.	Interaction Partner	Interaction Type	Interaction Description
CMR	Central Model Repository	1	Inference Manager	Accept QRC Agent	Request for net structure
CMR			"	Send Reply	Supply net structure
CMR			"	Send QRC Agent	Notify Model Update
CMR			Inference Agent	Method Invoked	Supply Model
IM	Inference Manager	K	CMR	Send QRC Agent	Request for net structure
IM			"	Accept Reply	Supplied net structure.
IM			"	Accept QRC Agent	Notification on Updated Bayes Model
IM			Client	Accept QRC Agent	Product Order Evidence
IM			"	"	Category change.
IM			Rule - Business object.	Accept QRC Agent	Request for recommenda-tion.
IM			Inference Handler	Invoke Method	Perform Query
IM			"	"	Perform Recommend
IM			"	"	Propagate Evidence
IM			"	"	Category Change
IM			Inference Agent	Remote Method	Session Terminated
IM			"	"	Update model

Fig. 15A

IH	Inference Handler	K	Inference Manager	Method Invoked	Request for query.
IH			"	"	Request for recommend.
IH			"	"	Propagate Evidence.
IH			"	"	Handle category change.
IH			Rule - Business object.	Send Reply	Recommend-ation.
IH			Inference Agent	Remote Method	Perform Query
IH			"	"	Perform Recommend.
IH			"	"	Propagate evidence.
IH			"	"	Absorb evidence from neighbors.
IH			"	"	Update Probabilistic Evidence.

Fig. 15B

IA	Inference Agent	M	Inference Handler	Method Invoked	Receive Real Evidence.
IA			"	"	Request for query.
IA			"	"	Request for recommend.
IA			"	"	Absorb Neighbors Evidence
IA			"	"	Update Probabilistic Evidence.
IA			Inference Manager	Method Invoked	Update Model
IA			"	"	Session Terminated
IA			Inference Agent	Remote Method	Propagate Prob Evidence
IA			"	Remote Method	Absorb Neighbors Evidence
IA			"	Method Invoked	Propagate Prob Evidence
IA			"	Method Invoke	Absorb Neighbors Evidence
IA			Inference Session	Invoke Method	Update Session state
IA			"	Invoke Method	Get Session's Snapshot
IA			JBayes Handler	Invoke Method	Request for calculation
IS	Inference Session	S * M	Inference Agent	Method Invoked	Update Session state
IS			"	Method Invoked	Get Session's Snapshot
JBH	Jbayes Handler	M	Inference Agent	Method Invoked	Request for calculation

Fig. 15C

QRC Agent	From	To	Data	Reply	Scenario
Request Net Structure	IM	CMR		Net Structure object	IM initiation
Update Model notification	CMR	IM	Model name		IM update
New Evidence	AI SYS	IM	New Real evidence		IM evidence
Category Change	AI SYS	IM	Category Sliding Window		IM category change
Inference Recommend	AI Business object	IM	Query Recommend-ation	Query Result object	IM query

