MULTI-SOURCE, MULTI-DIMENSIONAL, CROSS-ENTITY, MULTIMEDIA DATABASE PLATFORM APPARATUSES, METHODS AND SYSTEMS

[0001] This application for letters patent disclosure document describes inventive aspects that include various novel innovations (hereinafter "disclosure") and contains material that is subject to copyright, mask work, and/or other intellectual property protection. The respective owners of such intellectual property have no objection to the facsimile reproduction of the disclosure by anyone as it appears in published Patent Office file/records, but otherwise reserve all rights.

PRIORITY CLAIM

[0002] This application is a non-provisional of and claims priority under 35 USC §§ 119, 120 to: United States provisional patent application serial no. 61/594,063 filed February 2, 2012, entitled "CENTRALIZED PERSONAL INFORMATION PLATFORM APPARATUSES, METHODS AND SYSTEMS," attorney docket no. P-42185PRV|VISA-122/01US, and United States patent application serial no. 13/520,481 filed July 3, 2012, entitled "Universal Electronic Payment Apparatuses, Methods and Systems," attorney docket no. P-42051US02|20270-136US.

[0003] The entire contents of the aforementioned application(s) are expressly incorporated by reference herein.

FIELD

[0004] The present innovations generally address apparatuses, methods and systems for consumer data management and analytics, and more particularly, include MULTI-SOURCE, MULTI-DIMENSIONAL, CROSS-ENTITY, MULTIMEDIA DATABASE PLATFORM APPARATUSES, METHODS AND SYSTEMS ("MMCDB").

[0005] However, in order to develop a reader's understanding of the innovations, disclosures have been compiled into a single description to illustrate and clarify how aspects of these innovations operate independently, interoperate as between individual innovations, and/or cooperate collectively. The application goes on to further describe the interrelations and synergies as between the various innovations; all of which is to further compliance with 35 U.S.C. §112.

BACKGROUND

[0006] Many people share information through online social media streams using various communication devices or applications. Moreover, consumers may access online stores using the Internet on a computer or mobile device to make purchases from various service providers or merchants.

SUMMARY

[0006a] According to an aspect, there is provided a centralized personal information platform processor-implemented method for enhancing transaction speed through the reduction of user input data transfer requirements, comprising: aggregating data records including search results, purchase transaction data, service usage data, service enrollment data, email data and social data; identifying data field types within the data records and their associated data values; identifying an entity from the data field types and their associated data values; generating, via a processor, correlations of the entity to other entities identifiable from the data field types and their associated data values by comparing fields from a non-normalized transaction data record with fields extracted from a normalized transaction data record template and determining whether the field identifiers of fields in the non-normalized transaction data record match any of the normalized transaction data record template, based on the comparison, wherein a map is generated between fields of the non-normalized transaction data record and normalized transaction data record template; associating, via the processor, attributes to the entity by drawing inferences related to the entity from the data field types and their associated data values; generating an updated profile and social graph of the entity using the generated correlations and associated attributes; and providing the updated profile and social graph for an automated web form filling request.

[0006b] There is disclosed a merchant analytics platform processor-implemented method for reduced transaction wait processing requirements through the use of customized transaction parameters based on a distributed linking node mesh, comprising: obtaining a request for a merchant analytics report including a user identification; aggregating user data of the user in a centralized personal information database; retrieving the aggregated user data in response to obtaining the request for the merchant analytics report; generating a user behavior profile using an analytical model, based on the aggregated user data retrieved from the centralized personal information database; providing the user behavior profile as part of the merchant analytics report.

[0006c] There is disclosed an analytical model sharing processor-implemented method for privacy enhanced analytical model sharing through the use of contextual privacy dataset modifications, comprising: obtaining a request to publish an analytical model operating on user data retrieved from a centralized personal information database; determining whether the analytical model includes a model dataset; upon determining that the analytical model includes a model dataset, determining whether the model dataset includes personally identifiable information; and generating a determination of whether to accept the request to publish the analytical model based on whether the model dataset includes personally identifiable information.

[0006d] There is disclosed an encryptmatics extensible markup language data conversion processor-implemented method for increased efficiency in contextless user model sharing through the use of intermediary meta-language processing, comprising: receiving an input model containing non-meta-data based language commands; retrieving input language definition records corresponding to the input model language commands; retrieving meta-data based language definition records; and generating a meta-data based language command file using the input language definition records and the meta-based language definition records.

[0006e] There is disclosed a processor-implemented method, comprising: aggregating, from a plurality of entities, raw mesh entries comprising any of: emails, engagement transactions, financial transactions, social media entries, into memory; determining an mesh entry type for each raw mesh entry; placing contents of each raw mesh entry into an unprocessed mesh entry structure; setting the mesh entry type for the unprocessed mesh entry from the determined mesh entry type; generating a

dictionary hash entry from the raw mesh entry and saving it into the unprocessed mesh entry structure; updating a mesh entry dictionary with the unprocessed mesh entry structure; replicating the mesh entry dictionary to another location without the raw mesh entry in the mesh entry structure, wherein the replicated mesh entry dictionary is actionable for analysis without the raw mesh entry and with the dictionary has entry and set mesh entry type; storing the unprocessed mesh entry structure into a multi-directionally linked multimedia data mesh (MLMD mesh); determining correlations within the unprocessed mesh entry structure with other stored mesh entry structures in the MLMD mesh; creating links to the determined correlated stored mesh entry structures and storing them in the stored unprocessed mesh entry structure; marking the unprocessed mesh entry structure as a processed mesh entry structure.

[0006f] According to another aspect, there is provided a centralized personal information platform processor-implemented system for enhancing transaction speed through the reduction of user input data transfer requirements, comprising: means to aggregate data records including search results, purchase transaction data, service usage data, service enrollment data, email data and social data; means to identify data field types within the data records and their associated data values; means to identify an entity from the data field types and their associated data values; means to generate, via a processor, correlations of the entity to other entities identifiable from the data field types and their associated data values by comparing fields from a non-normalized transaction data record with fields extracted from a normalized transaction data record template and determining whether the field identifiers of fields in the non-normalized transaction data record match any of the normalized transaction data record template, based on the comparison, wherein a map is generated between fields of the non-normalized transaction data record and normalized transaction data record template; means to associate, via the processor, attributes to the entity by drawing inferences related to the entity from the data field types and their associated data values; means to generate an updated profile and social graph of the entity using the generated correlations and associated attributes; and means to provide the updated profile and social graph for an automated web form filling request.

[0006g] There is disclosed a merchant analytics platform processor-implemented system for reduced transaction wait processing requirements through the use of customized transaction parameters based on a distributed linking node mesh, comprising: means to obtain a request for a merchant analytics report including a user

identification; means to aggregate user data of the user in a centralized personal information database; means to retrieve the aggregated user data in response to obtaining the request for the merchant analytics report; means to generate a user behavior profile using an analytical model, based on the aggregated user data retrieved from the centralized personal information database; means to provide the user behavior profile as part of the merchant analytics report.

[0006h] There is disclosed an analytical model sharing processor-implemented system for privacy enhanced analytical model sharing through the use of contextual privacy dataset modifications, comprising: means to obtain a request to publish an analytical model operating on user data retrieved from a centralized personal information database; means to determine whether the analytical model includes a model dataset; means to upon determining that the analytical model includes a model dataset, determining whether the model dataset includes personally identifiable information; and means to generate a determination of whether to accept the request to publish the analytical model based on whether the model dataset includes personally identifiable information.

[0006i] There is disclosed an encryptmatics extensible markup language data conversion processor-implemented system for increased efficiency in contextless user model sharing through the use of intermediary meta-language processing, comprising: means to receive an input model containing non-meta-data based language commands; means to retrieve input language definition records corresponding to the input model language commands; means to retrieve meta-data based language definition records; and means to generate a meta-data based language command file using the input language definition records and the meta-based language definition records.

[0006j] There is disclosed an processor-implemented system, comprising: means to aggregate, from a plurality of entities, raw mesh entries comprising any of: emails, engagement transactions, financial transactions, social media entries, into memory; means to determine an mesh entry type for each raw mesh entry; means to place contents of each raw mesh entry into an unprocessed mesh entry structure; means to set the mesh entry type for the unprocessed mesh entry from the determined mesh entry type; means to generate a dictionary hash entry from the raw mesh entry and saving it into the unprocessed mesh entry structure; means to update a mesh entry dictionary with the unprocessed mesh entry structure; means to replicate the mesh entry dictionary to another location without the raw mesh entry in the mesh entry structure, wherein the replicated mesh

entry dictionary is actionable for analysis without the raw mesh entry and with the dictionary has entry and set mesh entry type; means to store the unprocessed mesh entry structure into a multi-directionally linked multimedia data mesh (MLMD mesh); means to determine correlations within the unprocessed mesh entry structure with other stored mesh entry structures in the MLMD mesh; means to create links to the determined correlated stored mesh entry structures and storing them in the stored unprocessed mesh entry structure; means to mark the unprocessed mesh entry structure as a processed mesh entry structure.

[0006k] According to another aspect, there is provided a centralized personal information platform processor-implemented apparatus for enhancing transaction speed through the reduction of user input data transfer requirements, comprising: a memory; a processor disposed in communication with said memory, and configured to issue a plurality of processing instructions stored in the memory, wherein the processor issues instructions to: aggregate data records including search results, purchase transaction data, service usage data, service enrollment data, email data and social data; identify data field types within the data records and their associated data values; identify an entity from the data field types and their associated data values; generate, via a processor, correlations of the entity to other entities identifiable from the data field types and their associated data values by comparing fields from a non-normalized transaction data record with fields extracted from a normalized transaction data record template and determining whether the field identifiers of fields in the non-normalized transaction data record match any of the normalized transaction data record template, based on the comparison, wherein a map is generated between fields of the non-normalized transaction data record and normalized transaction data record template; associate, via the processor, attributes to the entity by drawing inferences related to the entity from the data field types and their associated data values; generate an updated profile and social graph of the entity using the generated correlations and associated attributes; and provide the updated profile and social graph for an automated web form filling request.

[0006l] There is disclosed a merchant analytics platform processor-implemented apparatus for reduced transaction wait processing requirements through the use of customized transaction parameters based on a distributed linking node mesh, comprising: a memory; a processor disposed in communication with said memory, and configured to issue a plurality of processing instructions stored in the memory, wherein the processor issues instructions to: obtainin a request for a merchant analytics report

including a user identification; aggregate user data of the user in a centralized personal information database; retrieve the aggregated user data in response to obtaining the request for the merchant analytics report; generate a user behavior profile using an analytical model, based on the aggregated user data retrieved from the centralized personal information database; provide the user behavior profile as part of the merchant analytics report.

[0006m] There is disclosed an analytical model sharing processor-implemented apparatus for privacy enhanced analytical model sharing through the use of contextual privacy dataset modifications, comprising: a memory; a processor disposed in communication with said memory, and configured to issue a plurality of processing instructions stored in the memory, wherein the processor issues instructions to: obtain a request to publish an analytical model operating on user data retrieved from a centralized personal information database; determine whether the analytical model includes a model dataset; upon determining that the analytical model includes a model dataset, determining whether the model dataset includes personally identifiable information; and generate a determination of whether to accept the request to publish the analytical model based on whether the model dataset includes personally identifiable information.

[0006n] There is disclosed an encryptmatics extensible markup language data conversion processor-implemented apparatus for increased efficiency in contextless user model sharing through the use of intermediary meta-language processing, comprising: a memory; a processor disposed in communication with said memory, and configured to issue a plurality of processing instructions stored in the memory, wherein the processor issues instructions to: receive an input model containing non-meta-data based language commands; retrieve input language definition records corresponding to the input model language commands; retrieve meta-data based language definition records; and generate a meta-data based language command file using the input language definition records and the meta-based language definition records.

[0006o] There is disclosed an processor-implemented apparatus, comprising: a memory; a processor disposed in communication with said memory, and configured to issue a plurality of processing instructions stored in the memory, wherein the processor issues instructions to: aggregate, from a plurality of entities, raw mesh entries comprising any of: emails, engagement transactions, financial transactions, social media entries, into memory; determine an mesh entry type for each raw mesh entry;

place contents of each raw mesh entry into an unprocessed mesh entry structure; set the mesh entry type for the unprocessed mesh entry from the determined mesh entry type; generate a dictionary hash entry from the raw mesh entry and saving it into the unprocessed mesh entry structure; update a mesh entry dictionary with the unprocessed mesh entry structure; replicate the mesh entry dictionary to another location without the raw mesh entry in the mesh entry structure, wherein the replicated mesh entry dictionary is actionable for analysis without the raw mesh entry and with the dictionary has entry and set mesh entry type; store the unprocessed mesh entry structure into a multi-directionally linked multimedia data mesh (MLMD mesh); determine correlations within the unprocessed mesh entry structure with other stored mesh entry structures in the MLMD mesh; create links to the determined correlated stored mesh entry structures and storing them in the stored unprocessed mesh entry structure; mark the unprocessed mesh entry structure as a processed mesh entry structure.

[0006p] According to another aspect, there is provided a non-transitory medium storing processor-issuable instructions for a centralized personal information platform processor-implemented to: aggregate data records including search results, purchase transaction data, service usage data, service enrollment data, email data and social data; identify data field types within the data records and their associated data values; identify an entity from the data field types and their associated data values; generate, via a processor, correlations of the entity to other entities identifiable from the data field types and their associated data values by comparing fields from a non-normalized transaction data record with fields extracted from a normalized transaction data record template and determining whether the field identifiers of fields in the non-normalized transaction data record match any of the normalized transaction data record template, based on the comparison, wherein a map is generated between fields of the non-normalized transaction data record and normalized transaction data record template; associate, via the processor, attributes to the entity by drawing inferences related to the entity from the data field types and their associated data values; generate an updated profile and social graph of the entity using the generated correlations and associated attributes; and provide the updated profile and social graph for an automated web form filling request.

[0006q] There is disclosed a merchant analytics platform processor-implemented medium storing instructions for reduced transaction wait processing requirements through the use of customized transaction parameters based on a

distributed linking node mesh to: obtain a request for a merchant analytics report including a user identification; aggregate user data of the user in a centralized personal information database; retrieve the aggregated user data in response to obtaining the request for the merchant analytics report; generate a user behavior profile using an analytical model, based on the aggregated user data retrieved from the centralized personal information database; provide the user behavior profile as part of the merchant analytics report.

[0006r] There is disclosed an analytical model sharing processor-implemented medium for privacy enhanced analytical model sharing through the use of contextual privacy dataset modifications, comprising instructions to: obtain a request to publish an analytical model operating on user data retrieved from a centralized personal information database; determine whether the analytical model includes a model dataset; upon determining that the analytical model includes a model dataset, determining whether the model dataset includes personally identifiable information; and generate a determination of whether to accept the request to publish the analytical model based on whether the model dataset includes personally identifiable information.

[0006s] There is disclosed an encryptmatics extensible markup language data conversion processor-implemented medium storing instructions for increased efficiency in contextless user model sharing through the use of intermediary meta-language processing to: receive an input model containing non-meta-data based language commands; retrieve input language definition records corresponding to the input model language commands; retrieve meta-data based language definition records; and generate a meta-data based language command file using the input language definition records and the meta-based language definition records.

[0006t] There is disclosed an processor-implemented medium containing instructions to: aggregate, from a plurality of entities, raw mesh entries comprising any of: emails, engagement transactions, financial transactions, social media entries, into memory; determine an mesh entry type for each raw mesh entry; place contents of each raw mesh entry into an unprocessed mesh entry structure; set the mesh entry type for the unprocessed mesh entry from the determined mesh entry type; generate a dictionary hash entry from the raw mesh entry and saving it into the unprocessed mesh entry structure; update a mesh entry dictionary with the unprocessed mesh entry structure; replicate the mesh entry dictionary to another location without the raw mesh entry in the mesh entry

structure, wherein the replicated mesh entry dictionary is actionable for analysis without the raw mesh entry and with the dictionary has entry and set mesh entry type; store the unprocessed mesh entry structure into a multi-directionally linked multimedia data mesh (MLMD mesh); determine correlations within the unprocessed mesh entry structure with other stored mesh entry structures in the MLMD mesh; create links to the determined correlated stored mesh entry structures and storing them in the stored unprocessed mesh entry structure; mark the unprocessed mesh entry structure as a processed mesh entry structure.

BRIEF DESCRIPTION OF THE DRAWINGS

[0007] The accompanying appendices and/or drawings illustrate various non-limiting, example, innovative aspects in accordance with the present descriptions:

[0008] FIGURE 1 shows a block diagram illustrating example aspects of a centralized personal information platform in some embodiments of the MMCDB;

[0009] FIGURES 2A-F show block diagrams illustrating example aspects of data models within a centralized personal information platform in some embodiments of the MMCDB;

[0010] FIGURE 3 shows a block diagram illustrating example MMCDB component configurations in some embodiments of the MMCDB;

[0011] FIGURE 4 shows a data flow diagram illustrating an example search result aggregation procedure in some embodiments of the MMCDB;

[0012] FIGURE 5 shows a logic flow diagram illustrating example aspects of aggregating search results in some embodiments of the MMCDB, e.g., a Search Results Aggregation ("SRA") component 500;

[0013] FIGURES 6A-D show data flow diagrams illustrating an example card-based transaction execution procedure in some embodiments of the MMCDB;

[0014] FIGURES 7A-E show logic flow diagrams illustrating example aspects of card-based transaction execution, resulting in generation of card-based transaction data and service usage data, in some embodiments of the MMCDB, e.g., a Card-Based Transaction Execution ("CTE") component 700;

[0015] FIGURE 8 shows a data flow diagram illustrating an example procedure to aggregate card-based transaction data in some embodiments of the MMCDB;

[0016] FIGURE 9 shows a logic flow diagram illustrating example aspects of aggregating card-based transaction data in some embodiments of the MMCDB, e.g., a Transaction Data Aggregation ("TDA") component 900;

[0017] FIGURE 10 shows a data flow diagram illustrating an example social data aggregation procedure in some embodiments of the MMCDB;

[0018] FIGURE 11 shows a logic flow diagram illustrating example aspects of aggregating social data in some embodiments of the MMCDB, e.g., a Social Data Aggregation ("SDA") component 1100;

[0019] FIGURE 12 shows a data flow diagram illustrating an example procedure for enrollment in value-add services in some embodiments of the MMCDB;

[0020] FIGURE 13 shows a logic flow diagram illustrating example aspects of social network payment authentication enrollment in some embodiments of the MMCDB, e.g., a Value-Add Service Enrollment ("VASE") component 1300;

[0021] FIGURES 14A-B show flow diagrams illustrating example aspects of normalizing aggregated search, enrolled, service usage, transaction and/or other aggregated data into a standardized data format in some embodiments of the MMCDB, e.g., a Aggregated Data Record Normalization ("ADRN") component 1400;

[0022] FIGURE 15 shows a logic flow diagram illustrating example aspects of recognizing data fields in normalized aggregated data records in some embodiments of the MMCDB, e.g., a Data Field Recognition ("DFR") component 1500;

[0023] FIGURE 16 shows a logic flow diagram illustrating example aspects of classifying entity types in some embodiments of the MMCDB, e.g., an Entity Type Classification ("ETC") component 1600;

[0024] FIGURE 17 shows a logic flow diagram illustrating example aspects of identifying cross-entity correlation in some embodiments of the MMCDB, e.g., a Cross-Entity Correlation ("CEC") component 1700;

[0025] FIGURE 18 shows a logic flow diagram illustrating example aspects of associating attributes to entities in some embodiments of the MMCDB, e.g., an Entity Attribute Association ("EAA") component 1800;

[0026] FIGURE 19 shows a logic flow diagram illustrating example aspects of updating entity profile-graphs in some embodiments of the MMCDB, e.g., an Entity Profile-Graph Updating ("EPGU") component 1900;

[0027] FIGURE 20 shows a logic flow diagram illustrating example aspects of generating search terms for profile-graph updating in some embodiments of the MMCDB, e.g., a Search Term Generation ("STG") component 2000;

[0028] FIGURES 21A-E show user interface diagrams illustrating example features of user interfaces for an electronic virtual wallet in some embodiments of the MMCDB;

[0029] FIGURE 22 shows a block diagram illustrating example aspects of a merchant analytics platform in some embodiments of the MMCDB;

[0030] FIGURES 23A-B show data flow diagrams illustrating an example procedure to provide a user and/or merchant offers for products, services and/or the like, using user behavior patterns derived from card-based transaction data in some embodiments of the MMCDB;

[0031] FIGURE 24 shows a logic flow diagram illustrating example aspects of providing a user and/or merchant offers for products, services and/or the like, using user behavior patterns derived from card-based transaction data in some embodiments of the MMCDB, e.g., a Merchant Analytics ("MA") component;

[0032] FIGURE 25 shows a logic flow diagram illustrating example aspects of generating a user behavior pattern analysis in some embodiments of the MMCDB, e.g., a User Behavioral Pattern Analytics ("UBPA") component;

[0033] FIGURE 26 shows a logic flow diagram illustrating example aspects of identifying user behavioral patterns from aggregated card-based transaction data in some embodiments of the MMCDB, e.g., a User Patten Identification ("UPI") component;

[0034] FIGURES 27A-B show block diagrams illustrating example aspects of merchant analytics in a second set of embodiments of the MMCDB;

[0035] FIGURES 28A-C show data flow diagrams illustrating an example procedure for econometrical analysis of a proposed investment strategy based on card-based transaction data in some embodiments of the MMCDB;

[0036] FIGURE 29 shows a logic flow diagram illustrating example aspects of normalizing raw card-based transaction data into a standardized data format in some embodiments of the MMCDB, e.g., a Transaction Data Normalization ("TDN") component;

[0037] FIGURE 30 shows a logic flow diagram illustrating example aspects of generating classification labels for card-based transactions in some embodiments of the MMCDB, e.g., a Card-Based Transaction Classification ("CTC") component;

[0038] FIGURE 31 shows a logic flow diagram illustrating example aspects of filtering card-based transaction data for econometrical investment strategy analysis in some embodiments of the MMCDB, e.g., a Transaction Data Filtering ("TDF") component;

[0039] FIGURE 32 shows a logic flow diagram illustrating example aspects of anonymizing consumer data from card-based transactions for econometrical investment strategy analysis in some embodiments of the MMCDB, e.g., a Consumer Data Anonymization ("CDA") component;

[0040] FIGURES 33A-B show logic flow diagrams illustrating example aspects of econometrically analyzing a proposed investment strategy based on card-based

transaction data in some embodiments of the MMCDB, e.g., an Econometrical Strategy Analysis ("ESA") component;

[0041] FIGURE 34 shows a logic flow diagram illustrating example aspects of reporting business analytics derived from an econometrical analysis based on card-based transaction data in some embodiments of the MMCDB, e.g., a Business Analytics Reporting ("BAR") component;

[0042] FIGURE 35 shows a logic flow diagram illustrating example aspects of sharing an analytical model generated using data acquired using the centralized personal information platform in some embodiments of the MMCDB, e.g., an Analytical Model Sharing ("AMS") component;

[0043] FIGURE 36 shows a logic flow diagram illustrating example aspects of a metadata based interpretation engine of the MMCDB that generates standardized encryptmatics XML from structured data obtained from various sources via the centralized personal information platform, e.g., an Encryptmatics XML Converter ("EXC") component;

[0044] FIGURE 37 shows a data flow diagram illustrating an example email data aggregation procedure, in one embodiment of the MMCDB;

[0045] FIGURE 38 shows a block diagram illustrating an example distributed linking node mesh, in one embodiment of the MMCDB;

[0046] FIGURES 39A-F show a block diagram illustrating an example distributed linking node mesh search, in one embodiment of the MMCDB;

[0047] FIGURES 40A-C show a block diagram illustrating an example distributed linking node mesh index creation, in one embodiment of the MMCDB;

[0048] FIGURE 41 shows a logic flow illustrating an example Encryptmatics XML Converter component, in one embodiment of the MMCDB;

[0049] FIGURE 42 shows a logic flow illustrating input language loading by an Encryptmatics XML Converter component, in one embodiment of the MMCDB;

[0050] FIGURES 43A-B show a logic flow illustrating input model conversion by an Encryptmatics XML Converter component, in one embodiment of the MMCDB;

[0051] FIGURE 44 shows a block diagram illustrating aspects of a tumblar data source manipulation / anonymization component, e.g., a TDS component, in one embodiment of the MMCDB;

[0052] FIGURE 45 shows a logic flow diagram illustrating an example tumblar data source manipulation / anonymization component, in one embodiment of the MMCDB;

[0053] FIGURE 46 shows an example data flow illustrating mesh aggregation and cluster querying, in one embodiment of a MMCDB;

[0054] FIGURE 47 shows an example logic flow illustrating cluster response analysis and transaction triggering, in one embodiment of a MMCDB;

[0055] FIGURE 48A-C illustrate an example MMCDB application embodiment, in one embodiment of the MMCDB; and

[0056] FIGURE 49 shows a block diagram illustrating embodiments of a MMCDB controller.

[0057] The leading number of each reference number within the drawings indicates the figure in which that reference number is introduced and/or detailed. As such, a detailed discussion of reference number 101 would be found and/or introduced in Figure 1. Reference number 201 is introduced in Figure 2, etc. The leading number of each reference number within the drawings indicates the figure in which that reference number is introduced and/or detailed. As such, a detailed discussion of reference number 101 would be found and/or introduced in Figure 1. Reference number 201 is introduced in Figure 2, etc.

DETAILED DESCRIPTION

MMCDB

[0058] The MULTI-SOURCE, MULTI-DIMENSIONAL, CROSS-ENTITY, MULTIMEDIA DATABASE PLATFORM APPARATUSES, METHODS AND SYSTEMS (hereinafter "MMCDB") transform data aggregated from various computer resources, via MMCDB components, into updated entity profiles, social graphs and/or investment recommendations. The MMCDB components, in various embodiments, implement advantageous features as set forth below.

Centralized Personal Information Platform

[0059] FIGURE 1 shows a block diagram illustrating example aspects of a centralized personal information platform in some embodiments of the MMCDB. In various scenarios, originators 111 such as merchants 111b, consumers 111c (including, e.g., social networking sites), account issuers, acquirers 111a, and/or the like, desire to utilize information from payment network systems for enabling various features for consumers, and may provide input for the generation of a centralized personal information platform.

[0060] For all of the input types (e.g., consumer transactions 111b, social network interactions 111d (e.g., emails, reviews, text posts, photos, audio/video/multimedia, conversations, chats, etc.), financial institution activity 111a (e.g., acquirers, authorizations, denials, issuers, fraud detection, etc.), merchant activities 111b (e.g., offers, coupons, redemptions, etc.), and/or the like, the mesh server 105 may aggregate and store such inputs in consolidated database 104b.

[0061] The mesh server aggregation may be achieved by obtaining a feed of financial transactions (e.g., if the mesh server is also a pay network server), by obtaining complete feed access (e.g., firehose feeds), from social networks (e.g., Facebook, Twitter, etc.), using publically available data API's (e.g., Google seach API), and/or the like.

[0062] In one embodiment, the feeds may be obtained via high-bandwidth network connections. An example of the high-bandwidth network connections may include multiple optical fiber connections to an Internet backplane such as the multinational Equinix Exchange, New York International Internet eXchange (e.g., "NYIIX"), and/or the like.

[0063] The obtained feeds may be stored in fast storage array servers for processing or access by other processing servers. Examples of the fast storage array servers may include server blades such as those manufactured by Dell Computer (e.g., Dell model M820, M620, and/or the like), having multiple RAID fast SSD drives of type SAS with memory cache of type L1, L2, L3, and/or the like. In another embodiment, the feeds may be stored in a public cloud storage service (e.g., Amazon S3, and/or the like) or private cloud (e.g., OpenStack Storage object and/or OpenStack Storage block storage running on servers such as those described above).

[0064] In one embodiment, the fast storage servers may employ a distributed file system that provides high-throughput access to stored data. Example file systems suitable for this purpose may include the Hadoop Distributed File System (e.g., "HDFS"), Google BigTable, and/or the like. The file system may be implemented substantially as a key/value store or, in other embodiments, as a structured file system containing directories and files. In some embodiments, a hybrid key/value structured file system may be used in order to utilize the capabilities of both a key/value store and a structured file system. In one embodiment, the fast storage array servers may be connected to one or mesh servers (e.g., 105) for feed processing.

[0065] In one embodiment, the mesh servers (e.g., 105) may be server blades such as those described above. In another embodiment, the servers may be virtualized and running on a private virtualization platform such as VMWare ESXi, Xen, OpenStack Compute and/or the like. In still other embodiments, the servers may be virtualized using a publically available cloud service such as Amazon EC2 (e.g., via an Amazon Machine Image / "AMI", and/or the like) or Rackspace (e.g., by providing a machine image such as a VDI or OVA file suitable for creating a virtualized machine).

[0066] The mesh server may generate dictionary short code words for every type of input and associate with that short word with the input (e.g., a MD5 hash, etc. may generate a short word for every type of input, where the resulting short code is unique to each input). This short code to actual data input association, when aggregated, may form the basis of a mesh dictionary. An example of mesh dictionary entry substantially in the following form of XML is:

```
<dictionary_entry>
        {id: "1h65323765gtyuf#uy76355",
         type: email,
         category: {cat1: "food", cat2: "dinner"},
         from_addr: "john.doe@gmail.com",
         to_addr: "jane.doe@gmail.com",
         subject: "Korean BBQ this weekend?",
         dictionary_keywords: "Korean, dinner, nyc",
         content_hash: "7m865323476feeaniiji"}
</dictionary_entry>
```

[0067] Segmented portions, complete dictionaries, and/or updates thereto, may thus be sent *en masse* to mesh analytics clone servers; for example, such update may be done at off-network peak hours set to occur at dynamically and/or at set intervals. This allows the analytics servers to perform analytics operations, and it allows those analytics servers to operate on short codes even without the full underlying backend data being available. In so doing, dictionaries may be analyised using less space than the full underlying raw data would require. Additionally, dictionaries may be distributed between multiple servers. In one embodiment, the dictionaries are replicated across multiple servers and, periodically, synchronized. In one embodiment, any inconstancies in distributed and/or separated dictionaries may be reconciled using demarcation protocol and/or controlled inconsistency reconciliation for replicated data (see D. Barbara H. Garcia-Molina, The case for controlled inconsistency in replicated data," Proc. of the Workshop on Management of Replicated Data, Houston, TX, Nov. 1990; D. Barbara H. Garcia-Molina, The demarcation protocol a technique for maintaining arithmetic constraints in distributed database systems, CS-TR-320-91, Princeton University, April 1991; the contents of both which are herein expressly incorporated by

reference). In one embodiment, dictionaries may defer any analytic operations that require the backend data until when the caching of the dictionary is complete. It should be noted that throughout this disclosure, reference is made to "payment network server" or "pay network server." It should be understood that such server may incorporate mesh servers, and it also contemplates that such mesh servers may include a network of mesh analytics clone servers, clustering node servers, clustering servers, and/or the like.

[0068] Features that entities may desire include application services 112 such as alerts 112a, offers 112c, money transfers 112n, fraud detection 112b, and/or the like. In some embodiments of the MMCDB, such originators may request data to enable application services from a common, secure, centralized information platform including a consolidated, cross-entity profile-graph database 101. For example, the originators may submit complex queries to the MMCDB in a structure format, such as the example below. In this example, the query includes a query to determine a location (e.g., of a user), determine the weather associated with the location, perform analyses on the weather data, and provide an exploded graphical view of the results of the analysis:

```
<int
  Model_id  ="1"
  environment_type="RT"
  meta_data="./fModels/robotExample.meta"
  tumblar_location="./fModels/robotExample.tumblar.location"
  input_format="JSON"
  pmmls="AUTONOMOUS_AGENTS.PMML"
  Model_type ="AUTONOMOUS_AGENTS"
>
<vault  >

<door:LOCATION>
       <lock name="DETERMINE LOCATION"
         inkey="INPUT" inkeyname="lat"
         inkey2="INPUT" inkeyname2="long"
         function="ROUND"
         fnct1-prec="-2"
         function-1="JOIN"
         fnct2-delim=":"
```

```
    tumblar='LAT_LONG.key'
    outkey="TEMP"  outkeyname="location"
    type="STRING"
  />
  <lock name="DETERMINE WEATHER"
    inkey="TEMP" inkeyname="location"
    mesh='MESHRT.RECENTWEATHER'
    mesh-query='HASH'
    outkey="TEMP"  outkeyname="WEATHERDATA"
    type="ARRAY"
  />
  <lock name="EXPLODE DATA"
    inkey="TEMP" inkeyname="WEATHERDATA"
    function="EXPLODE"
    fnct-delim=":"
    outkey="MODELDATA"  outkeystartindex=1
  />
  <lock name="USER SETTINGS"
    inkey="INPUT" inkeyname="USERID"
    mesh='MESHRT.AUTONOMOUSAGENT.SETTINGS'
    mesh-query='HASH'
    outkey="TEMP"  outkeyname="USERSETTINGS"
    type="ARRAY"
  />
  <lock name="EXPLODE USER"
    inkey="TEMP" inkeyname="USERSETTINGS"
    function="EXPLODE"
    fnct-delim=":"
    outkey="USERDATA"  outkeystartindex=1
  />
  <lock name="RUN MODEL"
    inkey="MODELDATA"
    inkey1="USERDATA"
    function="TREE"
    fnc-pmml="AUTONOMOUS_AGENTS.PMML"
    outkey="OUTPUT"  outkeyname="WEATHER"
    type="NUMERIC"
```

```
        />
</door>
</vault>
```

[0069] A non-limiting, example listing of data that the MMCDB may return based on a query is provided below. In this example, a user may log into a website via a computing device. The computing device may provide a IP address, and a timestamp to the MMCDB. In response, the MMCDB may identify a profile of the user from its database, and based on the profile, return potential merchants for offers or coupons:

```
-------------------------------------------------
------------------  Use Case 3 -------------------
-- User log into a website
-- Only IP address, GMT and day of week is passed to Mesh
-- Mesh matches profile based on Affinity Group
-- Mesh returns potential Merchants for offers or coupons based
on temporary model using suppression rules
-------------------------------------------------
-- Test case 1 IP:24:227:206 Hour:9 Day:3
-- Test case 2 IP:148:181:75 Hour:4 Day:5
-------------------------------------------------
-------    AffinityGroup Lookup ------------------
-------------------------------------------------
Look up test case 1
[OrderedDict([('ISACTIVE', 'True'), ('ENTITYKEY',
'24:227:206:3:1'), ('XML', None), ('AFFINITYGROUPNAME',
'24:227:206:3:1'), ('DESCRIPTION', None), ('TYPEOF', None),
('UUID', '5f8df970b9ff11e09ab9270cf67eca90')]),
OrderedDict([('ISACTIVE', 'True'), ('BASEUUID',
'4fbea327b9ff11e094f433b5d7c45677'), ('TOKENENTITYKEY',
'4fbea327b9ff11e094f433b5d7c45677:TOKEN:349:F'), ('BASETYPE',
'MODEL_002_001_00'), ('STATUS', 'ACTIVE'), ('ISSUEDDATE', None),
('WEIGHT', '349'), ('CATEGORY', 'F'), ('DOUBLELINKED', None),
('UUID', '6b6aab39b9ff11e08d850dc270e3ea06')]),
OrderedDict([('ISACTIVE', 'True'), ('BASEUUID',
'4fbea328b9ff11e0a5f833b5d7c45677'), ('TOKENENTITYKEY',
'4fbea328b9ff11e0a5f833b5d7c45677:TOKEN:761:1'), ('BASETYPE',
```

```
'MODEL_003_001_00'), ('STATUS', 'ACTIVE'), ('ISSUEDDATE', None),
('WEIGHT', '761'), ('CATEGORY', '1'), ('DOUBLELINKED', None),
('UUID', '68aaca40b9ff11e0ac799fd4e415d9de')]),
OrderedDict([('ISACTIVE', 'True'), ('BASEUUID',
'4fbea328b9ff11e0a5f833b5d7c45677'), ('TOKENENTITYKEY',
'4fbea328b9ff11e0a5f833b5d7c45677:TOKEN:637:2'), ('BASETYPE',
'MODEL_003_001_00'), ('STATUS', 'ACTIVE'), ('ISSUEDDATE', None),
('WEIGHT', '637'), ('CATEGORY', '2'), ('DOUBLELINKED', None),
('UUID', '6b6d1c38b9ff11e08ce10dc270e3ea06')]),
OrderedDict([('ISACTIVE', 'True'), ('BASEUUID',
'4fbea328b9ff11e0a5f833b5d7c45677'), ('TOKENENTITYKEY',
'4fbea328b9ff11e0a5f833b5d7c45677:TOKEN:444:3'), ('BASETYPE',
'MODEL_003_001_00'), ('STATUS', 'ACTIVE'), ('ISSUEDDATE', None),
('WEIGHT', '444'), ('CATEGORY', '3'), ('DOUBLELINKED', None),
('UUID', '6342aa53b9ff11e0bcdb9fd4e415d9de')]),
OrderedDict([('ISACTIVE', 'True'), ('BASEUUID',
'4fbea328b9ff11e0a5f833b5d7c45677'), ('TOKENENTITYKEY',
'4fbea328b9ff11e0a5f833b5d7c45677:TOKEN:333:4'), ('BASETYPE',
'MODEL_003_001_00'), ('STATUS', 'ACTIVE'), ('ISSUEDDATE', None),
('WEIGHT', '333'), ('CATEGORY', '4'), ('DOUBLELINKED', None),
('UUID', '62bd26a2b9ff11e0bc239fd4e415d9de')]),
OrderedDict([('ISACTIVE', 'True'), ('BASEUUID',
'4fbea328b9ff11e0a5f833b5d7c45677'), ('TOKENENTITYKEY',
'4fbea328b9ff11e0a5f833b5d7c45677:TOKEN:307:5'), ('BASETYPE',
'MODEL_003_001_00'), ('STATUS', 'ACTIVE'), ('ISSUEDDATE', None),
('WEIGHT', '307'), ('CATEGORY', '5'), ('DOUBLELINKED', None),
('UUID', '6b6d1c39b9ff11e0986c0dc270e3ea06')]),
OrderedDict([('ISACTIVE', 'True'), ('BASEUUID',
'4fbea32db9ff11e09f3e33b5d7c45677'), ('TOKENENTITYKEY',
'4fbea32db9ff11e09f3e33b5d7c45677:TOKEN:801:Spend'), ('BASETYPE',
'MODEL_008_001_00'), ('STATUS', 'ACTIVE'), ('ISSUEDDATE', None),
('WEIGHT', '801'), ('CATEGORY', 'Spend'), ('DOUBLELINKED', None),
('UUID', '6b6d1c3ab9ff11e0a4ec0dc270e3ea06')]),
OrderedDict([('ISACTIVE', 'True'), ('BASEUUID',
'4fbea32eb9ff11e0b55133b5d7c45677'), ('TOKENENTITYKEY',
'4fbea32eb9ff11e0b55133b5d7c45677:TOKEN:1:Volume'), ('BASETYPE',
'MODEL_009_001_00'), ('STATUS', 'ACTIVE'), ('ISSUEDDATE', None),
```

```
('WEIGHT', '1'), ('CATEGORY', 'Volume'), ('DOUBLELINKED', None),
('UUID', '62a09df3b9ff11e090d79fd4e415d9de')])]
Found a direct match
148:181:75:1:2
-- Failed to find a direct match
-- Try again with only IP address and hour
[OrderedDict([('ISACTIVE', 'True'), ('ENTITYKEY',
'148:181:75:1:1'), ('XML', None), ('AFFINITYGROUPNAME',
'148:181:75:1:1'), ('DESCRIPTION', None), ('TYPEOF', None)])]
-- Found match for case 2
------------------------------------------------------------
------------------ Temporary model rules-------------------
------------------------------------------------------------
{1: {'LOWER': 10, 'BASETYPE': ['MODEL_002_001_00',
'MODEL_003_001_00'], 'attribute': 'WEIGHT', 'rule': 'NEAR', 'OP':
'PROX', 'type': 'TOKENENTITY', 'HIGHER': 10}, 2: {'type':
['MERCHANT'], 'rule': 'FOLLOW'}, 3: {'rule': 'RESTRICTSUBTYPE',
'BASETYPE': ['MODEL_002_001_00', 'MODEL_003_001_00']}}
------------------------------------------------------------
------------------ Temporary Model Output------------------
------------------- For Use Case 1    ---------------------
------------------------------------------------------------
-- Number of Nodes:102
_________LIVRARIASICILIAN
___________________GDPCOLTD
_______GOODWILLINDUSTRIES
_________________DISCOUNTDE
_______________BARELANCHOE
____________BLOOMINGDALES
__________PARCWORLDTENNIS
_________STRIDERITEOUTLET
_________________PARCCEANOR
_________________PONTOFRIO
_____________FNACPAULISTA
_________________FINISHLINE
___________WALMARTCENTRAL
_________BESNIINTERLARGOS
```