Fig. 16

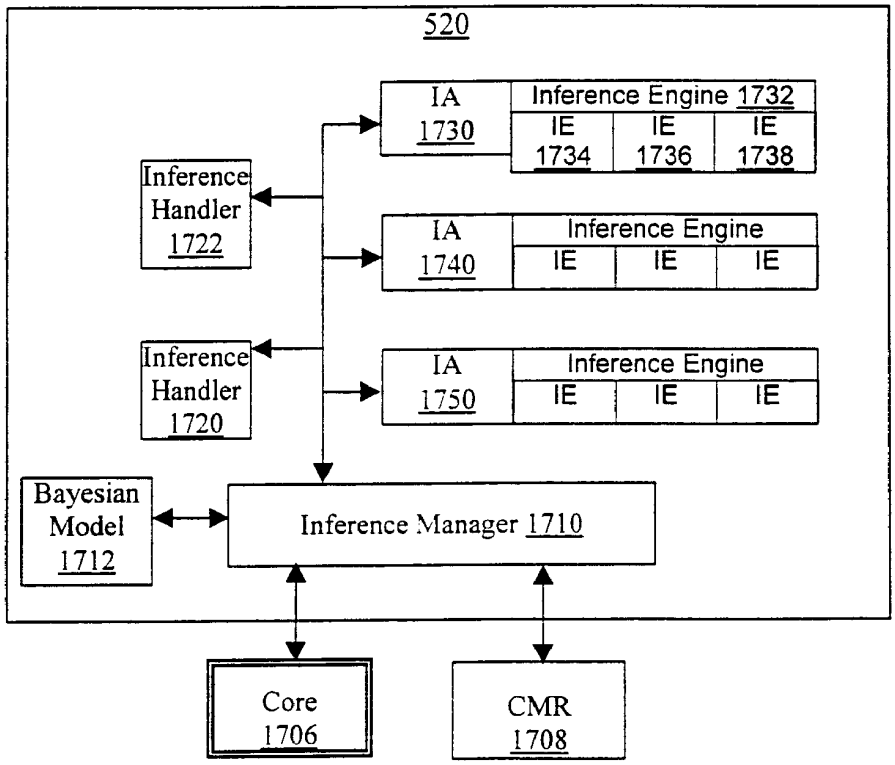


Fig. 17

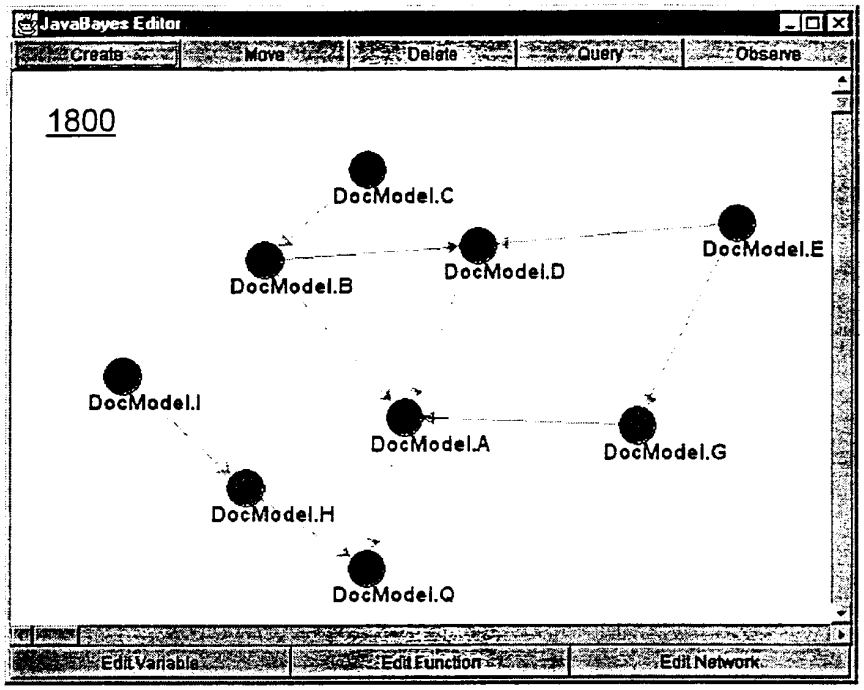

JavaBayes Editor
Create
Move
Delete
Query
Observe
1800
DocModel.C
DocModel.B
DocModel.D
DocModel.E
DocModel.I
DocModel.A
DocModel.G
DocModel.H
DocModel.Q
Edit Variable
Edit Function
Edit Network