1 PARCLOJASCOLOMBO
2 SHOPTIMEINTER
3 BEDBATHBEYOND
4 MACYSWEST
5 PARCRIACHUELOFILIAL
6 JCPENNEYCORPINC
7 PARCLOJASRENNERFL
8 PARCPAQUETAESPORTES
9 MARISALJ
10 PARCLEADERMAGAZINE
11 INTERFLORA
12 DECATHLON
13 PERNAMBUCANASFL
14 KARSTADTDE
15 PARCCEAMCO
16 CHAMPS
17 ACCESSORIZE
18 BLOOMINGDALESDVRS
19 PARCLIVRARIACULTURA
20 PARCCEALOJA
21 ARQUIBANCADA
22 KITBAG
23 FREDERICKSOFHLWD
24 WALMART
25 PARCLOJASINSINUANTE
26 WALMARTCONTAGEM
27 FOOTLOCKER
28 PARCSANTALOLLA
29 RICARDOELETRO
30 PARCPONTOFRIO
31 DOTPAYPLPOLSKA
32 CAMICADO
33 KARSTADT
34 PARCRAMSONS
35 PARCGREGORY
36 GREMIOFBPA
37 WALMARTSJC

1 _____PRODIRECTSOCCERLTD
2 _____LAVIEENROSE
3 _____PARCMARISALJ
4 _____ORDERS
5 _____PARCNSNNATALNORTE
6 _____LOJASINSINUANTE
7 _____B
8 _____CITYCOUNTY
9 _____WALMARTPACAEMBU
10 _____SOHO
11 _____WALMARTOSASCO
12 _____FOSSILSTORESIINC
13 _____MENARDSCLIO
14 _____PARCPEQUENTE
15 _____BEALLS
16 _____THEHOMEDEPOT
17 _____VIAMIA
18 _____PARCLOJASRIACHUELO
19 _____PARCLOJASMILANO
20 _____NORDSTROM
21 _____WAILANACOFFEEHOUSE
22 _____LANCHOEBELLA
23 _____PUKET
24 _____WALMARTSTORESINC
25 _____PARCPERNAMBUCANASFL
26 _____SMARTSHOPPER
27 _____PARCMAGAZINELUIZASP
28 _____COLUMBIASPORTSWEARCO
29 _____BARELANCESTADA
30 _____DONATEEBAY
31 _____PARCRICARDOELETRO
32 _____PARCDISANTINNI
33 _____SCHUHCOUK
34 _____CEANOR
35 _____PARCCAMICADO
36 _____PARCCENTAUROCE
37 _____PARCMARLUIJOIAS

```
                    ALBADAH
                   MARTINEZ
            MONEYBOOKERSLTD
                      MACYS
              PARCRIOCENTER
             PARCCASASBAHIA
          PARCSUBMARINOLOJA
                       INC
              SUBMARINOLOJA
             LOJASRENNERFL
            RIACHUELOFILIAL
            PARCSONHODOSPES
                  PINKBIJU
                 PARCCEAMRB
-------------------------------------------------------
---------- Temporary model Output ----------
---------- For Use Case 2    ----------
-------------------------------------------------------
-- Number of Nodes:3
                  KITBAG
      COLUMBIASPORTSWEARCO
               GREMIOFBPA
-------------------------------------------------------
--------          End of Example Use Case          ---
-------------------------------------------------------
```

[0070] In some embodiments, the MMCDB may provide access to information on a need-to-know basis to ensure the security of data of entities on which the MMCDB stores information. Thus, in some embodiments, access to information from the centralized platform may be restricted based on the originator as well as application services for which the data is requested. In some embodiments, the MMCDB may thus allow a variety of flexible application services to be built on a common database infrastructure, while preserving the integrity, security, and accuracy of entity data. In some implementations, the MMCDB may generate, update, maintain, store and/or provide profile information on entities, as well as a social graph that maintains and

updates interrelationships between each of the entities stored within the MMCDB. For example, the MMCDB may store profile information on an issuer bank 102a (see profile 103a), a acquirer bank 102b (see profile 103b), a consumer 102c (see profile 103c), a user 102d (see profile 103d), a merchant 102e (see profile 103e), a second merchant 102f (see profile 103f). The MMCDB may also store relationships between such entities. For example, the MMCDB may store information on a relationship of the issuer bank 102a to the consumer 102c shopping at merchant 102e, who in turn may be related to user 102d, who might bank at the back 102b that serves as acquirer for merchant 102f.

[0071] FIGURES 2A-F show block diagrams illustrating example aspects of data models within a centralized personal information platform in some embodiments of the MMCDB. In various embodiments, the MMCDB may store a variety of attributes of entities according to various data models. A few non-limiting example data models are provided below. In some embodiments, the MMCDB may store user profile attributes. For example, a user profile model may store user identifying information 201, user aliases 202, email addresses 203, phone numbers 204, addresses 205, email address types 206, address types 207, user alias types 208, notification statuses 209, ISO country 210, phone number types 211, contract information with the MMCDB 212, user authorization status 213, user profile status 214, security answer 215, security questions 216, language 217, time zone 218, and/or the like, each of the above field types including one or more fields and field values. As another example, a user financial attributes model may store user identifying information 220, user financial account information 221, account contract information 222, user financial account role 223, financial account type 224, financial account identifying information 225, contract information 226, financial account validation 227, financial account validation type 228, and/or the like. As another example, a user payment card attributes data model may include field types such s, but not limited to: user identifying information 230, user financial account information 231, user financial account role 232, account consumer applications 233, user consumer application 234, financial account type 235, financial account validation type 236, financial account information 237, consumer application information 238, consumer application provider information 239, and/or the like. As another example, a user services attributes data model may include field types such as, but not limited to:

user identifying information 240, user alias 241, consumer application user alias status 242, user alias status 243, status change reason code 244, user contract 245, contract information 246, user service attribute value 247, consumer application attributes 248, account service attribute value, account contract 250, user profile status 261, contract business role 252, contract business 253, client information 254, contract role 255, consumer application 256, user activity audit 257, login results 258, and/or the like. As another example, a user services usage attributes data model may include field types such as, but not limited to: user identifying information 260, user alias 261, consumer application user alias status 262, status change reason code 263, user alias status 264, user consumer application 265, user login audit 266, login result 267, account service attribute value 268, account consumer application 269, consumer application 270, consumer application provider 271, login result 272, and/or the like. As another example, a user graph attributes data model may include field types such as, but not limited to: user identifying information 280, user contact 281, consumer application user alias status 282, relationship 283, and/or the like. In some embodiments, the MMCDB may store each object (e.g., user, merchant, issuer, acquirer, IP address, household, etc.) as a node in graph database, and store data with respect to each node in a format such as the example format provided below:

```
<Nodes Data>
ID,Nodes,Label
2fdc7e3fbd1c11e0be645528b00e8d0e,2fdc7e3fbd1c11e0be645528b00e8d0e
,AFFINITYGROUPNAME:49:95:0:3:1
32b1d53ebd1c11e094172557fb829fdf,32b1d53ebd1c11e094172557fb829fdf
,TOKENENTITYKEY:2b8494f0bd1c11e09c856d888c43f7c2:TOKEN:0:F
2e6381e4bd1c11e0b9ffc929a54bb0fd,2e6381e4bd1c11e0b9ffc929a54bb0fd
,MERCHANTNAME:_____MERCHANT_ABC
2fdc7e3dbd1c11e0a22d5528b00e8d0e,2fdc7e3dbd1c11e0a22d5528b00e8d0e
,AFFINITYGROUPNAME:49:95:0:1:1
2e6381e7bd1c11e091b7c929a54bb0fd,2e6381e7bd1c11e091b7c929a54bb0fd
,MERCHANTNAME:_____MERCHANT_XYZ
2cf8cbabbd1c11e0894a5de4f9281135,2cf8cbabbd1c11e0894a5de4f9281135
,USERNAME:000060FF6557F103
```

```
2e6381debd1c11e0b336c929a54bb0fd,2e6381debd1c11e0b336c929a54bb0fd
,MERCHANTNAME:_____MERCHANT_123
2e6381e0bd1c11e0b4e8c929a54bb0fd,2e6381e0bd1c11e0b4e8c929a54bb0fd
,MERCHANTNAME:_____MERCHANT_FGH
2cf681c1bd1c11e0b8815de4f9281135,2cf681c1bd1c11e0b8815de4f9281135
,USERNAME:000030C57080FFE8
2b8494f1bd1c11e0acbd6d888c43f7c2,2b8494f1bd1c11e0acbd6d888c43f7c2
,MODELNAME:MODEL_003_001_00
32b44638bd1c11e0b01c2557fb829fdf,32b44638bd1c11e0b01c2557fb829fdf
,TOKENENTITYKEY:2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:1
2fdc7e40bd1c11e094675528b00e8d0e,2fdc7e40bd1c11e094675528b00e8d0e
,AFFINITYGROUPNAME:49:95:0:4:1
2b8494f0bd1c11e09c856d888c43f7c2,2b8494f0bd1c11e09c856d888c43f7c2
,MODELNAME:MODEL_002_001_00
32b44639bd1c11e0b15b2557fb829fdf,32b44639bd1c11e0b15b2557fb829fdf
,TOKENENTITYKEY:2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:2
32ce84febd1c11e0b0112557fb829fdf,32ce84febd1c11e0b0112557fb829fdf
,TOKENENTITYKEY:2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:4
2e6381e3bd1c11e095b1c929a54bb0fd,2e6381e3bd1c11e095b1c929a54bb0fd
,MERCHANTNAME:_____MERCHANT_789
34582a87bd1c11e080820167449bc60f,34582a87bd1c11e080820167449bc60f
,TOKENENTITYKEY:2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:778:5
2e6381e5bd1c11e0b62cc929a54bb0fd,2e6381e5bd1c11e0b62cc929a54bb0fd
,MERCHANTNAME:_____MERCHANT_456
2fdc7e3ebd1c11e088b55528b00e8d0e,2fdc7e3ebd1c11e088b55528b00e8d0e
,AFFINITYGROUPNAME:49:95:0:2:1
32c4e80dbd1c11e09e442557fb829fdf,32c4e80dbd1c11e09e442557fb829fdf
,TOKENENTITYKEY:2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:774:5
2e6381e1bd1c11e0bf28c929a54bb0fd,2e6381e1bd1c11e0bf28c929a54bb0fd
,MERCHANTNAME:_____MERCHANT_WER
2cf681b8bd1c11e08be85de4f9281135,2cf681b8bd1c11e08be85de4f9281135
,USERNAME:00002552FC930FF8
2cf8cba8bd1c11e09fbc5de4f9281135,2cf8cba8bd1c11e09fbc5de4f9281135
,USERNAME:0000570FF1B46A24
32b4463abd1c11e0bdaa2557fb829fdf,32b4463abd1c11e0bdaa2557fb829fdf
,TOKENENTITYKEY:2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:3
```

```
2cf8cbaebd1c11e0b6515de4f9281135,2cf8cbaebd1c11e0b6515de4f9281135
,USERNAME:000064A20FF962D4
2e6381e6bd1c11e08087c929a54bb0fd,2e6381e6bd1c11e08087c929a54bb0fd
,MERCHANTNAME:_____MERCHANT_496
2e6381e2bd1c11e0941dc929a54bb0fd,2e6381e2bd1c11e0941dc929a54bb0fd
,MERCHANTNAME:_____MERCHANT_SDF
<Edge Data>Source,Target,Type,label, Weight
32ce84febd1c11e0b0112557fb829fdf,2e6381e6bd1c11e08087c929a54bb0fd
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:4,1
000
2fdc7e3ebd1c11e088b55528b00e8d0e,32ce84febd1c11e0b0112557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:4,1
000
2e6381e2bd1c11e0941dc929a54bb0fd,34582a87bd1c11e080820167449bc60f
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:778:5,77
8
2b8494f1bd1c11e0acbd6d888c43f7c2,34582a87bd1c11e080820167449bc60f
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:778:5,77
8
2e6381e1bd1c11e0bf28c929a54bb0fd,32b44639bd1c11e0b15b2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:2,0
2e6381e0bd1c11e0b4e8c929a54bb0fd,32ce84febd1c11e0b0112557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:4,1
000
32b44639bd1c11e0b15b2557fb829fdf,2e6381e6bd1c11e08087c929a54bb0fd
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:2,0
2e6381e1bd1c11e0bf28c929a54bb0fd,32ce84febd1c11e0b0112557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:4,1
000
2e6381debd1c11e0b336c929a54bb0fd,32ce84febd1c11e0b0112557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:4,1
000
2e6381e3bd1c11e095b1c929a54bb0fd,34582a87bd1c11e080820167449bc60f
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:778:5,77
8
2fdc7e40bd1c11e094675528b00e8d0e,32b44639bd1c11e0b15b2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:2,0
```

```
2b8494f1bd1c11e0acbd6d888c43f7c2,32b4463abd1c11e0bdaa2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:3,0
2e6381e3bd1c11e095b1c929a54bb0fd,32b4463abd1c11e0bdaa2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:3,0
2e6381e3bd1c11e095b1c929a54bb0fd,32b1d53ebd1c11e094172557fb829fdf
,MODEL_002_001_00,2b8494f0bd1c11e09c856d888c43f7c2:TOKEN:0:F,0
2e6381e5bd1c11e0b62cc929a54bb0fd,34582a87bd1c11e080820167449bc60f
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:778:5,77
8
2cf8cbabbd1c11e0894a5de4f9281135,32b44638bd1c11e0b01c2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:1,1
000
2cf681b8bd1c11e08be85de4f9281135,32b1d53ebd1c11e094172557fb829fdf
,MODEL_002_001_00,2b8494f0bd1c11e09c856d888c43f7c2:TOKEN:0:F,0
32b4463abd1c11e0bdaa2557fb829fdf,2e6381e6bd1c11e08087c929a54bb0fd
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:3,0
2e6381debd1c11e0b336c929a54bb0fd,32b44639bd1c11e0b15b2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:2,0
2e6381e1bd1c11e0bf28c929a54bb0fd,32b44638bd1c11e0b01c2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:1,1
000
2e6381e5bd1c11e0b62cc929a54bb0fd,32ce84febd1c11e0b0112557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:4,1
000
2e6381e1bd1c11e0bf28c929a54bb0fd,32b4463abd1c11e0bdaa2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:3,0
2e6381e2bd1c11e0941dc929a54bb0fd,32b44639bd1c11e0b15b2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:2,0
2b8494f1bd1c11e0acbd6d888c43f7c2,32c4e80dbd1c11e09e442557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:774:5,77
4
2e6381e2bd1c11e0941dc929a54bb0fd,32b1d53ebd1c11e094172557fb829fdf
,MODEL_002_001_00,2b8494f0bd1c11e09c856d888c43f7c2:TOKEN:0:F,0
2e6381e4bd1c11e0b9ffc929a54bb0fd,32b4463abd1c11e0bdaa2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:3,0
2fdc7e3fbd1c11e0be645528b00e8d0e,32b4463abd1c11e0bdaa2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:3,0
```

```
2e6381e1bd1c11e0bf28c929a54bb0fd,32b1d53ebd1c11e094172557fb829fdf
,MODEL_002_001_00,2b8494f0bd1c11e09c856d888c43f7c2:TOKEN:0:F,0
2fdc7e40bd1c11e094675528b00e8d0e,32ce84febd1c11e0b0112557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:4,1
000
2cf8cba8bd1c11e09fbc5de4f9281135,32c4e80dbd1c11e09e442557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:774:5,77
4
2e6381e2bd1c11e0941dc929a54bb0fd,32b44638bd1c11e0b01c2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:1,1
000
2e6381e4bd1c11e0b9ffc929a54bb0fd,32b1d53ebd1c11e094172557fb829fdf
,MODEL_002_001_00,2b8494f0bd1c11e09c856d888c43f7c2:TOKEN:0:F,0
2e6381e5bd1c11e0b62cc929a54bb0fd,32b44639bd1c11e0b15b2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:2,0
32b1d53ebd1c11e094172557fb829fdf,2e6381e6bd1c11e08087c929a54bb0fd
,MODEL_002_001_00,2b8494f0bd1c11e09c856d888c43f7c2:TOKEN:0:F,0
2b8494f1bd1c11e0acbd6d888c43f7c2,32b44639bd1c11e0b15b2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:2,0
2e6381e3bd1c11e095b1c929a54bb0fd,32b44638bd1c11e0b01c2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:1,1
000
2fdc7e3dbd1c11e0a22d5528b00e8d0e,32ce84febd1c11e0b0112557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:4,1
000
2cf681c1bd1c11e0b8815de4f9281135,32b44638bd1c11e0b01c2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:1,1
000
2cf681c1bd1c11e0b8815de4f9281135,32b1d53ebd1c11e094172557fb829fdf
,MODEL_002_001_00,2b8494f0bd1c11e09c856d888c43f7c2:TOKEN:0:F,0
2e6381e3bd1c11e095b1c929a54bb0fd,32b44639bd1c11e0b15b2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:2,0
2fdc7e3fbd1c11e0be645528b00e8d0e,32b1d53ebd1c11e094172557fb829fdf
,MODEL_002_001_00,2b8494f0bd1c11e09c856d888c43f7c2:TOKEN:0:F,0
32b44638bd1c11e0b01c2557fb829fdf,2e6381e6bd1c11e08087c929a54bb0fd
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:1,1
000
```

```
2cf8cbaebd1c11e0b6515de4f9281135,32ce84febd1c11e0b0112557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:4,1
000
2e6381e6bd1c11e08087c929a54bb0fd,32b1d53ebd1c11e094172557fb829fdf
,MODEL_002_001_00,2b8494f0bd1c11e09c856d888c43f7c2:TOKEN:0:F,0
2e6381e7bd1c11e091b7c929a54bb0fd,34582a87bd1c11e080820167449bc60f
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:778:5,77
8
2e6381e1bd1c11e0bf28c929a54bb0fd,34582a87bd1c11e080820167449bc60f
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:778:5,77
8
2e6381e5bd1c11e0b62cc929a54bb0fd,32b1d53ebd1c11e094172557fb829fdf
,MODEL_002_001_00,2b8494f0bd1c11e09c856d888c43f7c2:TOKEN:0:F,0
2b8494f0bd1c11e09c856d888c43f7c2,32b1d53ebd1c11e094172557fb829fdf
,MODEL_002_001_00,2b8494f0bd1c11e09c856d888c43f7c2:TOKEN:0:F,0
2b8494f1bd1c11e0acbd6d888c43f7c2,32b44638bd1c11e0b01c2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:1,1
000
2e6381e6bd1c11e08087c929a54bb0fd,32b4463abd1c11e0bdaa2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:3,0
2b8494f1bd1c11e0acbd6d888c43f7c2,32ce84febd1c11e0b0112557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:4,1
000
2cf681c1bd1c11e0b8815de4f9281135,32b44639bd1c11e0b15b2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:2,0
2cf681c1bd1c11e0b8815de4f9281135,32b4463abd1c11e0bdaa2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:3,0
2e6381e2bd1c11e0941dc929a54bb0fd,32b4463abd1c11e0bdaa2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:3,0
2e6381e3bd1c11e095b1c929a54bb0fd,32ce84febd1c11e0b0112557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:4,1
000
2e6381e6bd1c11e08087c929a54bb0fd,32ce84febd1c11e0b0112557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:4,1
000
```

```
2e6381e6bd1c11e08087c929a54bb0fd,34582a87bd1c11e080820167449bc60f
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:778:5,77
8
2e6381e6bd1c11e08087c929a54bb0fd,32b44638bd1c11e0b01c2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:1,1
000
2fdc7e3ebd1c11e088b55528b00e8d0e,32b44639bd1c11e0b15b2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:2,0
2e6381e5bd1c11e0b62cc929a54bb0fd,32b4463abd1c11e0bdaa2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:3,0
2e6381e4bd1c11e0b9ffc929a54bb0fd,34582a87bd1c11e080820167449bc60f
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:778:5,77
8
2e6381e4bd1c11e0b9ffc929a54bb0fd,32b44638bd1c11e0b01c2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:1,1
000
34582a87bd1c11e080820167449bc60f,2e6381e6bd1c11e08087c929a54bb0fd
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:778:5,77
8
2e6381e6bd1c11e08087c929a54bb0fd,32b44639bd1c11e0b15b2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:2,0
2e6381e5bd1c11e0b62cc929a54bb0fd,32b44638bd1c11e0b01c2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:1,1
000
2fdc7e3fbd1c11e0be645528b00e8d0e,32b44638bd1c11e0b01c2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:1,1
000
2cf681b8bd1c11e08be85de4f9281135,32b44639bd1c11e0b15b2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:2,0
2e6381e4bd1c11e0b9ffc929a54bb0fd,32b44639bd1c11e0b15b2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:2,0
2cf681b8bd1c11e08be85de4f9281135,32b4463abd1c11e0bdaa2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:3,0
2e6381e4bd1c11e0b9ffc929a54bb0fd,32ce84febd1c11e0b0112557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:4,1
000
```

```
2e6381e2bd1c11e0941dc929a54bb0fd,32ce84febd1c11e0b0112557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:4,1
000
2fdc7e3dbd1c11e0a22d5528b00e8d0e,32b44639bd1c11e0b15b2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:0:2,0
2cf681b8bd1c11e08be85de4f9281135,32b44638bd1c11e0b01c2557fb829fdf
,MODEL_003_001_00,2b8494f1bd1c11e0acbd6d888c43f7c2:TOKEN:1000:1,1
000
```