Fig. 18

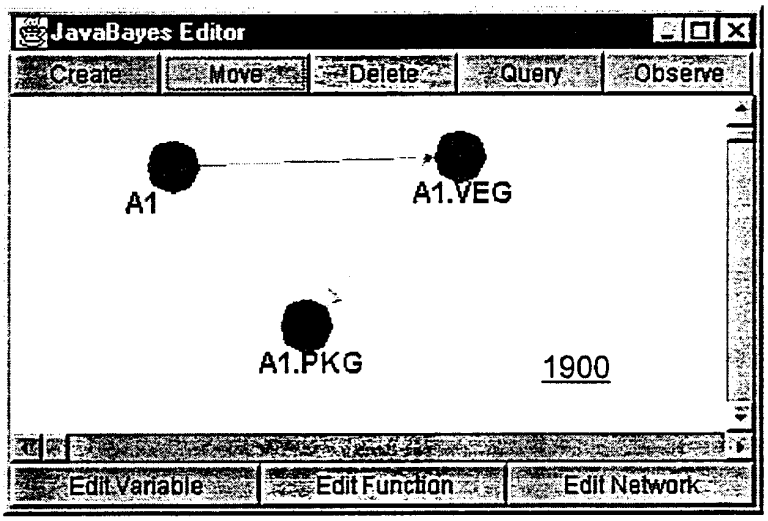

JavaBayes Editor
Create
Move
Delete
Query
Observe
A1
A1.VEG
A1.PKG
1900
Edit Variable
Edit Function
Edit Network

Fig. 19

```
<DAT>
<INF_CATEGORY_CHANGE_EVENT>
<EVENT_ID VALUE= eventID \> </EVENT_ID>
<CATEGORY VALUE= AI entity code string \>
</CATEGORY>
<CATEGORY VALUE= AI entity code string \>
</CATEGORY>
<CATEGORY VALUE= AI entity code string \>
</CATEGORY>
<CATEGORY VALUE= AI entity code string \>
</CATEGORY>
</INF_CATEGORY_CHANGE_EVENT>
</DAT>                                          2000
```

Fig. 20

```
<DAT>
   <INF_RECOMMEND_EVENT >
         <  MODEL          VALUE ="A1.B1"       />
          <  SUBJECT       VALUE ="A1.B1.C1"  />
          <  MAX_ITEMS_NUM    VALUE = 5              />
          <  EVAL_CRIT     VALUE =  PROB_RECOMMEND_TYPE/>
  </INF_RECOMMEND_EVENT>
</DAT>
```

2110

Fig. 21A

```
<DAT>
   <INF_RECOMMEND_EVENT >
         <  MODEL          VALUE ="A4"       />
          <  SUBJECT       VALUE =" Shallow-Agent-Recommend"   />
          <  MAX_ITEMS_NUM    VALUE = 5              />
          <EVAL_CRIT       VALUE =  DELTA_RECOMMEND_TYPE />
   </INF_RECOMMEND_EVENT>
</DAT>
```

2120

Fig. 21B

```
<DAT>
   <INF_RECOMMEND_EVENT >
         <  MODEL          VALUE ="A3.B2"       />
          <  SUBJECT       VALUE ="Deep-Agent-Recommend"  />
          <  MAX_ITEMS_NUM     VALUE = 5              />
          <  EVAL_CRIT     VALUE =  PROB_RECOMMEND_TYPE  />
          <  MAX_ATTRIBS   VALUE = 2 />
          <  MAX_ATTRIB_VALS   VALUE =  5 />
   </INF_RECOMMEND_EVENT>
</DAT>
```

2130

Fig. 21C

```
<DAT>
   <INF_RECOMMEND_EVENT >
         < MODEL VALUE ="A3"       />
         < SUBJECT VALUE ="Shallow-Neighbors-Agent-Recommend"/>
         < MAX_ITEMS_NUM VALUE = 5              />
         < EVAL_CRIT VALUE =  PROB_RECOMMEND_TYPE/>
         < MAX_NEIGHBORS   VALUE = 3 />
         < MAX_NEIGHBOR_ATTRIBS   VALUE =  2 />
   </INF_RECOMMEND_EVENT>
</DAT>
```

2140

Fig. 21D

```
<DAT>
   <INF_RECOMMEND_EVENT >
         <  MODEL   VALUE ="A1"   />
          <  SUBJECT VALUE ="Shallow-Neighbors-Agent-Recommend"/>
          <  MAX_ITEMS_NUM  VALUE = 10    />
          <  EVAL_CRIT VALUE =  PROB_RECOMMEND_TYPE  />
          <  MAX_NEIGHBORS   VALUE = 2 />
          <  MAX_NEIGHBOR_ATTRIBS   VALUE =  3 />
   </INF_RECOMMEND_EVENT>
</DAT>
```