[0072] In alternate examples, the MMCDB may store data in a JavaScript Object Notation ("JSON") format. The stored information may include data regarding the object, such as, but not limited to: commands, attributes, group information, payment information, account information, etc., such as in the example below:

```
{'MERCHANT': {'TYPEOFTYPES': ['MERCHANTS', 'SYNTHETICNETWORKS'],
'FUNCTIONS': {'ENTITYCREATION': 'putNetwork'}
, 'UNIQUEATTIBUTES': ['MERCHANTNAME'],
'TOKENENTITIESRELATIONSHIPS': [], 'ATTRIBUTES': {'MERCHANT': (2,
'STRING', 0, 'VALUE'), 'MERCH_ZIP_CD': (7, 'STRING', 0, 'VALUE'),
'MERCH_NAME': (8, 'STRING', 0, 'VALUE'), 'MERCHANTNAME': (3,
'STRING', 0, 'VALUE'), 'ACQ_CTRY_NUM': (4, 'STRING', 0, 'VALUE'),
'ACQ_PCR': (6, 'STRING', 0, 'VALUE'), 'ACQ_REGION_NUM': (5,
'STRING', 0, 'VALUE'), 'ISACTIVE': (0, 'BOOL', 1, 'VALUE'),
'ENTITYKEY': (1, 'STRING', 0, 'VALUE')}
}
, 'AFFINITYGROUP': {'TYPEOFTYPES': ['AFFINITYGROUPS'],
'FUNCTIONS': {'ENTITYCREATION': 'putNetwork'}
, 'UNIQUEATTIBUTES': ['AFFINITYGROUPNAME'],
'TOKENENTITIESRELATIONSHIPS': [], 'ATTRIBUTES': {'XML': (2,
'STRING', 0, 'VALUE'), 'DESCRIPTION': (4, 'STRING', 0, 'VALUE'),
'ENTITYKEY': (1, 'STRING', 0, 'VALUE'), 'TYPEOF': (5, 'STRING',
0, 'VALUE'), 'AFFINITYGROUPNAME': (3, 'STRING', 0, 'VALUE'),
'ISACTIVE': (0, 'BOOL', 1, 'VALUE')}
}
, 'CASCADINGPAYMENT': {'TYPEOFTYPES': ['CASCADINGPAYMENT'],
'FUNCTIONS': {'ENTITYCREATION': 'putNetwork'}
```

```
, 'UNIQUEATTIBUTES': ['CASCADINGPAYMENTNAME'],
'TOKENENTITIESRELATIONSHIPS': ['GROUP'], 'ATTRIBUTES': {'STATUS':
(2, 'STRING', 0, 'VALUE'), 'EXPDT': (6, 'DATETIME', 0, 'VALUE'),
'GROUP': (3, 'STRING', 0, 'VALUE'), 'RESTRICTIONS': (7, 'DICT',
0, 'VALUE'), 'CASCADINGPAYMENTNAME': (4, 'STRING', 0, 'VALUE'),
'STARTDT': (5, 'DATETIME', 0, 'VALUE'), 'ISACTIVE': (0, 'BOOL',
1, 'VALUE'), 'ENTITYKEY': (1, 'STRING', 0, 'VALUE')}
}
, 'GROUP': {'TYPEOFTYPES': [], 'FUNCTIONS': {'ENTITYCREATION':
'putNetwork'}
, 'UNIQUEATTIBUTES': ['GROUPNAME'], 'TOKENENTITIESRELATIONSHIPS':
{}
, 'ATTRIBUTES': {'GROUPNAME': (2, 'STRING', 0, 'VALUE'),
'DESCRIPTION': (2, 'STRING', 0, 'VALUE'), 'ISACTIVE': (0, 'BOOL',
1, 'VALUE'), 'ENTITYKEY': (1, 'STRING', 0, 'VALUE')}
}
, 'USERS': {'TYPEOFTYPES': [], 'FUNCTIONS': {'ENTITYCREATION':
'putNetwork'}
, 'UNIQUEATTIBUTES': ['USERSID'], 'TOKENENTITIESRELATIONSHIPS':
{}
, 'ATTRIBUTES': {'USERSID': (2, 'STRING', 0, 'VALUE'),
'ISACTIVE': (0, 'BOOL', 1, 'VALUE'), 'ENTITYKEY': (1, 'STRING',
0, 'VALUE')}
}
, 'TWITTERUSER': {'TYPEOFTYPES': ['TOKENENTITY'], 'FUNCTIONS':
{'ENTITYCREATION': 'putWGTNetwork'}
, 'UNIQUEATTIBUTES': ['USERNAME'], 'TOKENENTITIESRELATIONSHIPS':
['USER'], 'ATTRIBUTES': {'USERNAME': (2, 'STRING', 0, 'VALUE'),
'CITY': (5, 'STRING', 0, 'VALUE'), 'ENTITYKEY': (1, 'STRING', 0,
'VALUE'), 'USERLINK': (6, 'STRING', 0, 'VALUE'), 'FULLNAME': (4,
'STRING', 0, 'VALUE'), 'USERTAG': (3, 'STRING', 0, 'VALUE'),
'ISACTIVE': (0, 'BOOL', 1, 'VALUE')}
}
, 'COUPON': {'TYPEOFTYPES': ['COUPON'], 'FUNCTIONS':
{'ENTITYCREATION': 'putNetwork'}
, 'UNIQUEATTIBUTES': ['COUPONNAME'],
'TOKENENTITIESRELATIONSHIPS': ['MERCHANT'], 'ATTRIBUTES':
```

```
{'STATUS': (2, 'STRING', 0, 'VALUE'), 'MERCHANT': (3, 'STRING',
0, 'VALUE'), 'TITLE': (5, 'STRING', 0, 'VALUE'), 'NOTES': (7,
'STRING', 0, 'VALUE'), 'UPDATEDBY': (11, 'STRING', 0, 'VALUE'),
'ENTITYKEY': (1, 'STRING', 0, 'VALUE'), 'DECRIPTION': (6,
'STRING', 0, 'VALUE'), 'CREATEDBY': (10, 'STRING', 0, 'VALUE'),
'LASTUPDATEDT': (9, 'DATETIME', 0, 'VALUE'), 'EXPDT': (13,
'DATETIME', 0, 'VALUE'), 'RESTRICTIONS': (14, 'DICT', 0,
'VALUE'), 'COUPONNAME': (4, 'STRING', 0, 'VALUE'), 'CREATIONDT':
(8, 'DATETIME', 0, 'VALUE'), 'STARTDT': (12, 'DATETIME', 0,
'VALUE'), 'ISACTIVE': (0, 'BOOL', 1, 'VALUE')}
}
, 'MEMBERSHIP': {'TYPEOFTYPES': ['MEMBERSHIPS'], 'FUNCTIONS':
{'ENTITYCREATION': 'putNetwork'}
, 'UNIQUEATTIBUTES': ['MEMBERSHIPNAME'],
'TOKENENTITIESRELATIONSHIPS': ['MERCHANT'], 'ATTRIBUTES':
{'STATUS': (2, 'STRING', 0, 'VALUE'), 'MERCHANT': (3, 'STRING',
0, 'VALUE'), 'RESTRICTIONS': (7, 'DICT', 0, 'VALUE'),
'MEMBERSHIPNAME': (4, 'STRING', 0, 'VALUE'), 'STARTDT': (5,
'DATETIME', 0, 'VALUE'), 'EXPDT': (6, 'DATETIME', 0, 'VALUE'),
'ISACTIVE': (0, 'BOOL', 1, 'VALUE'), 'ENTITYKEY': (1, 'STRING',
0, 'VALUE')}
}
, 'USERSECURITY': {'TYPEOFTYPES': ['SECURITY'], 'FUNCTIONS':
{'ENTITYCREATION': 'putNetwork'}
, 'UNIQUEATTIBUTES': ['USERSECURITYNAME'],
'TOKENENTITIESRELATIONSHIPS': ['USER'], 'ATTRIBUTES': {'STATUS':
(2, 'STRING', 0, 'VALUE'), 'EXPDT': (6, 'DATETIME', 0, 'VALUE'),
'USERSECURITYNAME': (4, 'STRING', 0, 'VALUE'), 'USER': (3,
'STRING', 0, 'VALUE'), 'RESTRICTIONS': (7, 'DICT', 0, 'VALUE'),
'STARTDT': (5, 'DATETIME', 0, 'VALUE'), 'ISACTIVE': (0, 'BOOL',
1, 'VALUE'), 'ENTITYKEY': (1, 'STRING', 0, 'VALUE')}
}
, 'MCC': {'TYPEOFTYPES': ['MCC'], 'FUNCTIONS': {'ENTITYCREATION':
'putWGTNetwork'}
, 'UNIQUEATTIBUTES': ['MCCNAME', 'MCC'],
'TOKENENTITIESRELATIONSHIPS': ['MCCSEG'], 'ATTRIBUTES':
{'MCCSEG': (4, 'STRING', 0, 'VALUE'), 'MCC': (2, 'STRING', 0,
```

```
'VALUE'), 'MCCNAME': (3, 'STRING', 0, 'VALUE'), 'ISACTIVE': (0,
'BOOL', 1, 'VALUE'), 'ENTITYKEY': (1, 'STRING', 0, 'VALUE')}
}
, 'ZIPCODE': {'TYPEOFTYPES': ['LOCATION'], 'FUNCTIONS':
{'ENTITYCREATION': 'putNetwork'}
, 'UNIQUEATTIBUTES': ['ZIPCODE'], 'TOKENENTITIESRELATIONSHIPS':
[], 'ATTRIBUTES': {'STATE': (4, 'STRING', 0, 'VALUE'),
'POPULATION': (3, 'STRING', 0, 'VALUE'), 'ZIPCODE': (2, 'STRING',
0, 'VALUE'), 'ISACTIVE': (0, 'BOOL', 1, 'VALUE'), 'ENTITYKEY':
(1, 'STRING', 0, 'VALUE')}
}
, 'PAYMENTCARD': {'TYPEOFTYPES': ['PAYMENTCARDS'], 'FUNCTIONS':
{'ENTITYCREATION': 'putNetwork'}
, 'UNIQUEATTIBUTES': ['CARDNUMBER'],
'TOKENENTITIESRELATIONSHIPS': ['USER'], 'ATTRIBUTES': {'EXPDATE':
(5, 'DATETIME', 0, 'VALUE'), 'ENTITYKEY': (1, 'STRING', 0,
'VALUE'), 'CARDTYPE': (4, 'STRING', 0, 'VALUE'), 'CARDNUMBER':
(2, 'STRING', 0, 'VALUE'), 'USER': (3, 'STRING', 0, 'VALUE'),
'ISACTIVE': (0, 'BOOL', 1, 'VALUE')}
}
, 'GENERICTOKEN': {'TYPEOFTYPES': ['COUPON'], 'FUNCTIONS':
{'ENTITYCREATION': 'putNetwork'}
, 'UNIQUEATTIBUTES': ['GENERICTOKENNAME'],
'TOKENENTITIESRELATIONSHIPS': ['MERCHANT'], 'ATTRIBUTES':
{'STATUS': (2, 'STRING', 0, 'VALUE'), 'MERCHANT': (3, 'STRING',
0, 'VALUE'), 'TITLE': (5, 'STRING', 0, 'VALUE'), 'NOTES': (7,
'STRING', 0, 'VALUE'), 'UPDATEDBY': (11, 'STRING', 0, 'VALUE'),
'ENTITYKEY': (1, 'STRING', 0, 'VALUE'), 'DECRIPTION': (6,
'STRING', 0, 'VALUE'), 'CREATEDBY': (10, 'STRING', 0, 'VALUE'),
'LASTUPDATEDT': (9, 'DATETIME', 0, 'VALUE'), 'EXPDT': (13,
'DATETIME', 0, 'VALUE'), 'RESTRICTIONS': (14, 'DICT', 0,
'VALUE'), 'STARTDT': (12, 'DATETIME', 0, 'VALUE'), 'CREATIONDT':
(8, 'DATETIME', 0, 'VALUE'), 'GENERICTOKENNAME': (4, 'STRING', 0,
'VALUE'), 'ISACTIVE': (0, 'BOOL', 1, 'VALUE')}
}
, 'USER': {'TYPEOFTYPES': ['USERS', 'SYNTHETICNETWORKS'],
'FUNCTIONS': {'ENTITYCREATION': 'putNetwork'}
```

```
, 'UNIQUEATTIBUTES': ['USERNAME'], 'TOKENENTITIESRELATIONSHIPS':
['USERS'], 'ATTRIBUTES': {'USERNAME': (5, 'STRING', 0, 'VALUE'),
'USERS': (2, 'STRING', 0, 'VALUE'), 'FIRSTNAME': (3, 'STRING', 0,
'VALUE'), 'LASTNAME': (4, 'STRING', 0, 'VALUE'), 'ENTITYKEY': (1,
'STRING', 0, 'VALUE'), 'ISACTIVE': (0, 'BOOL', 1, 'VALUE')}
}
, 'TWEETS': {'TYPEOFTYPES': ['TOKENENTITY'], 'FUNCTIONS':
{'ENTITYCREATION': 'putWGTNetwork'}
, 'UNIQUEATTIBUTES': ['TWEETID'], 'TOKENENTITIESRELATIONSHIPS':
['TWITTERUSER'], 'ATTRIBUTES': {'Title': (4, 'STRING', 0,
'VALUE'), 'RawTweet': (5, 'STRING', 0, 'VALUE'), 'DATETIME': (3,
'STRING', 0, 'VALUE'), 'CLEANEDTWEET': (6, 'STRING', 0, 'VALUE'),
'ENTITYKEY': (1, 'STRING', 0, 'VALUE'), 'TWEETID': (2, 'STRING',
0, 'VALUE'), 'ISACTIVE': (0, 'BOOL', 1, 'VALUE')}
}
, 'MODEL': {'TYPEOFTYPES': ['MODELS'], 'FUNCTIONS':
{'ENTITYCREATION': 'putNetwork'}
, 'UNIQUEATTIBUTES': ['MODELNAME'], 'TOKENENTITIESRELATIONSHIPS':
['USER', 'MERCHANT', 'PAYMENTCARD'], 'ATTRIBUTES': {'XML': (2,
'STRING', 0, 'VALUE'), 'MODELNAME': (3, 'STRING', 0, 'VALUE'),
'DESCRIPTION': (4, 'STRING', 0, 'VALUE'), 'ENTITYKEY': (1,
'STRING', 0, 'VALUE'), 'TYPEOF': (5, 'STRING', 0, 'VALUE'),
'ISACTIVE': (0, 'BOOL', 1, 'VALUE')}
}
, 'MCCSEG': {'TYPEOFTYPES': ['MCCSEG'], 'FUNCTIONS':
{'ENTITYCREATION': 'putWGTNetwork'}
, 'UNIQUEATTIBUTES': ['MCCSEGID'], 'TOKENENTITIESRELATIONSHIPS':
{}
, 'ATTRIBUTES': {'MCCSEGID': (2, 'STRING', 0, 'VALUE'),
'MCCSEGNAME': (3, 'STRING', 0, 'VALUE'), 'ISACTIVE': (0, 'BOOL',
1, 'VALUE'), 'ENTITYKEY': (1, 'STRING', 0, 'VALUE')}
}
, 'TOKENENTITY': {'TYPEOFTYPES': ['TOKENENTITY'], 'FUNCTIONS':
{'ENTITYCREATION': 'putWGTNetwork'}
, 'UNIQUEATTIBUTES': ['TOKENENTITYKEY'],
'TOKENENTITIESRELATIONSHIPS': {}
```

```
, 'ATTRIBUTES': {'STATUS': (4, 'STRING', 0, 'VALUE'),
'ISSUEDDATE': (5, 'STRING', 0, 'VALUE'), 'DOUBLELINKED': (8,
'BOOL', 1, 'VALUE'), 'BASEUUID': (1, 'STRING', 0, 'VALUE'),
'WEIGHT': (6, 'STRING', 0, 'VALUE'), 'BASETYPE': (3, 'STRING', 0,
'VALUE'), 'CATEGORY': (7, 'STRING', 0, 'VALUE'), 'ISACTIVE': (0,
'BOOL', 1, 'VALUE'), 'TOKENENTITYKEY': (2, 'STRING', 0, 'VALUE')}
}
}
```

[0073] FIGURE 3 shows a block diagram illustrating example MMCDB component configurations in some embodiments of the MMCDB. In some embodiments, the MMCDB may aggregate data from a variety of sources to generate centralized personal information. The may also aggregate various types of data in order to generate the centralized personal information. For example, the MMCDB may utilize search results aggregation component(s) 301 (e.g., such as described in FIGS. 4-5) to aggregate search results from across a wide range of computer networked systems, e.g., the Internet. As another example, the MMCDB may utilize transaction data aggregation component(s) 302 (e.g., such as described in FIGS. 6-9) to aggregate transaction data, e.g., from transaction processing procedure by a payment network. As another example, the MMCDB may utilize service usage data aggregation component(s) 303 (e.g., such as described in FIGS. 6-9) to aggregate data on user's usage of various services associated with the MMCDB. As another example, the MMCDB may utilize enrollment data component(s) 304 (e.g., such as described in FIGS. 12-13) to aggregate data on user's enrollment into various services associated with the MMCDB. As another example, the MMCDB may utilize email data component(s) 305a (e.g., such as described in FIG. 37) to aggregate data regarding the user's email correspondence history into various services associated with the MMCDB. As another example, the MMCDB may utilize social data aggregation component(s) 305 (e.g., such as described in FIGS. 10-11) to aggregate data on user's usage of various social networking services accessible by the MMCDB. In one embodiment, the aggregated data may be used to generate dictionary entries. Further detail regarding the generation of dictionary entries may be found throughout this specification, drawings, and claims and particularly with reference to Fig. 1 and Fig. 46.

[0074] In some embodiments, the MMCDB may acquire the aggregated data, and normalize the data into formats that are suitable for uniform storage, indexing, maintenance, and/or further processing via data record normalization component(s) 306 (e.g., such as described in FIGS. 14A-B). The MMCDB may extract data from the normalized data records, and recognize data fields, e.g., the MMCDB may identify the attributes of each field of data included in the normalized data records via data field recognition component(s) 307 (e.g., such as described in FIG. 15). For example, the MMCDB may identify names, user ID(s), addresses, network addresses, comments and/or specific words within the comments, images, blog posts, video, content within the video, and/or the like from the aggregated data. In some embodiments, for each field of data, the MMCDB may classify entity types associated with the field of data, as well as entity identifiers associated with the field of data, e.g., via component(s) 308 (e.g., such as described in FIG. 16). For example, the MMCDB may identify an Internet Protocol (IP) address data field to be associated with a user ID john.q.public (consumer entity type), a user John Q. Public (consumer entity type), a household (the Public household - a multi-consumer entity type / household entity type), a merchant entity type with identifier Acme Merchant Store, Inc. from which purchases are made from the IP address, an Issuer Bank type with identifier First National Bank associated with the purchases made from the IP address, and/or the like. In some embodiments, the MMCDB may utilize the entity types and entity identifiers to correlate entities across each other, e.g., via cross-entity correlation component(s) 309 (e.g., such as described in FIG. 17). For example, the MMCDB may identify, from the aggregated data, that a household entity with identifier H123 may include a user entity with identifier John Q. Public and social identifier john.q.public@facebook.com, a second user entity with identifier Jane P. Doe with social identifier jpdoe@twitter.com, a computer entity with identifier IP address 192.168.4.5, a card account entity with identifier ****1234, a bank issuer entity with identifier AB23145, a merchant entity with identifier Acme Stores, Inc. where the household sub-entities make purchases, and/or the like. In some embodiments, the MMCDB may utilize the entity identifiers, data associated with each entity and/or correlated entities to identify associations to other entities, e.g., via entity attribute association component(s) 310 (e.g., such as described in FIG. 18). For example, the MMCDB may identify specific purchases made via purchase transactions

by members of the household, and thereby identify attributes of members of the household on the basis of the purchases in the purchase transactions made by members of the household. Based on such correlations and associations, the MMCDB may update a profile for each entity identified from the aggregated data, as well as a social graph interrelating the entities identified in the aggregated data, e.g., via entity profile-graph updating component(s) 311 (e.g., such as described in FIGS. 19, 40, 41A-E and 42A-C). In some embodiments, the updating of profile and/or social graphs for an entity may trigger a search for additional data that may be relevant to the newly identified correlations and associations for each entity, e.g., via search term generation component(s) 313-314 (e.g., such as described in FIG. 20). For example, the updating of a profile and/or social graph may trigger searches across the Internet, social networking websites, transaction data from payment networks, services enrolled into and/or utilized by the entities, and/or the like. In some embodiments, such updating of entity profiles and/or social graphs may be performed continuously, periodically, on-demand, and/or the like.