2150

Fig. 21E

Context	Context Score	Subject Type	Subject	Subject Score
A1.BRD	0.47	C level category	A1.BRD.BRDPIZ	0.25
A1.BRD	0.47	C level category	A1.BRD.BRDCRM	0.17
A1.FRT	0.32	C level category	A1.FRT.PROJAR	0.61
A1.FRT	0.32	C level category	A1.FRT.FROPA	0.12
A1.FRT	0.32	C level category	A1.FRT.PROFGR	0.05

2200

Fig. 22

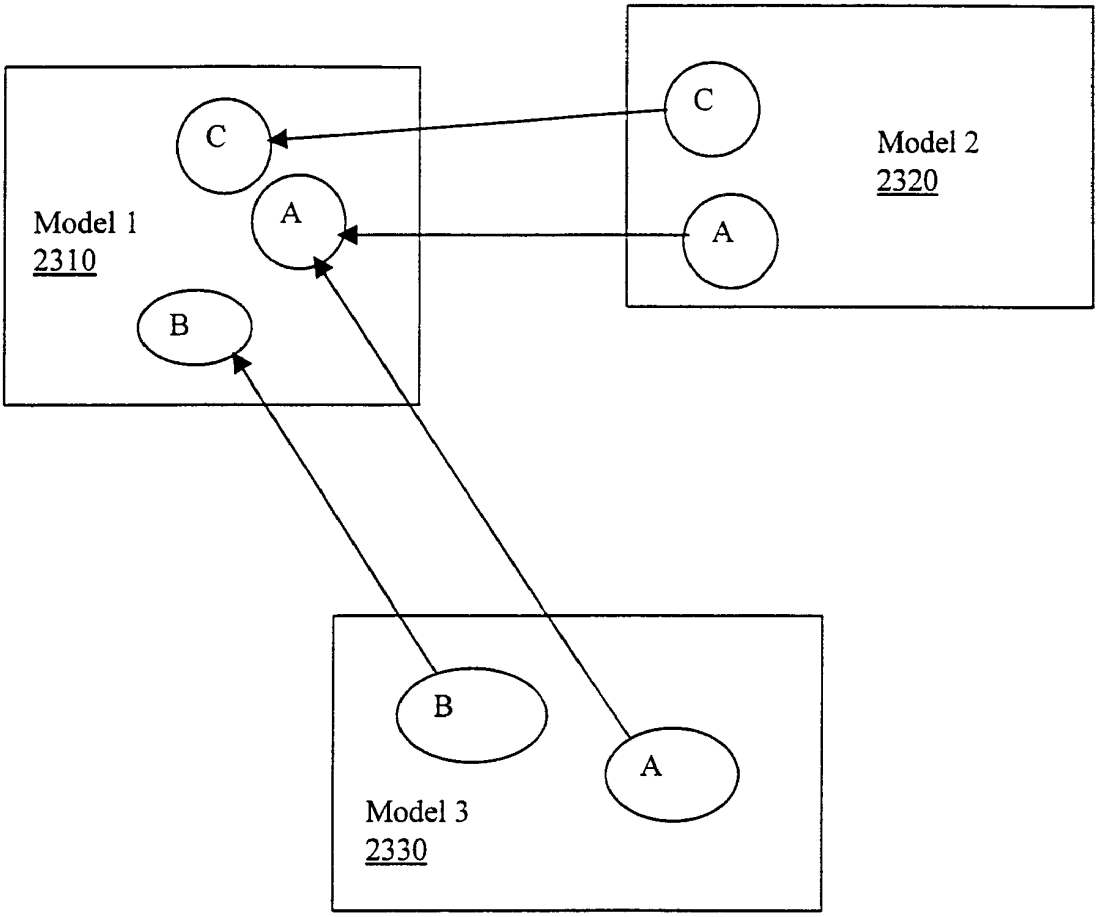


Fig. 23

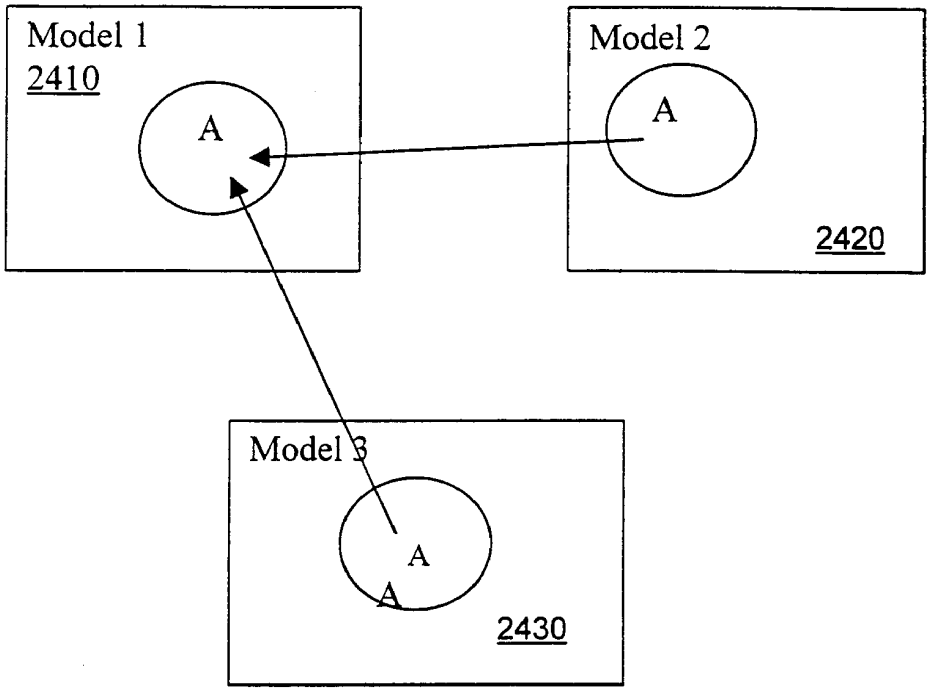

Model 1
2410
A
Model 2
A
2420
Model 3
A
A
2430

Fig. 24

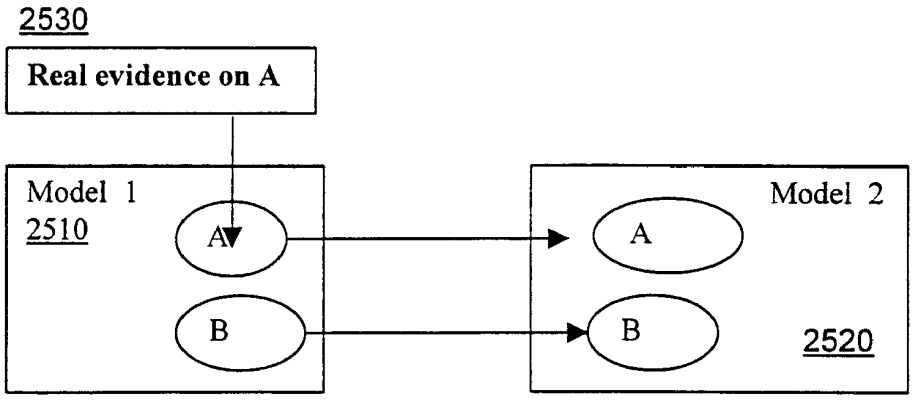

2530
Real evidence on A
Model 1
2510
A
B
Model 2
A
B
2520

Fig. 25

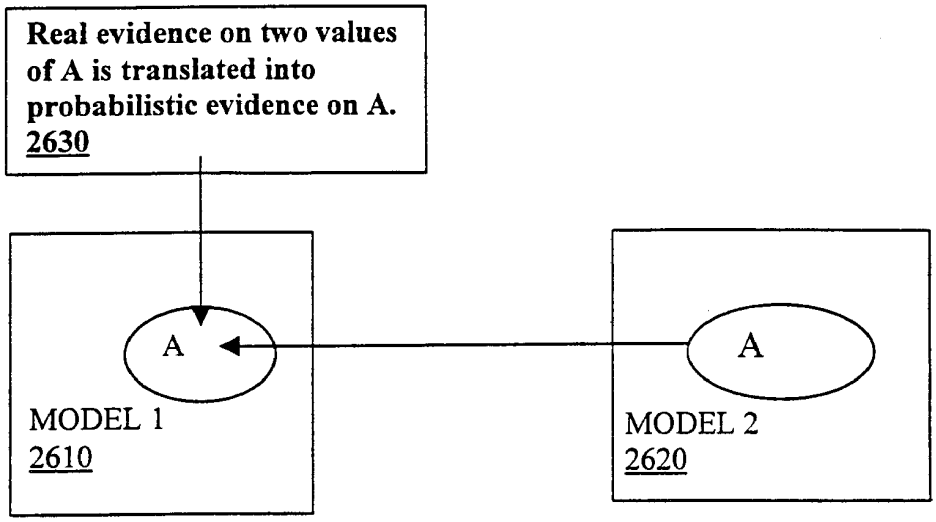

Real evidence on two values of A is translated into probabilistic evidence on A.
2630
A
MODEL 1
2610
A
MODEL 2
2620

Fig. 26

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/13360

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) :G06F 15/18, 17/30, 17/60; G06N, 5/02; G05B, 13/02
US CL : 705/1; 706/52, 59; 707/1;364/513,148.04
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 705/1; 706/52, 59; 707/1;364/513,148.04

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

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

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 4,884,217 A (SKEIRIK et al) 28 November 1989, col. 10-col. 20, 1-68.	1-87
Y	US 5,822,745 A (HEKMATPOUR) 13 October 1998, col. 8-col. 33, lines 1-68.	1-87
Y,P	US 6,076,083 A (BAKER) 13 June 2000, abstract, col. 9, line 20-col. 13, line 68.	1-87
Y,P	US 5,999,908 A (ABELOW) 07 December 1999, col. 16-col. 90, lines 1-68.	1-87