[0075] FIGURE 4 shows a data flow diagram illustrating an example search result aggregation procedure in some embodiments of the MMCDB. In some implementations, the pay network server may obtain a trigger to perform a search. For example, the pay network server may periodically perform a search update of its aggregated search database, e.g., 410, with new information available from a variety of sources, such as the Internet. As another example, a request for on-demand search update may be obtained as a result of a user wishing to enroll in a service, for which the pay network server may facilitate data entry by providing an automated web form filling system using information about the user obtained from the search update. In some implementations, the pay network server may parse the trigger to extract keywords using which to perform an aggregated search. The pay network server may generate a query for application programming interface (API) templates for various search engines (e.g., Google™, Bing®, AskJeeves, market data search engines, etc.) from which to collect data for aggregation. The pay network server may query, e.g., 412, a pay network database, e.g., 407, for search API templates for the search engines. For example, the pay network server may utilize PHP/SQL commands similar to the examples provided

above. The database may provide, e.g., 413, a list of API templates in response. Based on the list of API templates, the pay network server may generate search requests, e.g., 414. The pay network server may issue the generated search requests, e.g., 415a-c, to the search engine servers, e.g., 401a-c. For example, the pay network server may issue PHP commands to request the search engine for search results. An example listing of commands to issue search requests 415a-c, substantially in the form of PHP commands, is provided below:

```
<?PHP
// API URL with access key
$url =
["https://ajax.googleapis.com/ajax/services/search/web?v=1.0&"
    . "q=" $keywords
"&key=12345678909876S4&userip=datagraph.cpip.com"];

// Send Search Request
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_REFERER, "datagraph.cpip.com");
$body = curl_exec($ch);
curl_close($ch);

// Obtain, parse search results
$json = json_decode($body);
?>
```

[0076] In some embodiments, the search engine servers may query, e.g., 417a-c, their search databases, e.g., 402a-c, for search results falling within the scope of the search keywords. In response to the search queries, the search databases may provide search results, e.g., 418a-c, to the search engine servers. The search engine servers may return the search results obtained from the search databases, e.g., 419a-c, to the pay network server making the search requests. An example listing of search results 419a-c, substantially in the form of JavaScript Object Notation (JSON)-formatted data, is provided below:

```
{"responseData": {
 "results": [
  {
   "GsearchResultClass": "GwebSearch",
   "unescapedUrl": "http://en.wikipedia.org/wiki/John_Q_Public",
   "url": "http://en.wikipedia.org/wiki/John_Q_Public",
   "visibleUrl": "en.wikipedia.org",
   "cacheUrl":
"http://www.google.com/search?q\u003dcache:TwrPfhd22hYJ:en.wikipe
dia.org",
   "title": "\u003cb\u003eJohn Q. Public\u003c/b\u003e -
Wikipedia, the free encyclopedia",
   "titleNoFormatting": "John Q. Public - Wikipedia, the free
encyclopedia",
   "content": "\[1\] In 2006, he served as Chief Technology
Officer..."
  },
  {
   "GsearchResultClass": "GwebSearch",
   "unescapedUrl": "http://www.imdb.com/name/nm0385296/",
   "url": "http://www.imdb.com/name/nm0385296/",
   "visibleUrl": "www.imdb.com",
   "cacheUrl":
"http://www.google.com/search?q\u003dcache:1i34KkqnsooJ:www.imdb.
com",
   "title": "\u003cb\u003eJohn Q. Public\u003c/b\u003e",
   "titleNoFormatting": "John Q. Public",
   "content": "Self: Zoolander. Socialite \u003cb\u003eJohn Q.
Public\u003c/b\u003e..."
  },
  ...
 ],
 "cursor": {
  "pages": [
   { "start": "0", "label": 1 },
   { "start": "4", "label": 2 },
   { "start": "8", "label": 3 },
```

```
      { "start": "12","label": 4 }
    ],
    "estimatedResultCount": "59600000",
    "currentPageIndex": 0,
    "moreResultsUrl":
  "http://www.google.com/search?oe\u003dutf8\u0026ie\u003dutf8..."
   }
  }
  , "responseDetails": null, "responseStatus": 200}
```


[0077] In some embodiments, the pay network server may store the aggregated search results, e.g., 420, in an aggregated search database, e.g., 410a.

[0078] FIGURE 5 shows a logic flow diagram illustrating example aspects of aggregating search results in some embodiments of the MMCDB, e.g., a Search Results Aggregation ("SRA") component 500. In some implementations, the pay network server may obtain a trigger to perform a search, e.g., 501. For example, the pay network server may periodically perform a search update of its aggregated search database with new information available from a variety of sources, such as the Internet. As another example, a request for on-demand search update may be obtained as a result of a user wishing to enroll in a service, for which the pay network server may facilitate data entry by providing an automated web form filling system using information about the user obtained from the search update. In some implementations, the pay network server may parse the trigger, e.g., 502, to extract keywords using which to perform an aggregated search. The pay network server may determine the search engines to search, e.g., 503, using the extracted keywords. Then, the pay network server may generate a query for application programming interface (API) templates for the various search engines (e.g., Google™, Bing®, AskJeeves, market data search engines, etc.) from which to collect data for aggregation, e.g., 504. The pay network server may query, e.g., 505, a pay network database for search API templates for the search engines. For example, the pay network server may utilize PHP/SQL commands similar to the examples provided above. The database may provide, e.g., 505, a list of API templates in response. Based on the list of API templates, the pay network server may generate search requests, e.g., 506. The pay network server may issue the generated search requests to the search

engine servers. The search engine servers may parse the obtained search results(s), e.g., 507, and query, e.g., 508, their search databases for search results falling within the scope of the search keywords. In response to the search queries, the search databases may provide search results, e.g., 509, to the search engine servers. The search engine servers may return the search results obtained from the search databases, e.g., 510, to the pay network server making the search requests. The pay network server may generate, e.g., 511, and store the aggregated search results, e.g., 512, in an aggregated search database.

[0079] FIGURES 6A-D show data flow diagrams illustrating an example card-based transaction execution procedure in some embodiments of the MMCDB. In some implementations, a user, e.g., 601, may desire to purchase a product, service, offering, and/or the like ("product"), from a merchant. The user may communicate with a merchant server, e.g., 603, via a client such as, but not limited to: a personal computer, mobile device, television, point-of-sale terminal, kiosk, ATM, and/or the like (e.g., 602). For example, the user may provide user input, e.g., purchase input 611, into the client indicating the user's desire to purchase the product. In various implementations, the user input may include, but not be limited to: keyboard entry, card swipe, activating a RFID/NFC enabled hardware device (e.g., electronic card having multiple accounts, smartphone, tablet, etc.), mouse clicks, depressing buttons on a joystick/game console, voice commands, single/multi-touch gestures on a touch-sensitive interface, touching user interface elements on a touch-sensitive display, and/or the like. For example, the user may direct a browser application executing on the client device to a website of the merchant, and may select a product from the website via clicking on a hyperlink presented to the user via the website. As another example, the client may obtain track 1 data from the user's card (e.g., credit card, debit card, prepaid card, charge card, etc.), such as the example track 1 data provided below:

```
%B123456789012345^PUBLIC/J.Q.^99011200000000000000**901******?*
(wherein '123456789012345' is the card number of 'J.Q. Public'
and has a CVV number of 901. '990112' is a service code, and ***
represents decimal digits which change randomly each time the
card is used.)
```

[0080] In some implementations, the client may generate a purchase order message, e.g., 612, and provide, e.g., 613, the generated purchase order message to the merchant server. For example, a browser application executing on the client may provide, on behalf of the user, a (Secure) Hypertext Transfer Protocol ("HTTP(S)") GET message including the product order details for the merchant server in the form of data formatted according to the eXtensible Markup Language ("XML"). Below is an example HTTP(S) GET message including an XML-formatted purchase order message for the merchant server:

```
GET /purchase.php HTTP/1.1
Host: www.merchant.com
Content-Type: Application/XML
Content-Length: 1306
<?XML version = "1.0" encoding = "UTF-8"?>
<purchase_order>
        <order_ID>4NFU4RG94</order_ID>
        <timestamp>2011-02-22 15:22:43</timestamp>
        <user_ID>john.q.public@gmail.com</user_ID>
        <client_details>
             <client_IP>192.168.23.126</client_IP>
             <client_type>smartphone</client_type>
             <client_model>HTC Hero</client_model>
             <OS>Android 2.2</OS>
             <app_installed_flag>true</app_installed_flag>
        </client_details>
        <purchase_details>
             <num_products>1</num_products>
             <product>
                  <product_type>book</product_type>
                  <product_params>
                       <product_title>XML for
dummies</product_title>
                       <ISBN>938-2-14-168710-0</ISBN>
                       <edition>2nd ed.</edition>
                       <cover>hardbound</cover>
                       <seller>bestbuybooks</seller>
```

```
                    </product_params>
                    <quantity>1</quantity>
               </product>
          </purchase_details>
          <account_params>
               <account_name>John Q. Public</account_name>
               <account_type>credit</account_type>
               <account_num>123456789012345</account_num>
               <billing_address>123 Green St., Norman, OK
98765</billing_address>
               <phone>123-456-7809</phone>
               <sign>/jqp/</sign>
               <confirm_type>email</confirm_type>

          <contact_info>john.q.public@gmail.com</contact_info>
          </account_params>
          <shipping_info>
               <shipping_adress>same as billing</shipping_address>
               <ship_type>expedited</ship_type>
               <ship_carrier>FedEx</ship_carrier>
               <ship_account>123-45-678</ship_account>
               <tracking_flag>true</tracking_flag>
               <sign_flag>false</sign_flag>
          </shipping_info>
</purchase_order>
```

[0081] In some implementations, the merchant server may obtain the purchase order message from the client, and may parse the purchase order message to extract details of the purchase order from the user. The merchant server may generate a card query request, e.g., 614 to determine whether the transaction can be processed. For example, the merchant server may attempt to determine whether the user has sufficient funds to pay for the purchase in a card account provided with the purchase order. The merchant server may provide the generated card query request, e.g., 615, to an acquirer server, e.g., 604. For example, the acquirer server may be a server of an acquirer financial institution ("acquirer") maintaining an account of the merchant. For example,

the proceeds of transactions processed by the merchant may be deposited into an account maintained by the acquirer. In some implementations, the card query request may include details such as, but not limited to: the costs to the user involved in the transaction, card account details of the user, user billing and/or shipping information, and/or the like. For example, the merchant server may provide a HTTP(S) POST message including an XML-formatted card query request similar to the example listing provided below:

```
POST /cardquery.php HTTP/1.1
Host: www.acquirer.com
Content-Type: Application/XML
Content-Length: 624
<?XML version = "1.0" encoding = "UTF-8"?>
<card_query_request>
        <query_ID>VNEI39FK</query_ID>
        <timestamp>2011-02-22 15:22:44</timestamp>
        <purchase_summary>
             <num_products>1</num_products>
             <product>
                  <product_summary>Book - XML for
dummies</product_summary>
                  <product_quantity>1</product_quantity?
             </product>
        </purchase_summary>
        <transaction_cost>$34.78</transaction_cost>
        <account_params>
             <account_name>John Q. Public</account_name>
             <account_type>credit</account_type>
             <account_num>123456789012345</account_num>
             <billing_address>123 Green St., Norman, OK
98765</billing_address>
             <phone>123-456-7809</phone>
             <sign>/jqp/</sign>
        </account_params>
        <merchant_params>
             <merchant_id>3FBCR4INC</merchant_id>
```

```
                <merchant_name>Books & Things, Inc.</merchant_name>

        <merchant_auth_key>1NNF484MCP59CHB27365</merchant_auth_ke
y>
        </merchant_params>
</card_query_request>
```

[0082] In some implementations, the acquirer server may generate a card authorization request, e.g., 616, using the obtained card query request, and provide the card authorization request, e.g., 617, to a pay network server, e.g., 605. For example, the acquirer server may redirect the HTTP(S) POST message in the example above from the merchant server to the pay network server.

[0083] In some implementations, the pay network server may determine whether the user has enrolled in value-added user services. For example, the pay network server may query 618 a database, e.g., pay network database 407, for user service enrollment data. For example, the server may utilize PHP/SQL commands similar to the example provided above to query the pay network database. In some implementations, the database may provide the user service enrollment data, e.g., 619. The user enrollment data may include a flag indicating whether the user is enrolled or not, as well as instructions, data, login URL, login API call template and/or the like for facilitating access of the user-enrolled services. For example, in some implementations, the pay network server may redirect the client to a value-add server (e.g., such as a social network server where the value-add service is related to social networking) by providing a HTTP(S) REDIRECT 300 message, similar to the example below:

```
HTTP/1.1 300 Multiple Choices
Location:
https://www.facebook.com/dialog/oauth?client_id=snpa_app_ID&redir
ect_uri= www.paynetwork.com/purchase.php
<html>
        <head><title>300 Multiple Choices</title></head>
        <body><h1>Multiple Choices</h1></body>
</html>
```

[0084] In some implementations, the pay network server may provide payment information extracted from the card authorization request to the value-add server as part of a value add service request, e.g., 620. For example, the pay network server may provide a HTTP(S) POST message to the value-add server, similar to the example below:

```
POST /valueservices.php HTTP/1.1
Host: www.valueadd.com
Content-Type: Application/XML
Content-Length: 1306
<?XML version = "1.0" encoding = "UTF-8"?>
<service_request>
        <request_ID>4NFU4RG94</order_ID>
        <timestamp>2011-02-22 15:22:43</timestamp>
        <user_ID>john.q.public@gmail.com</user_ID>
        <client_details>
             <client_IP>192.168.23.126</client_IP>
             <client_type>smartphone</client_type>
             <client_model>HTC Hero</client_model>
             <OS>Android 2.2</OS>
             <app_installed_flag>true</app_installed_flag>
        </client_details>
        <account_params>
             <account_name>John Q. Public</account_name>
             <account_type>credit</account_type>
             <account_num>123456789012345</account_num>
             <billing_address>123 Green St., Norman, OK
98765</billing_address>
             <phone>123-456-7809</phone>
             <sign>/jqp/</sign>
             <confirm_type>email</confirm_type>

        <contact_info>john.q.public@gmail.com</contact_info>
        </account_params>
        <!--optional-->
        <merchant>
             <merchant_id>CQN3Y42N</merchant_id>
             <merchant_name>Acme Tech, Inc.</merchant_name>
```

```
                <user_name>john.q.public</user_name>
                <cardlist>
www.acme.com/user/john.q.public/cclist.xml<cardlist>
                <user_account_preference>1 3 2 4 7 6
5<user_account_preference>
        </merchant>
</service_request>
```

[0085] In some implementations, the value-add server may provide a service input request, e.g., 621, to the client. For example, the value-add server may provide a HTML input/login form to the client. The client may display, e.g., 622, the login form for the user. In some implementations, the user may provide login input into the client, e.g., 623, and the client may generate a service input response, e.g., 624, for the value-add server. In some implementations, the value-add server may provide value-add services according to user value-add service enrollment data, user profile, etc., stored on the value-add server, and based on the user service input. Based on the provision of value-add services, the value-add server may generate a value-add service response, e.g., 626, and provide the response to the pay network server. For example, the value-add server may provide a HTTP(S) POST message similar to the example below:

```
POST /serviceresponse.php HTTP/1.1
Host: www.paynet.com
Content-Type: Application/XML
Content-Length: 1306
<?XML version = "1.0" encoding = "UTF-8"?>
<service_response>
        <request_ID>4NFU4RG94</order_ID>
        <timestamp>2011-02-22 15:22:43</timestamp>
        <result>serviced</result>
        <servcode>943528976302-45569-003829-04</servcode>
</service_response>
```

[0086] In some implementations, upon receiving the value-add service response from the value-add server, the pay network server may extract the enrollment service data from the response for addition to a transaction data record. In some

implementations, the pay network server may forward the card authorization request to an appropriate pay network server, e.g., 628, which may parse the card authorization request to extract details of the request. Using the extracted fields and field values, the pay network server may generate a query, e.g., 629, for an issuer server corresponding to the user's card account. For example, the user's card account, the details of which the user may have provided via the client-generated purchase order message, may be linked to an issuer financial institution ("issuer"), such as a banking institution, which issued the card account for the user. An issuer server, e.g., 608a-n, of the issuer may maintain details of the user's card account. In some implementations, a database, e.g., pay network database 607, may store details of the issuer servers and card account numbers associated with the issuer servers. For example, the database may be a relational database responsive to Structured Query Language ("SQL") commands. The pay network server may execute a hypertext preprocessor ("PHP") script including SQL commands to query the database for details of the issuer server. An example PHP/SQL command listing, illustrating substantive aspects of querying the database, is provided below:

```
<?PHP
header('Content-Type: text/plain');
mysql_connect("254.93.179.112",$DBserver,$password); // access
database server
mysql_select_db("ISSUERS.SQL"); // select database table to
search
//create query for issuer server data
$query = "SELECT issuer_name issuer_address issuer_id ip_address
mac_address auth_key port_num security_settings_list FROM
IssuerTable WHERE account_num LIKE '%' $accountnum";
$result = mysql_query($query); // perform the search query
mysql_close("ISSUERS.SQL"); // close database access
?>
```

[0087] In response to obtaining the issuer server query, e.g., 629, the pay network database may provide, e.g., 630, the requested issuer server data to the pay network server. In some implementations, the pay network server may utilize the issuer server

data to generate a forwarding card authorization request, e.g., 631, to redirect the card authorization request from the acquirer server to the issuer server. The pay network server may provide the card authorization request, e.g., 632, to the issuer server. In some implementations, the issuer server, e.g., 608, may parse the card authorization request, and based on the request details may query 633 a database, e.g., user profile database 609, for data of the user's card account. For example, the issuer server may issue PHP/SQL commands similar to the example provided below:

```
<?PHP
header('Content-Type: text/plain');
mysql_connect("254.93.179.112",$DBserver,$password); // access
database server
mysql_select_db("USERS.SQL"); // select database table to search
//create query for user data
$query = "SELECT user_id user_name user_balance account_type FROM
UserTable WHERE account_num LIKE '%' $accountnum";
$result = mysql_query($query); // perform the search query
mysql_close("USERS.SQL"); // close database access
?>
```

[0088] In some implementations, on obtaining the user data, e.g., 634, the issuer server may determine whether the user can pay for the transaction using funds available in the account, e.g., 635. For example, the issuer server may determine whether the user has a sufficient balance remaining in the account, sufficient credit associated with the account, and/or the like. If the issuer server determines that the user can pay for the transaction using the funds available in the account, the server may provide an authorization message, e.g., 636, to the pay network server. For example, the server may provide a HTTP(S) POST message similar to the examples above.

[0089] In some implementations, the pay network server may obtain the authorization message, and parse the message to extract authorization details. Upon determining that the user possesses sufficient funds for the transaction, the pay network server may generate a transaction data record from the card authorization request it received, and store, e.g., 639, the details of the transaction and authorization relating to the transaction in a database, e.g., pay network database 607. For example, the pay

network server may issue PHP/SQL commands similar to the example listing below to store the transaction data in a database:

```
<?PHP
header('Content-Type: text/plain');
mysql_connect("254.92.185.103",$DBserver,$password); // access
database server
mysql_select("TRANSACTIONS.SQL"); // select database to append
mysql_query("INSERT INTO PurchasesTable (timestamp,
purchase_summary_list, num_products, product_summary,
product_quantity, transaction_cost, account_params_list,
account_name, account_type, account_num, billing_addres, zipcode,
phone, sign, merchant_params_list, merchant_id, merchant_name,
merchant_auth_key)
VALUES (time(), $purchase_summary_list, $num_products,
$product_summary, $product_quantity, $transaction_cost,
$account_params_list, $account_name, $account_type, $account_num,
$billing_addres, $zipcode, $phone, $sign, $merchant_params_list,
$merchant_id, $merchant_name, $merchant_auth_key)"); // add data
to table in database
mysql_close("TRANSACTIONS.SQL"); // close connection to database
?>
```

[0090] In some implementations, the pay network server may forward the authorization message, e.g., 640, to the acquirer server, which may in turn forward the authorization message, e.g., 640, to the merchant server. The merchant may obtain the authorization message, and determine from it that the user possesses sufficient funds in the card account to conduct the transaction. The merchant server may add a record of the transaction for the user to a batch of transaction data relating to authorized transactions. For example, the merchant may append the XML data pertaining to the user transaction to an XML data file comprising XML data for transactions that have been authorized for various users, e.g., 641, and store the XML data file, e.g., 642, in a database, e.g., merchant database 604. For example, a batch XML data file may be structured similar to the example XML data structure template provided below:

```
<?XML version = "1.0" encoding = "UTF-8"?>
```

```
<merchant_data>
        <merchant_id>3FBCR4INC</merchant_id>
        <merchant_name>Books & Things, Inc.</merchant_name>
        <merchant_auth_key>1NNF484MCP59CHB27365</merchant_auth_ke
y>
        <account_number>123456789</account_number>
</merchant_data>
<transaction_data>
        <transaction 1>
              ...
        </transaction 1>
        <transaction 2>
              ...
        </transaction 2>
              .
              .
              .
        <transaction n>
              ...
        </transaction n>
</transaction_data>
```

[0091] In some implementations, the server may also generate a purchase receipt, e.g., 643, and provide the purchase receipt to the client. The client may render and display, e.g., 644, the purchase receipt for the user. For example, the client may render a webpage, electronic message, text / SMS message, buffer a voicemail, emit a ring tone, and/or play an audio message, etc., and provide output including, but not limited to: sounds, music, audio, video, images, tactile feedback, vibration alerts (e.g., on vibration-capable client devices such as a smartphone etc.), and/or the like.

[0092] With reference to FIGURE 6C, in some implementations, the merchant server may initiate clearance of a batch of authorized transactions. For example, the merchant server may generate a batch data request, e.g., 645, and provide the request, e.g., 646, to a database, e.g., merchant database 604. For example, the merchant server may utilize PHP/SQL commands similar to the examples provided above to query a

relational database. In response to the batch data request, the database may provide the requested batch data, e.g., 647. The server may generate a batch clearance request, e.g., 648, using the batch data obtained from the database, and provide, e.g., 641, the batch clearance request to an acquirer server, e.g., 610. For example, the merchant server may provide a HTTP(S) POST message including XML-formatted batch data in the message body for the acquirer server. The acquirer server may generate, e.g., 650, a batch payment request using the obtained batch clearance request, and provide the batch payment request to the pay network server, e.g., 651. The pay network server may parse the batch payment request, and extract the transaction data for each transaction stored in the batch payment request, e.g., 652. The pay network server may store the transaction data, e.g., 653, for each transaction in a database, e.g., pay network database 607. For each extracted transaction, the pay network server may query, e.g., 654-655, a database, e.g., pay network database 607, for an address of an issuer server. For example, the pay network server may utilize PHP/SQL commands similar to the examples provided above. The pay network server may generate an individual payment request, e.g., 656, for each transaction for which it has extracted transaction data, and provide the individual payment request, e.g., 657, to the issuer server, e.g., 608. For example, the pay network server may provide a HTTP(S) POST request similar to the example below:

```
POST /requestpay.php HTTP/1.1
Host: www.issuer.com
Content-Type: Application/XML
Content-Length: 788
<?XML version = "1.0" encoding = "UTF-8"?>
<pay_request>
        <request_ID>CNI4ICNW2</request_ID>
        <timestamp>2011-02-22 17:00:01</timestamp>
        <pay_amount>$34.78</pay_amount>
        <account_params>
            <account_name>John Q. Public</account_name>
            <account_type>credit</account_type>
            <account_num>123456789012345</account_num>
```

```
                <billing_address>123 Green St., Norman, OK
98765</billing_address>
                <phone>123-456-7809</phone>
                <sign>/jqp/</sign>
          </account_params>
          <merchant_params>
                <merchant_id>3FBCR4INC</merchant_id>
                <merchant_name>Books & Things, Inc.</merchant_name>

          <merchant_auth_key>1NNF484MCP59CHB27365</merchant_auth_ke
y>
          </merchant_params>
          <purchase_summary>
                <num_products>1</num_products>
                <product>
                      <product_summary>Book - XML for
dummies</product_summary>
                      <product_quantity>1</product_quantity?
                </product>
          </purchase_summary>
</pay_request>
```

[0093] In some implementations, the issuer server may generate a payment command, e.g., 658. For example, the issuer server may issue a command to deduct funds from the user's account (or add a charge to the user's credit card account). The issuer server may issue a payment command, e.g., 659, to a database storing the user's account information, e.g., user profile database 608. The issuer server may provide a funds transfer message, e.g., 660, to the pay network server, which may forward, e.g., 661, the funds transfer message to the acquirer server. An example HTTP(S) POST funds transfer message is provided below:

```
POST /clearance.php HTTP/1.1
Host: www.acquirer.com
Content-Type: Application/XML
Content-Length: 206
<?XML version = "1.0" encoding = "UTF-8"?>
```

```
<deposit_ack>
        <request_ID>CNI4ICNW2</request_ID>
        <clear_flag>true</clear_flag>
        <timestamp>2011-02-22 17:00:02</timestamp>
        <deposit_amount>$34.78</deposit_amount>
</deposit_ack>
```

[0094] In some implementations, the acquirer server may parse the funds transfer message, and correlate the transaction (e.g., using the request_ID field in the example above) to the merchant. The acquirer server may then transfer the funds specified in the funds transfer message to an account of the merchant, e.g., 662.

[0095] FIGURES 7A-E show logic flow diagrams illustrating example aspects of card-based transaction execution, resulting in generation of card-based transaction data and service usage data, in some embodiments of the MMCDB, e.g., a Card-Based Transaction Execution ("CTE") component 700. In some implementations, a user may provide user input, e.g., 701, into a client indicating the user's desire to purchase a product from a merchant. The client may generate a purchase order message, e.g., 702, and provide the generated purchase order message to the merchant server. In some implementations, the merchant server may obtain, e.g., 703, the purchase order message from the client, and may parse the purchase order message to extract details of the purchase order from the user. Example parsers that the merchant client may utilize are discussed further below with reference to FIGURE 49. The merchant may generate a product data query, e.g., 704, for a merchant database, which may in response provide the requested product data, e.g., 705. The merchant server may generate a card query request using the product data, e.g., 704, to determine whether the transaction can be processed. For example, the merchant server may process the transaction only if the user has sufficient funds to pay for the purchase in a card account provided with the purchase order. The merchant server may optionally provide the generated card query request to an acquirer server. The acquirer server may generate a card authorization request using the obtained card query request, and provide the card authorization request to a pay network server.

[0096] In some implementations, the pay network server may determine whether the user has enrolled in value-added user services. For example, the pay network server may query a database, e.g., 707, for user service enrollment data. For example, the server may utilize PHP/SQL commands similar to the example provided above to query the pay network database. In some implementations, the database may provide the user service enrollment data, e.g., 708. The user enrollment data may include a flag indicating whether the user is enrolled or not, as well as instructions, data, login URL, login API call template and/or the like for facilitating access of the user-enrolled services. For example, in some implementations, the pay network server may redirect the client to a value-add server (e.g., such as a social network server where the value-add service is related to social networking) by providing a HTTP(S) REDIRECT 300 message. In some implementations, the pay network server may provide payment information extracted from the card authorization request to the value-add server as part of a value add service request, e.g., 710.

[0097] In some implementations, the value-add server may provide a service input request, e.g., 711, to the client. The client may display, e.g., 712, the input request for the user. In some implementations, the user may provide input into the client, e.g., 713, and the client may generate a service input response for the value-add server. In some implementations, the value-add server may provide value-add services according to user value-add service enrollment data, user profile, etc., stored on the value-add server, and based on the user service input. Based on the provision of value-add services, the value-add server may generate a value-add service response, e.g., 717, and provide the response to the pay network server. In some implementations, upon receiving the value-add service response from the value-add server, the pay network server may extract the enrollment service data from the response for addition to a transaction data record, e.g., 719-720.

[0098] With reference to FIGURE 7B, in some implementations, the pay network server may obtain the card authorization request from the acquirer server, and may parse the card authorization request to extract details of the request, e.g., 720. Using the extracted fields and field values, the pay network server may generate a query, e.g., 721-722, for an issuer server corresponding to the user's card account. In response to

obtaining the issuer server query the pay network database may provide, e.g., 722, the requested issuer server data to the pay network server. In some implementations, the pay network server may utilize the issuer server data to generate a forwarding card authorization request, e.g., 723, to redirect the card authorization request from the acquirer server to the issuer server. The pay network server may provide the card authorization request to the issuer server. In some implementations, the issuer server may parse, e.g., 724, the card authorization request, and based on the request details may query a database, e.g., 725, for data of the user's card account. In response, the database may provide the requested user data. On obtaining the user data, the issuer server may determine whether the user can pay for the transaction using funds available in the account, e.g., 726. For example, the issuer server may determine whether the user has a sufficient balance remaining in the account, sufficient credit associated with the account, and/or the like, but comparing the data from the database with the transaction cost obtained from the card authorization request. If the issuer server determines that the user can pay for the transaction using the funds available in the account, the server may provide an authorization message, e.g., 727, to the pay network server.

[0099] In some implementations, the pay network server may obtain the authorization message, and parse the message to extract authorization details. Upon determining that the user possesses sufficient funds for the transaction (e.g., 730, option "Yes"), the pay network server may extract the transaction card from the authorization message and/or card authorization request, e.g., 733, and generate a transaction data record using the card transaction details. The pay network server may provide the transaction data record for storage, e.g., 734, to a database. In some implementations, the pay network server may forward the authorization message, e.g., 735, to the acquirer server, which may in turn forward the authorization message, e.g., 736, to the merchant server. The merchant may obtain the authorization message, and parse the authorization message o extract its contents, e.g., 737. The merchant server may determine whether the user possesses sufficient funds in the card account to conduct the transaction. If the merchant server determines that the user possess sufficient funds, e.g., 738, option "Yes," the merchant server may add the record of the transaction for the user to a batch of transaction data relating to authorized transactions, e.g., 739-740.

The merchant server may also generate a purchase receipt, e.g., 741, for the user. If the merchant server determines that the user does not possess sufficient funds, e.g., 738, option "No," the merchant server may generate an "authorization fail" message, e.g., 742. The merchant server may provide the purchase receipt or the "authorization fail" message to the client. The client may render and display, e.g., 743, the purchase receipt for the user.

[00100] In some implementations, the merchant server may initiate clearance of a batch of authorized transactions by generating a batch data request, e.g., 744, and providing the request to a database. In response to the batch data request, the database may provide the requested batch data, e.g., 745, to the merchant server. The server may generate a batch clearance request, e.g., 746, using the batch data obtained from the database, and provide the batch clearance request to an acquirer server. The acquirer server may generate, e.g., 748, a batch payment request using the obtained batch clearance request, and provide the batch payment request to a pay network server. The pay network server may parse, e.g., 749, the batch payment request, select a transaction stored within the batch data, e.g., 750, and extract the transaction data for the transaction stored in the batch payment request, e.g., 751. The pay network server may generate a transaction data record, e.g., 752, and store the transaction data, e.g., 753, the transaction in a database. For the extracted transaction, the pay network server may generate an issuer server query, e.g., 754, for an address of an issuer server maintaining the account of the user requesting the transaction. The pay network server may provide the query to a database. In response, the database may provide the issuer server data requested by the pay network server, e.g., 755. The pay network server may generate an individual payment request, e.g., 756, for the transaction for which it has extracted transaction data, and provide the individual payment request to the issuer server using the issuer server data from the database.

[00101] In some implementations, the issuer server may obtain the individual payment request, and parse, e.g., 757, the individual payment request to extract details of the request. Based on the extracted data, the issuer server may generate a payment command, e.g., 758. For example, the issuer server may issue a command to deduct funds from the user's account (or add a charge to the user's credit card account). The

issuer server may issue a payment command, e.g., 759, to a database storing the user's account information. In response, the database may update a data record corresponding to the user's account to reflect the debit / charge made to the user's account. The issuer server may provide a funds transfer message, e.g., 760, to the pay network server after the payment command has been executed by the database.

[00102] In some implementations, the pay network server may check whether there are additional transactions in the batch that need to be cleared and funded. If there are additional transactions, e.g., 761, option "Yes," the pay network server may process each transaction according to the procedure described above. The pay network server may generate, e.g., 762, an aggregated funds transfer message reflecting transfer of all transactions in the batch, and provide, e.g., 763, the funds transfer message to the acquirer server. The acquirer server may, in response, transfer the funds specified in the funds transfer message to an account of the merchant, e.g., 764.

[00103] FIGURE 8 shows a data flow diagram illustrating an example procedure to aggregate card-based transaction data in some embodiments of the MMCDB. In some implementations, the pay network server may determine a scope of data aggregation required to perform the analysis, e.g., 811. The pay network server may initiate data aggregation based on the determined scope. The pay network server may generate a query for addresses of server storing transaction data within the determined scope. The pay network server may query, e.g., 812, a pay network database, e.g., 807a, for addresses of pay network servers that may have stored transaction data within the determined scope of the data aggregation. For example, the pay network server may utilize PHP/SQL commands similar to the examples provided above. The database may provide, e.g., 813, a list of server addresses in response to the pay network server's query. Based on the list of server addresses, the pay network server may generate transaction data requests, e.g., 814. The pay network server may issue the generated transaction data requests, e.g., 815a-c, to the other pay network servers, e.g., 805b-d. The other pay network servers may query, e.g., 817a-c, their pay network database, e.g., 807a-d, for transaction data falling within the scope of the transaction data requests. In response to the transaction data queries, the pay network databases may provide transaction data, e.g., 818a-c, to the other pay network servers. The other pay network

servers may return the transaction data obtained from the pay network databases, e.g., 819a-c, to the pay network server making the transaction data requests, e.g., 805a. The pay network server, e.g., 805a, may store the aggregated transaction data, e.g., 820, in an aggregated transactions database, e.g., 810a.

[00104] FIGURE 9 shows a logic flow diagram illustrating example aspects of aggregating card-based transaction data in some embodiments of the MMCDB, e.g., a Transaction Data Aggregation ("TDA") component 900. In some implementations, a pay network server may obtain a trigger to aggregate transaction data, e.g., 901. For example, the server may be configured to initiate transaction data aggregation on a regular, periodic, basis (e.g., hourly, daily, weekly, monthly, quarterly, semi-annually, annually, etc.). As another example, the server may be configured to initiate transaction data aggregation on obtaining information that the U.S. Government (e.g., Department of Commerce, Office of Management and Budget, etc) has released new statistical data related to the U.S. business economy. As another example, the server may be configured to initiate transaction data aggregation on-demand, upon obtaining a user investment strategy analysis request for processing. The pay network server may determine a scope of data aggregation required to perform the analysis, e.g., 902. For example, the scope of data aggregation may be pre-determined. As another example, the scope of data aggregation may be determined based on a received user investment strategy analysis request. The pay network server may initiate data aggregation based on the determined scope. The pay network server may generate a query for addresses of server storing transaction data within the determined scope, e.g., 903. The pay network server may query a database for addresses of pay network servers that may have stored transaction data within the determined scope of the data aggregation. The database may provide, e.g., 904, a list of server addresses in response to the pay network server's query. Based on the list of server addresses, the pay network server may generate transaction data requests, e.g., 905. The pay network server may issue the generated transaction data requests to the other pay network servers. The other pay network servers may obtain and parse the transaction data requests, e.g., 906. Based on parsing the data requests, the other pay network servers may generate transaction data queries, e.g., 907, and provide the transaction data queries to their pay network databases. In

response to the transaction data queries, the pay network databases may provide transaction data, e.g., 908, to the other pay network servers. The other pay network servers may return, e.g., 909, the transaction data obtained from the pay network databases to the pay network server making the transaction data requests. The pay network server may generate aggregated transaction data records from the transaction data received from the other pay network servers, e.g., 910, and store the aggregated transaction data in a database, e.g., 911.

[00105] FIGURE 10 shows a data flow diagram illustrating an example social data aggregation procedure in some embodiments of the MMCDB. In some implementations, the pay network server may obtain a trigger to perform a social data search. For example, the pay network server may periodically perform an update of its aggregated social database, e.g., 1010, with new information available from a variety of sources, such as the social networking services operating on the Internet. As another example, a request for on-demand social data update may be obtained as a result of a user wishing to enroll in a service, for which the pay network server may facilitate data entry by providing an automated web form filling system using information about the user obtained from the social data update. In some implementations, the pay network server may parse the trigger to extract keywords using which to perform an aggregated social data update. The pay network server may generate a query for application programming interface (API) templates for various social networking services (e.g., Facebook®, Twitter™, etc.) from which to collect social data for aggregation. The pay network server may query, e.g., 1012, a pay network database, e.g., 1007, for social network API templates for the social networking services. For example, the pay network server may utilize PHP/SQL commands similar to the examples provided above. The database may provide, e.g., 1013, a list of API templates in response. Based on the list of API templates, the pay network server may generate social data requests, e.g., 1014. The pay network server may issue the generated social data requests, e.g., 1015a-c, to the social network servers, e.g., 1001a-c. For example, the pay network server may issue PHP commands to request the social network servers for social data. An example listing of commands to issue social data requests 1015a-c, substantially in the form of PHP commands, is provided below:

```
<?PHP
header('Content-Type: text/plain');

// Obtain user ID(s) of friends of the logged-in user
$friends =
json_decode(file_get_contents('https://graph.facebook.com/me/frie
nds?access token='$cookie['oauth_access_token']), true);
$friend_ids = array_keys($friends);

// Obtain message feed associated with the profile of the logged-
in user
$feed =
json_decode(file_get_contents('https:llgraph.facebook.com/me/feed
?access_token='$cookie['oauth_access_token']), true);

// Obtain messages by the user's friends
$result = mysql_query('SELECT * FROM content WHERE uid IN ('
.implode($friend_ids, ',') . ')');
$friend_content = array();
while ($row = mysql_fetch_assoc($result))
$friend_content [] $row;
?>
```

[00106] In some embodiments, the social network servers may query, e.g., 1017a-c, their databases, e.g., 1002a-c, for social data results falling within the scope of the social keywords. In response to the queries, the databases may provide social data, e.g., 1018a-c, to the search engine servers. The social network servers may return the social data obtained from the databases, e.g., 1019a-c, to the pay network server making the social data requests. An example listing of social data 1019a-c, substantially in the form of JavaScript Object Notation (JSON)-formatted data, is provided below:

```
[ "data":     [
              {     "name": "Tabatha Orloff",
                    "id": "483722"},
              {     "name": "Darren Kinnaman",
```

```
                    "id": "86S743"},
                {   "name": "Sharron Jutras",
                    "id": "091274"}
  ] }
```

[00107] In some embodiments, the pay network server may store the aggregated search results, e.g., 1020, in an aggregated search database, e.g., 1010a.

[00108] FIGURE 11 shows a logic flow diagram illustrating example aspects of aggregating social data in some embodiments of the MMCDB, e.g., a Social Data Aggregation ("SDA") component 1100. In some implementations, the pay network server may obtain a trigger to perform a social search, e.g., 1101. For example, the pay network server may periodically perform an update of its aggregated social database with new information available from a variety of sources, such as the Internet. As another example, a request for on-demand social data update may be obtained as a result of a user wishing to enroll in a service, for which the pay network server may facilitate data entry by providing an automated web form filling system using information about the user obtained from the social data update. In some implementations, the pay network server may parse the trigger, e.g., 1102, to extract keywords and/or user ID(s) using which to perform an aggregated search for social data. The pay network server may determine the social networking services to search, e.g., 1103, using the extracted keywords and/or user ID(s). Then, the pay network server may generate a query for application programming interface (API) templates for the various social networking services (e.g., Facebook®, Twitter™, etc.) from which to collect social data for aggregation, e.g., 1104. The pay network server may query, e.g., 1105, a pay network database for search API templates for the social networking services. For example, the pay network server may utilize PHP/SQL commands similar to the examples provided above. The database may provide, e.g., 1105, a list of API templates in response. Based on the list of API templates, the pay network server may generate social data requests, e.g., 1106. The pay network server may issue the generated social data requests to the social networking services. The social network servers may parse the obtained search results(s), e.g., 1107, and query, e.g., 1108, their databases for social data falling within the scope of the search keywords. In response to

the social data queries, the databases may provide social data, e.g., 1109, to the social networking servers. The social networking servers may return the social data obtained from the databases, e.g., 1110, to the pay network server making the social data requests. The pay network server may generate, e.g., 1111, and store the aggregated social data, e.g., 1112, in an aggregated social database.