A	US 5,867,799 A (LANG et al) 02 February 1999, col. 7-col. 30, lines 1-68.	1-87
Y,P	US 5,963,447 A (KOHN et al) 05 October 1999, col. 11-col. 52, lines 1-68.	1-87

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

*	Special categories of cited documents	"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
"A"	document defining the general state of the art which is not considered to be of particular relevance	"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
"E"	earlier document published on or after the international filing date	"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
"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)	"&"	document member of the same patent family
"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		

Date of the actual completion of the international search 02 AUGUST 2000	Date of mailing of the international search report 15 AUG 2000
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer HIEU C. LE James R. Matthews Telephone No. (703) 306-3101

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/13360

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This international report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

Please See Extra Sheet.

1. ☒ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☒ No protest accompanied the payment of additional search fees.

BOX II. OBSERVATIONS WHERE UNITY OF INVENTION WAS LACKING
This ISA found multiple inventions as follows:

Group I, claim(s) 1-23, drawn to an intelligent computer system for making real-time recommendations and prediction within the context of a domain, said the system comprising:
- at least one applicant server linked to a first storage media, having a plurality of Bayesian models stored therein,
- at least one client device, and
- an intelligence applicant module.
Group II, claim(s) 24-28, drawn to an intelligent computer system for making real-time recommendations and prediction within the context of an e-commerce domain, said the system comprising:
- at least one applicant server linked to a first storage media, having a plurality of Bayesian models stored therein,and
- at least one Web server disposed
Group III, claim(s) 29-70 , drawn to an inference system as part of a computer system definig an application domain, wherein said intelligence computer system is accessible by a plurality of computers via a network, said the system comprising:
- at least one server linked to a storage media.
- an inference module hosted on said server,
. attribute value predictions and
. evidence derived from domain related information.
Group VI, claim(s) 71-87, drawn to an machine learning system comprising:
- a processing device;
-an interface to application system;and
-an on-line learning module

The inventions listed as Groups I,II,III, &VI do not relate to a single inventive concept under PCT Rule 13.1 because, under PCT Rule 13.2, they lack the same or corresponding special technical features for the following reasons: the four groups of inventions are not linked as to form a single general inventives concept.