[00109] FIGURE 12 shows a data flow diagram illustrating an example procedure for enrollment in value-add services in some embodiments of the MMCDB. In some implementations, a user, e.g., 1201, may desire to enroll in a value-added service. Let us consider an example wherein the user desires to enroll in social network authenticated purchase payment as a value-added service. It is to be understood that any other value-added service may take the place of the below-described value-added service. The user may communicate with a pay network server, e.g., 1203, via a client such as, but not limited to: a personal computer, mobile device, television, point-of-sale terminal, kiosk, ATM, and/or the like (e.g., 1202). For example, the user may provide user input, e.g., enroll input 1211, into the client indicating the user's desire to enroll in social network authenticated purchase payment. In various implementations, the user input may include, but not be limited to: a single tap (e.g., a one-tap mobile app purchasing embodiment) of a touchscreen interface, keyboard entry, card swipe, activating a RFID/NFC enabled hardware device (e.g., electronic card having multiple accounts, smartphone, tablet, etc.) within the user device, mouse clicks, depressing buttons on a joystick/game console, voice commands, single/multi-touch gestures on a touch-sensitive interface, touching user interface elements on a touch-sensitive display, and/or the like. For example, the user may swipe a payment card at the client 1202. In some implementations, the client may obtain track 1 data from the user's card as enroll input 1211 (e.g., credit card, debit card, prepaid card, charge card, etc.), such as the example track 1 data provided below:

```
%B123456789012345^PUBLIC/J.Q.^99011200000000000000**901******?*
(wherein '123456789012345' is the card number of 'J.Q. Public'
and has a CVV number of 901. '990112' is a service code, and ***
represents decimal digits which change randomly each time the
card is used.)
```

[00110] In some implementations, using the user's input, the client may generate an enrollment request, e.g., 1212, and provide the enrollment request, e.g., 1213, to the pay network server. For example, the client may provide a (Secure) Hypertext Transfer Protocol ("HTTP(S)") POST message including data formatted according to the eXtensible Markup Language ("XML"). Below is an example HTTP(S) POST message including an XML-formatted enrollment request for the pay network server:

```
POST /enroll.php HTTP/1.1
Host: www.merchant.com
Content-Type: Application/XML
Content-Length: 718
<?XML version = "1.0" encoding = "UTF-8"?>
<enrollment_request>
        <cart_ID>4NFU4RG94</order_ID>
        <timestamp>2011-02-22 15:22:43</timestamp>
        <user_ID>john.q.public@gmail.com</user_ID>
        <client_details>
             <client_IP>192.168.23.126</client_IP>
             <client_type>smartphone</client_type>
             <client_model>HTC Hero</client_model>
             <OS>Android 2.2</OS>
             <app_installed_flag>true</app_installed_flag>
        </client_details>
        <!--account_params> <optional>
             <account_name>John Q. Public</account_name>
             <account_type>credit</account_type>
             <account_num>123456789012345</account_num>
             <billing_address>123 Green St., Norman, OK
98765</billing_address>
             <phone>123-456-7809</phone>
             <sign>/jqp/</sign>
             <confirm_type>email</confirm_type>

        <contact_info>john.q.public@gmail.com</contact_info>
        </account_params-->
        <checkout_purchase_details>
             <num_products>1</num_products>
```

```
                <product>
                    <product_type>book</product_type>
                    <product_params>
                        <product_title>XML for
dummies</product_title>
                        <ISBN>938-2-14-168710-0</ISBN>
                        <edition>2nd ed.</edition>
                        <cover>hardbound</cover>
                        <seller>bestbuybooks</seller>
                    </product_params>
                    <quantity>1</quantity>
                </product>
            </checkout_purchase_details>
</enrollment_request>
```

[00111] In some implementations, the pay network server may obtain the enrollment request from the client, and extract the user's payment detail (e.g., XML data) from the enrollment request. For example, the pay network server may utilize a parser such as the example parsers described below in the discussion with reference to FIGURE 49. In some implementations, the pay network server may query, e.g., 1214, a pay network database, e.g., 1204, to obtain a social network request template, e.g., 1215, to process the enrollment request. The social network request template may include instructions, data, login URL, login API call template and/or the like for facilitating social network authentication. For example, the database may be a relational database responsive to Structured Query Language ("SQL") commands. The merchant server may execute a hypertext preprocessor ("PHP") script including SQL commands to query the database for product data. An example PHP/SQL command listing, illustrating substantive aspects of querying the database, e.g., 1214-1215, is provided below:

```
<?PHP
header('Content-Type: text/plain');
mysql_connect("254.93.179.112",$DBserver,$password); // access
database server
mysql_select_db("SOCIALAUTH.SQL"); // select database table to
search
```

```
//create query
$query = "SELECT template FROM EnrollTable WHERE network LIKE '%'
$socialnet";
$result = mysql_query($query); // perform the search query
mysql_close("SOCIALAUTH.SQL"); // close database access
?>
```

[00112] In some implementations, the pay network server may redirect the client to a social network server by providing a HTTP(S) REDIRECT 300 message, similar to the example below:

```
HTTP/1.1 300 Multiple Choices
Location:
https://www.facebook.com/dialog/oauth?client_id=snpa_app_ID&redir
ect_uri= www.paynetwork.com/enroll.php
<html>
        <head><title>300 Multiple Choices</title></head>
        <body><h1>Multiple Choices</h1></body>
</html>
```

[00113] In some implementations, the pay network server may provide payment information extracted from the card authorization request to the social network server as part of a social network authentication enrollment request, e.g., 1217. For example, the pay network server may provide a HTTP(S) POST message to the social network server, similar to the example below:

```
POST /authenticate_enroll.php HTTP/1.1
Host: www.socialnet.com
Content-Type: Application/XML
Content-Length: 1306
<?XML version = "1.0" encoding = "UTF-8"?>
<authenticate_enrollment_request>
        <request_ID>4NFU4RG94</order_ID>
        <timestamp>2011-02-22 15:22:43</timestamp>
        <user_ID>john.q.public@gmail.com</user_ID>
        <client_details>
            <client_IP>192.168.23.126</client_IP>
```

```
                <client_type>smartphone</client_type>
                <client_model>HTC Hero</client_model>
                <OS>Android 2.2</OS>
                <app_installed_flag>true</app_installed_flag>
          </client_details>
          <account_params>
                <account_name>John Q. Public</account_name>
                <account_type>credit</account_type>
                <account_num>123456789012345</account_num>
                <billing_address>123 Green St., Norman, OK
98765</billing_address>
                <phone>123-456-7809</phone>
                <sign>/jqp/</sign>
                <confirm_type>email</confirm_type>
          <contact_info>john.q.public@gmail.com</contact_info>
          </account_params>
</authenticate_enrollment_request>
```

[00114] In some implementations, the social network server may provide a social network login request, e.g., 1218, to the client. For example, the social network server may provide a HTML input form to the client. The client may display, e.g., 1219, the login form for the user. In some implementations, the user may provide login input into the client, e.g., 1220, and the client may generate a social network login response, e.g., 1221, for the social network server. In some implementations, the social network server may authenticate the login credentials of the user, and access payment account information of the user stored within the social network, e.g., in a social network database. Upon authentication, the social network server may generate an authentication data record for the user, e.g., 1222, and provide an enrollment notification, e.g., 1224, to the pay network server. For example, the social network server may provide a HTTP(S) POST message similar to the example below:

```
POST /enrollnotification.php HTTP/1.1
Host: www.paynet.com
Content-Type: Application/XML
Content-Length: 1306
```

```
<?XML version = "1.0" encoding = "UTF-8"?>
<enroll_notification>
        <request_ID>4NFU4RG94</order_ID>
        <timestamp>2011-02-22 15:22:43</timestamp>
        <result>enrolled</result>
</enroll_notification>
```

[00115] Upon receiving notification of enrollment from the social network server, the pay network server may generate, e.g., 1225, a user enrollment data record, and store the enrollment data record in a pay network database, e.g., 1226, to complete enrollment. In some implementations, the enrollment data record may include the information from the enrollment notification 1224.

[00116] FIGURE 13 shows a logic flow diagram illustrating example aspects of enrollment in a value-added service in some embodiments of the MMCDB, e.g., a Value-Add Service Enrollment ("VASE") component 1300. In some implementations, a user, e.g., 1201, may desire to enroll in a value-added service. Let us consider an example wherein the user desires to enroll in social network authenticated purchase payment as a value-added service. It is to be understood that any other value-added service may take the place of the below-described value-added service. The user may communicate with a pay network server via a client. For example, the user may provide user input, e.g., 1301, into the client indicating the user's desire to enroll in social network authenticated purchase payment. In various implementations, the user input may include, but not be limited to: a single tap (e.g., a one-tap mobile app purchasing embodiment) of a touchscreen interface, keyboard entry, card swipe, activating a RFID/NFC enabled hardware device (e.g., electronic card having multiple accounts, smartphone, tablet, etc.) within the user device, mouse clicks, depressing buttons on a joystick/game console, voice commands, single/multi-touch gestures on a touch-sensitive interface, touching user interface elements on a touch-sensitive display, and/or the like. In some implementations, using the user's input, the client may generate an enrollment request, e.g., 1302, and provide the enrollment request to the pay network server. In some implementations, the MMCDB may provide an enrollment button which may take the user to an enrollment webpage where account info may be entered

into web form fields. In some implementations, the pay network server may obtain the enrollment request from the client, and extract the user's payment detail from the enrollment request. For example, the pay network server may utilize a parser such as the example parsers described below in the discussion with reference to FIGURE 49. In some implementations, the pay network server may query, e.g., 1304, a pay network database to obtain a social network request template, e.g., 1305, to process the enrollment request. The social network request template may include instructions, data, login URL, login API call template and/or the like for facilitating social network authentication. In some implementations, the pay network server may provide payment information extracted from the card authorization request to the social network server as part of a social network authentication enrollment request, e.g., 1306. In some implementations, the social network server may provide a social network login request, e.g., 1307, to the client. For example, the social network server may provide a HTML input form to the client. The client may display, e.g., 1308, the login form for the user. In some implementations, the user may provide login input into the client, e.g., 1309, and the client may generate a social network login response for the social network server. In some implementations, the social network server may authenticate the login credentials of the user, and access payment account information of the user stored within the social network, e.g., in a social network database. Upon authentication, the social network server may generate an authentication data record for the user, e.g., 1311, and provide an enrollment notification to the pay network server, e.g., 1313. Upon receiving notification of enrollment from the social network server, the pay network server may generate, e.g., 1314, a user enrollment data record, and store the enrollment data record in a pay network database, e.g., 1315, to complete enrollment. The pay network server may provide an enrollment confirmation, and provide the enrollment confirmation to the client, which may display, e.g., 1317, the confirmation for the user.

[00117] FIGURES 14A-B show flow diagrams illustrating example aspects of normalizing aggregated search, enrolled, service usage, transaction and/or other aggregated data into a standardized data format in some embodiments of the MMCDB, e.g., a Aggregated Data Record Normalization ("ADRN") component 1400. With reference to FIGURE 14A, in some implementations, a pay network server ("server")

may attempt to convert any aggregated data records stored in an aggregated records database it has access to in a normalized data format. For example, the database may have a transaction data record template with predetermined, standard fields that may store data in pre-defined formats (e.g., long integer / double float / 4 digits of precision, etc.) in a pre-determined data structure. A sample XML transaction data record template is provided below:

```
<?XML version = "1.0" encoding = "UTF-8"?>
<transaction_record>
        <record_ID>00000000</record_ID>
        <norm_flag>false</norm_flag>
        <timestamp>yyyy-mm-dd hh:mm:ss</timestamp>
        <transaction_cost>$0,000,000,00</transaction_cost>
        <merchant_params>
            <merchant_id>00000000</merchant_id>
            <merchant_name>TBD</merchant_name>

        <merchant_auth_key>0000000000000000</merchant_auth_key>
        </merchant_params>
        <merchant_products>
            <num_products>000</num_products>
            <product>
                <product_type>TBD</product_type>
                <product_name>TBD</product_name>
                <class_labels_list>TBD<class_labels_list>
                <product_quantity>000</product_quantity>
                <unit_value>$0,000,000.00</unit_value>
                <sub_total>$0,000,000.00</sub_total>
                <comment>normalized transaction data record
template</comment>
            </product>
        </merchant_products>
        <user_account_params>
            <account_name>JTBD</account_name>
            <account_type>TBD</account_type>
            <account_num>0000000000000000</account_num>
            <billing_line1>TBD</billing_line1>
```

```
                <billing_line2>TBD</billing_line2>
                <zipcode>TBD</zipcode>
                <state>TBD</state>
                <country>TBD</country>
                <phone>00-00-000-000-0000</phone>
                <sign>TBD</sign>
            </user_account_params>
    </transaction_record>
```

[00118] In other embodiments, the transaction data record template may contain integrated logic, regular expressions, executable meta-commands, language commands and/or the like in order to facilitate properly matching aggregated data with the location and format of the data in the template. In some embodiments, the template may contain logic in a non-template language, such as PHP commands being included in an XML file. As such, in one example, a language key may be used by the template (e.g., "php:<command>", "java:<function>", and/or the like). In so doing, the matching template may match a vast array of disparate data formats down into a normalized and standardized format. An example transaction data template record substantially in the form of XML is as follows:

```
    <?XML version = "1.0" encoding = "UTF-8"?>
    <transaction_record>
            <record_ID
               default_value=false_return_error
               match_length=8
               format=integer
               regex_search="(?<=\s|^)\d+(?=\s|$)"
               start_search_offset="50bytes">00000000</record_ID>
            <norm_flag>false</norm_flag>
            <timestamp
               default_value="MySQL:'NOW()'"
               format_after_matching="php:mktime($value);">
                     yyyy-mm-dd hh:mm:ss</timestamp>
            <transaction_cost>$0,000,000,00</transaction_cost>
            <merchant_params>
                  <merch_id>00000000</merch_id>
```

```
                <merch_name>TBD</merch_name>
                <merch_auth_key>0000000000000000</merch_auth_key>
          </merchant_params>
          <merchant_products>
                <num_products
                  min_quantity=1
                  max_quantity=30>000</num_products>
                <product>
                      <product_type
                        from_group="array('BOOK','CD','DVD')">
                          TBD</product_type>
                      <product_name>TBD</product_name>
                      <class_labels_list>TBD<class_labels_list>
                      <product_quantity>000</product_quantity>
                      <unit_value>$0,000,000.00</unit_value>
                      <sub_total>$0,000,000.00</sub_total>
                      <comment>normalized transaction data record
template</comment>
                </product>
          </merchant_products>
          <user_account_params>
                <account_name>JTBD</account_name>
                <account_type>TBD</account_type>
                <account_num>0000000000000000</account_num>
                <billing_line1>TBD</billing_line1>
                <billing_line2>TBD</billing_line2>
                <zipcode>TBD</zipcode>
                <state>TBD</state>
                <country>TBD</country>
                <phone>00-00-000-000-0000</phone>
                <sign>TBD</sign>
          </user_account_params>
</transaction_record>
```

[00119] In some implementations, the server may query a database for a normalized data record template, e.g., 1401. The server may parse the normalized data

record template, e.g., 1402. In some embodiments, the parsing may parse the raw data record (such as using a parser as described herein and with respect to Fig. 49). In other embodiments, the parser may parse a dictionary entry containing a subset of the complete data. Based on parsing the normalized data record template, the server may determine the data fields included in the normalized data record template, and the format of the data stored in the fields of the data record template, e.g., 1403. The server may obtain transaction data records for normalization. The server may query a database, e.g., 1404, for non-normalized records. In one embodiment, no querying is required as the normalization of records may occur in flight (e.g., in real time as data is received). For example, the server may issue PHP/SQL commands to retrieve records that do not have the 'norm_flag' field from the example template above, or those where the value of the 'norm_flag' field is 'false'. Upon obtaining the non-normalized transaction data records, the server may select one of the non-normalized transaction data records, e.g., 1405. The server may parse the non-normalized transaction data record, e.g., 1406, and determine the fields present in the non-normalized transaction data record, e.g., 1407. For example, the server may utilize a procedure similar to one described below with reference to FIGURE 15 The server may compare the fields from the non-normalized transaction data record with the fields extracted from the normalized transaction data record template. For example, the server may determine whether the field identifiers of fields in the non-normalized transaction data record match those of the normalized transaction data record template, (e.g., via a dictionary, thesaurus, etc.), are identical, are synonymous, are related, and/or the like. Based on the comparison, the server may generate a 1:1 mapping between fields of the non-normalized transaction data record match those of the normalized transaction data record template, e.g., 1409. The server may generate a copy of the normalized transaction data record template, e.g., 1410, and populate the fields of the template using values from the non-normalized transaction data record, e.g., 1411. The server may also change the value of the 'norm_flag' field to 'true' in the example above. The server may store the populated record in a database (for example, replacing the original version), e.g., 1412. The server may repeat the above procedure for each non-normalized transaction data record (see e.g., 1413), until all the non-normalized transaction data records have been normalized.

[00120] With reference to FIGURE 14B, in some embodiments, the server may utilize metadata (e.g., easily configurable data) to drive an analytics and rule engine that may convert any structured data into a standardized XML format ("encryptmatics" XML). The encryptmatics XML may then be processed by an encryptmatics engine that is capable of parsing, transforming and analyzing data to generate decisions based on the results of the analysis. Accordingly, in some embodiments, the server may implement a metadata-based interpretation engine that parses structured data, including, but not limited to: web content (see e.g., 1421), graph databases (see e.g., 1422), micro blogs, images or software code (see e.g., 1424), and converts the structured data into commands in the encryptmatics XML file format. For example, the structured data may include, without limitation, software code, images, free text, relational database queries, graph queries, sensory inputs (see e.g., 1423, 1425), and/or the like. A metadata based interpretation engine engine, e.g., 1426, may populate a data/command object, e.g., 1427, based on a given record using configurable metadata, e.g., 1428. The configurable metadata may define an action for a given glyph or keyword contained within a data record. The engine may then process the object to export its data structure as a collection of encryptmatics vaults in a standard encryptmatics XML file format, e.g., 1429. The encryptmatics XML file may then be processed to provide various features by an encryptmatics engine, e.g., 1430.

[00121] In some embodiments, the server may obtain the structured data, and perform a standardization routine using the structured data as input (e.g., including script commands, for illustration). For example, the server may remove extra line breaks, spaces, tab spaces, etc. from the structured data, e.g. 1431. The server may determine and load a metadata library, e.g., 1432, using which the server may parse subroutines or functions within the script, based on the metadata, e.g., 1433-1434. In some embodiments, the server may pre-parse conditional statements based on the metadata, e.g., 1435-1436. The server may also parse data 1437 to populate a data/command object based on the metadata and prior parsing, e.g., 1438. Upon finalizing the data/command object, the server may export 1439 the data/command object as XML in standardized encryptmatics format.

[00122] FIGURE 15 shows a logic flow diagram illustrating example aspects of recognizing data fields in normalized aggregated data records in some embodiments of the MMCDB, e.g., a Data Field Recognition ("DFR") component 1500. In some implementations, a server may recognize the type of data fields included in a data record, e.g, date, address, zipcode, name, user ID, email address, payment account number (PAN), CVV2 numbers, and/or the like. The server may select an unprocessed data record for processing, e.g., 1501. The server may parse the data record rule, and extract data fields from the data record, e.g., 1502. The server may query a database for data field templates, e.g., 1503. For example, the server may compare the format of the fields from the data record to the data record templates to identify a match between one of the data field templates and each field within the data record, thus identifying the type of each field within the data record. In one embodiment, the data field templates may be implemented as a collection of regular expressions, a set of interpreted or compiled language commands that when run against the candidate match return boolean true or false if the candidate matches, and/or the like. The server may thus select an extracted data field from the data record, e.g., 1504. The server may select a data field template for comparison with the selected data field, e.g., 1505, and compare the data field template with the selected data field, e.g., 1506, to determine whether format of extracted data field matches format of data field template, e.g., 1507. If the format of the selected extracted data field matches the format of the data field template, e.g., 1508, option "Yes," the server may assign the type of data field template to the selected data field, e.g., 1509. If the format of the extracted data field does not match the format of the data field template, e.g., 1508, option "No," the server may try another data field template until no more data field templates are available for comparison, see e.g., 1510. If no match is found, the server may assign "unknown" string as the type of the data field, e.g., 1511. The server may store the updated data record in the database, e.g., 1512. The server may perform such data field recognition for each data field in the data record (and also for each data record in the database), see e.g., 1513.

[00123] FIGURE 16 shows a logic flow diagram illustrating example aspects of classifying entity types in some embodiments of the MMCDB, e.g., an Entity Type Classification ("ETC") component 1600. In some implementations, a server may apply

one or more classification labels to each of the data records. For example, the server may classify the data records according to entity type, according to criteria such as, but not limited to: geo-political area, number of items purchased, and/or the like. The server may obtain transactions from a database that are unclassified, e.g., 1601, and obtain rules and labels for classifying the records, e.g., 1602. For example, the database may store classification rules, such as the exemplary illustrative XML-encoded classification rule provided below:

```
<rule>
        <id>PURCHASE_44_45</id>
        <name>Number of purchasers</name>
        <inputs>num_purchasers</inputs>
        <operations>
            <1>label = 'null'</1>
            <2>IF (num_purchasers > 1) label = 'household'</2>
        </operations>
        <outputs>label</outputs>
</rule>
```

[00124] The server may select an unclassified data record for processing, e.g., 1603. The server may also select a classification rule for processing the unclassified data record, e.g., 1604. The server may parse the classification rule, and determine the inputs required for the rule, e.g., 1605. Based on parsing the classification rule, the server may parse the normalized data record template, e.g., 1606, and extract the values for the fields required to be provided as inputs to the classification rule. The server may parse the classification rule, and extract the operations to be performed on the inputs provided for the rule processing, e.g., 1607. Upon determining the operations to be performed, the server may perform the rule-specified operations on the inputs provided for the classification rule, e.g., 1608. In some implementations, the rule may provide threshold values. For example, the rule may specify that if the number of products in the transaction, total value of the transaction, average luxury rating of the products sold in the transaction, etc. may need to cross a threshold in order for the label(s) associated with the rule to be applied to the transaction data record. The server may parse the classification rule to extract any threshold values required for the rule to apply, e.g.,

1609. The server may compare the computed values with the rule thresholds, e.g., 1610. If the rule threshold(s) is crossed, e.g., 1611, option "Yes," the server may apply one or more labels to the transaction data record as specified by the classification rule, e.g., 1612. For example, the server may apply a classification rule to an individual product within the transaction, and/or to the transaction as a whole. In other embodiments, the rule may specify criteria that may be present in the mesh in order to generate a new entity (e.g., to create a deduced concept or deduced entity). For example, if a given set of mesh aggregated data contain references the a keyword iPhone, a rule may specify that "iPhone" is to be created as a deduced node within the mesh. This may be done in a recursive manner, such as when the creation of the meta-concept of an "iPhone" may subsequently be combined with created meta-concepts of "iMac" and "iPod" in order to create a master deduced concept of "Apple Computer", which is thereafter associated with "iPhone," "iMac," and "iPod". In so doing, the rules may allow the mesh, given the aggregated content available as well as inputs (such as category inputs) to automatically create meta-concepts based on rules that are themselves unaware of the concepts. In one embodiment, a rule for the creation of a meta-concept, substantially in the form of XML is:

```
<rule id="create_deduced_concept_5" type="deduced_concept">
  <criteria>
    <number_keyword_references>
      <is type="greater_than" value="50" />
      <isnot type="greater_than" value="500" />
    </number_keyword_references>
  </criteria>
  <if_criteria_met value="create_entity' />
</rule>
```

[00125] In the example above, a new deduced entity may be added to the mesh if the number of other entites referencing a given keyword is greater than 50 but less than 500. In one embodiment, the criteria may be specified as a scalar value as shown above. In other embodiments, the criteria may reference a percentage size of the mesh references (such as greater than 5% but less than 10%). In so doing, entities may be added only when they reach a certain absolute threshold, or alternatively when they

reach a threshold with respect to the mesh itself. In other embodiments, the criteria may be a function (such as a Python procedure) that may be performed in order to determine if a new meta-entity should be created. In such an embodiment, the rule may take advantage of any language features available (e.g., language method/functions) as well as external data sources (such as by querying Wikipedia for the presence of a page describing the candidate meta-concept, performing a Google Search and only creating the meta concept if greater than a given number of results are returned, and/or the like). In one embodiment, deduced entries may be created based on a specified or relative frequence of occurance matches (e.g., keyword matches, transaction occurances, and/or the like) within a certain time quantum (e.g., 5 orders for an item within a day/week/month, 100 tweeks a minute about a topic, and/or the like). Deduced entities may become actual mesh entities (and actual mesh entities may be come deduced entities) through the application of similar rules. For example, if an entity is deduced but subsequently the data aggregation shows a sufficient social media discussion regarding a deduced concept, the concept may be changed from a deduced concept to a mesh concept. In so doing, the mesh can adapt to evolving entities that may initially exist only by virtue of their relationship to other nodes, but may ultimately become concepts that the mesh may assign to actual entities.

[00126] In some implementations, the server may process the transaction data record using each rule (see, e.g., 1613). Once all classification rules have been processed for the transaction record, e.g., 1613, option "No," the server may store the transaction data record in a database, e.g., 1614. The server may perform such processing for each transaction data record until all transaction data records have been classified (see, e.g., 1615).

[00127] FIGURE 17 shows a logic flow diagram illustrating example aspects of identifying cross-entity correlation in some embodiments of the MMCDB, e.g., a Cross-Entity Correlation ("CEC") component 1700. In some implementations, a server may recognize that two entites in the the MMCDB share common or related data fields, e.g, date, address, zipcode, name, user ID, email address, payment account number (PAN), CVV2 numbers, and/or the like, and thus identify the entities as being correlated. The server may select a data record for cross-entity correlation, e.g., 1701. The server may

parse the data record rule, and extract data fields from the data record, e.g., 1702-1703. The server may select an extracted data field from the data record, e.g., 1704, and query a database for other data records having the same data field as the extracted data field, e.g., 1705. From the list of retrieved data records from the database query, the server may select a record for further analysis. The server may identify, e.g., 1707, an entity associated with the retrieved data record, e.g., using the ETC 1600 component discussed above in the description with reference to FIGURE 16. The server may add a data field to the data record obtained for cross-entity correlation specifying the correlation to the retrieved selected data record, e.g., 1708. In some embodiments, the server may utilize each data field in the data record obtained for cross-entity correlation to identify correlated entities, see e.g., 1709. The server may add, once complete, a "correlated" flag to the data record obtained for cross-entity correlation, e.g., 1710, e.g., along with as timestamp specifying the time at which the cross-entity correlation was performed. For example, such a timestamp may be used to determine at a later time whether the data record should be processed again for cross-entity correlation. The server may store the updated data record in a database.

[00128] FIGURE 18 shows a logic flow diagram illustrating example aspects of associating attributes to entities in some embodiments of the MMCDB, e.g., an Entity Attribute Association ("EAA") component 1800. In some implementations, a server may associate attributes to an entity, e.g., if the entity id a person, the server may identify a demographic (e.g., male/female), a spend character, a purchase preferences list, a merchants preference list, and/or the like, based on field values of data fields in data records that are related to the entity. In some implementations, a server may obtain a data record for entity attribute association, e.g., 1801. The server may parse the data record rule, and extract data fields from the data record, e.g., 1802-1803. The server may select an extracted data field from the data record, e.g., 1804, and identify a field value for the selected extracted data field from the data record, e.g., 1805. The server may query a database for demographic data, behavioral data, and/or the like, e.g., 1806, using the field value and field type. In response, the database may provide a list of potential attributes, as well as a confidence level in those attribute associations to the entity, see e.g., 1807. The server may add data fields to the data record obtained for

entity attribute association specifying the potentially associated attributes and their associated confidence levels, e.g., 1808. In some embodiments, the server may utilize each data field in the data record obtained for cross-entity correlation to identify correlated entities, see e.g., 1809. The server may store the updated data record in a database, e.g., 1810.

[00129] FIGURE 19 shows a logic flow diagram illustrating example aspects of updating entity profile-graphs in some embodiments of the MMCDB, e.g., an Entity Profile-Graph Updating ("EPGU") component 1900. In some implementations, a server may generate/update a profile for an entity whose data is stored within the MMCDB. The server may obtain an entity profile record for updating, e.g., 1901. The server may parse the entity profile record, and extract an entity identifier data field from the data record, e.g., 1902. The server may query a database for other data records that are related to the same entity, e.g., 1903, using the value for the entity identifier data field. In response, the database may provide a list of other data records for further processing. The server may select one of the other data records to update the entity profile record, e.g., 1904. The server may parse the data record, and extract all correlations, associations, and new data from the other record, e.g., 1905. The server may compare the correlations, attributes, associations, etc., from the other data record with the correlations, associations and attributes from the entity profile. Based on this comparison, the server may identify any new correlations, associations, etc., and generate an updated entity profile record using the new correlations, associations; flag new correlations, associations for further processing, e.g., 1907. In some embodiments, the server may utilize each data record obtained for updating the entity profile record as well as its social graph (e.g., as given by the correlations and associations for the entity), see e.g., 1909. The server may store the updated entity profile record in a database, e.g., 1908.

[00130] FIGURE 20 shows a logic flow diagram illustrating example aspects of generating search terms for profile-graph updating in some embodiments of the MMCDB, e.g., a Search Term Generation ("STG") component 2000. In some implementations, a server may generate/update a profile for an entity whose data is stored within the MMCDB, by performing search for new data, e.g., across the Internet

and social networking services. The server may obtain an entity profile record for updating, e.g., 2001. The server may parse the entity profile record, and extract data field types and field values from the entity profile record, e.g., 2002. The server may query a database for other data records that are related to the same entity, e.g., 2003, using the values for the extracted data fields. In response, the database may provide a list of other data records for further processing. The server may parse the data records, and extract all correlations, associations, and data from the data records, e.g., 2004. The server may aggregate all the data values from all the records and the entity profile record, e.g., 2005. Based on this, the server may return the aggregated data values as search terms to trigger search processes (see e.g., FIG.3, 301-305), e.g., 2006.

Electronic Virtual Wallet User Interface

[00131] FIGURES 21A-E show user interface diagrams illustrating example features of user interfaces for an electronic virtual wallet in some embodiments of the MMCDB. With reference to FIGURE 21A, in some embodiments, a virtual wallet mobile app, e.g., 2111, executing on a device, e.g., 2100, of a user may include an app interface providing various features for the user. For example, the device may include a camera via which the app may acquire image frames, video data, live video, and/or the like, e.g., 2116. The app may be configured to analyze the incoming data, and search, e.g., 2112, for a product identifier, e.g., 2114, such as barcodes, QR codes and/or the like.

[00132] In some embodiments, the app may be configured to automatically detect, e.g., 2112, the presence of a product identifier within an image or video frame grabbed by the device (e.g., via a webcam, in-built camera, etc.). For example, the app may provide a "hands-free" mode of operation wherein the user may move the device to bring product identifiers within the field of view of the image/video capture mechanism of the device, and the app may perform image/video processing operations to automatically detect the product identifier within the field of view. In some embodiments, the app may overlay cross-hairs, target box, and/or like alignment reference markers, e.g., 2115, so that a user may align the product identifier using the reference markers to facilitate product identifier recognition and interpretation.

[00133] In some embodiments, the detection of a product identifier may trigger various operations to provide products, services, information, etc. for the user. For example, the app may be configured to detect and capture a QR code having embedded merchant and/or product information, and utilize the information extracted from the QR code to process a transaction for purchasing a product from a merchant. As other examples, the app may be configured to provide information on related products, quotes, pricing information, related offers, (other) merchants related to the product identifier, rewards/loyalty points associated with purchasing the product related to the product identifier, analytics on purchasing behavior, alerts on spend tracking, and/or the like.

[00134] In some embodiments, the app may include user interface elements to allow the user to manually search, e.g., 2113, for products (e.g., by name, brand, identifier, etc.). In some embodiments, the app may provide the user with the ability to view prior product identifier captures (see, e.g., 2117a) so that the user may be able to better decide which product identifier the user desires to capture. In some embodiments, the app may include interface elements to allow the user to switch back and forth between the product identification mode and product offer interface display screens (see, e.g., 2117b), so that a user may accurately study deals available to the user before capturing a product identifier. In some embodiments, the user may be provided with information about products, user settings, merchants, offers, etc. in list form (see, e.g., 2117c) so that the user may better understand the user's purchasing options. Various other features may be provided for in the app (see, e.g., 2117d). In some embodiments, the user may desire to cancel product purchasing; the app may provide the user with a user interface element (e.g., 2118) to cancel the product identifier recognition procedure and return to the prior interface screen the user was utilizing.

[00135] With reference to FIGURE 21B, in some embodiments, the app may include an indication of the location (e.g., name of the merchant store, geographical location, information about the aisle within the merchant store, etc.) of the user, e.g., 2121. The app may provide an indication of a pay amount due for the purchase of the product, e.g., 2122. In some embodiments, the app may provide various options for the user to pay the amount for purchasing the product(s). For example, the app may utilize

GPS coordinates associated with the device to determine the merchant store within which the user is present, and direct the user to a website of the merchant. In some embodiments, the app may be configured to make an application programming interface ("API") call to participating merchants to directly facilitate transaction processing for purchases. In some embodiments, a merchant-branded app may be developed with an in-app purchasing mode, which may directly connect the user into the merchant's transaction processing system. For example, the user may choose from a number of cards (e.g., credit cards, debit cards, prepaid cards, etc.) from various card providers, e.g., 2123a. In some embodiments, the app may provide the user the option to pay the purchase amount using funds included in a bank account of the user, e.g., a checking, savings, money market, current account, etc., e.g., 2123b. In some embodiments, the user may have set default options for which card, bank account, etc. to use for the purchase transactions via the app. In some embodiments, such setting of default options may allow the user to initiate the purchase transaction via a single click, tap, swipe, and/or other remedial user input action, e.g., 2123c. In some embodiments, when the user utilizes such an option, the app may utilize the default settings of the user to initiate the purchase transaction. In some embodiments, the app may allow the user to utilize other accounts (e.g., Google™ Checkout, Paypal™ account, etc.) to pay for the purchase transaction, e.g., 2123d. In some embodiments, the app may allow the user to utilize rewards points, airline miles, hotel points, electronic coupons, printed coupons (e.g., by capturing the printed coupons similar to the product identifier) etc., to pay for the purchase transaction, e.g., 2123e. In some embodiments, the app may provide an option to provide express authorization before initiating the purchase transaction, e.g., 2124. In some embodiments, the app may provide a progress indicator provide indication on the progress of the transaction after the user has selected an option to initiate the purchase transaction, e.g., 2125. In some embodiments, the app may provide the user with historical information on the user's prior purchases via the app, e.g., 2127a. In some embodiments, the app may provide the user with an option to share information about the purchase (e.g., via email, SMS, wall posting on Facebook®, tweet on Twitter™, etc.) with other users and/or control information shared with the merchant, acquirer, payment network etc., to process the purchase transaction, e.g., 2127b. In some embodiments, the app may provide the user an option to display the

product identification information captured by the client device (e.g., in order to show a customer service representative at the exit of a store the product information), e.g., 2127c. In some embodiments, the user, app, device and or purchase processing system may encounter an error in the processing. In such scenarios, the user may be able to chat with a customer service representative (e.g., VerifyChat 2127d) to resolve the difficulties in the purchase transaction procedure.

[00136] In some embodiments, the user may select to conduct the transaction using a one-time anonymized credit card number, see e.g., 2123f. For example, the app may utilize a pre-designated anonymized set of card details (see, e.g., "AnonCard1," "AnonCard2"). As another example, the app may generate, e.g., in real-time, a one-time anonymous set of card details to securely complete the purchase transaction (e.g., "Anon It 1X"). In such embodiments, the app may automatically set the user profile settings such that the any personal identifying information of the user will not be provided to the merchant and/or other entities. In some embodiments, the user may be required to enter a user name and password to enable the anonymization features.

[00137] With reference to FIGURE 21C, in some embodiments, the user interface elements of the app may be advantageously arranged to provide the user the ability to process a purchase with customized payment parameters with a minimum number of user inputs applied to the user's device. For example, if the user has a QR pay code, e.g., 2132, within the viewing angle of a camera included in the user's mobile device, the user may activate a user interface element to snap the QR code. In some embodiments, the user may control the field of view of the camera using a user interface zoom element, e.g., 2133. In some embodiments, the user interface may be designed such that the user may touch an image of a QR code displayed on the screen to capture the QR code (see e.g., 2134). For example, the position of the user's touch may be utilized as an input by an image processing module executing on the user's device to process the displayed video frame (and/or adjacent video frames), and extract the QR code from the frame(s) based on the user's input. For example, the user's touch may provide an approximate center point of the QR code. Using this information, the image processing module may be able to better perform an automated QR code image recognition, and accordingly

capture the correct QR code (e.g., if portions of many QR codes are displayed within the video frame(s)) selected by the user for capture and processing.

[00138] In some embodiments, the app may utilize predetermined default settings for a particular merchant, e.g., 2131, to process the purchase based on the QR code (e.g., in response to the user touching an image of a QR code displayed on the screen of the user device). However, if the user wishes to customize the payment parameters, the user may activate a user interface element 2135 (or e.g., press and continue to hold the image of the QR code 2132). Upon doing so, the app may provide a pop-up menu, e.g., 2137, providing a variety of payment customization choices, such as those described with reference to FIGURE 21B. The user may, e.g., drag the user's finger to the appropriate settings the user prefers, and release the user's finger from the touchscreen of the user's mobile device to select the setting for payment processing. In alternate embodiments, the payment settings options, e.g., 2137, and QR capture activation button, e.g., 2136 may be included in the user interface along with a window for capturing the QR code via the mobile device's camera. In alternate embodiments, the user's mobile device may generate a hybrid QR code-payment settings graphic, and the POS terminal (or user's trusted computing device) may capture the entire graphic for payment processing. In some embodiments, the app may provide a user interface element 2138 for the user to minimize the payment options settings user interface elements. In some embodiments, the app may provide additional user interface elements, e.g., 2139, to display previous purchases, data shared about those purchases, purchase receipts (e.g., via barcodes) and customer support options (e.g., VerifyChat).

[00139] With reference to FIGURE 21D, in some embodiments, the user may be able to view and/or modify the user profile and/or settings of the user, e.g., by activating user interface element 2122 (of FIGURE 21B). For example, the user may be able to view/modify a user name (e.g., 2141a-b), account number (e.g., 2142a-b), user security access code (e.g., 2143a-b), user pin (e.g., 2144a-b), user address (e.g., 2145a-b), social security number associated with the user (e.g., 2146a-b), current device GPS location (e.g., 2147a-b), user account of the merchant in whose store the user currently is (e.g., 2148a-b), the user's rewards accounts (e.g., 2149a-b), and/or the like. In some embodiments, the user may be able to select which of the data fields and their associated

values should be transmitted to facilitate the purchase transaction, thus providing enhanced data security for the user. For example, in the example illustration in FIGURE 21D, the user has selected the name 2141a, account number 2142a, security code 2143a, merchant account ID 2148a and rewards account ID 2149a as the fields to be sent as part of the notification to process the purchase transaction. In some embodiments, the user may toggle the fields and/or data values that are sent as part of the notification to process the purchase transactions. In some embodiments, the app may provide multiple screens of data fields and/or associated values stored for the user to select as part of the purchase order transmission. In some embodiments, the app may obtain the GPS location of the user. Based on the GPS location of the user, the app may determine the context of the user (e.g., whether the user is in a store, doctor's office, hospital, postal service office, etc.). Based on the context, the app may present the appropriate fields to the user, from which the user may select fields and/or field values to send as part of the purchase order transmission.

[00140] For example, a user may go to doctor's office and desire to pay the co-pay for doctor's appointment. In addition to basic transactional information such as account number and name, the app may provide the user the ability to select to transfer medical records, health information, which may be provided to the medical provider, insurance company, as well as the transaction processor to reconcile payments between the parties. In some embodiments, the records may be sent in a Health Insurance Portability and Accountability Act (HIPAA)-compliant data format and encrypted, and only the recipients who are authorized to view such records may have appropriate decryption keys to decrypt and view the private user information.

[00141] With reference to FIGURE 21E, in some embodiments, the app executing on the user's device may provide a "VerifyChat" feature for fraud prevention (e.g., by activating UI element 2127d in FIGURE 21B). For example, the MMCDB may detect an unusual and/or suspicious transaction. The MMCDB may utilize the VerifyChat feature to communicate with the user, and verify the authenticity of the originator of the purchase transaction. In various embodiments, the MMCDB may send electronic mail message, text (SMS) messages, Facebook® messages, Twitter™ tweets, text chat, voice chat, video chat (e.g., Apple FaceTime), and/or the like to communicate with the user.

For example, the MMCDB may initiate a video challenge for the user, e.g., 2151a. For example, the user may need to present him/her-self via a video chat, e.g., 2152a. In some embodiments, a customer service representative, e.g., agent 2155a, may manually determine the authenticity of the user using the video of the user. In some embodiments, the MMCDB may utilize face, biometric and/or like recognition (e.g., using pattern classification techniques) to determine the identity of the user, e.g., 2154a. In some embodiments, the app may provide reference marker (e.g., cross-hairs, target box, etc.), e.g., 2153a, so that the user may the video to facilitate the MMCDB's automated recognition of the user. In some embodiments, the user may not have initiated the transaction, e.g., the transaction is fraudulent. In such embodiments, the user may cancel, e.g., 2158a, the challenge. The MMCDB may then cancel the transaction, and/or initiate fraud investigation procedures on behalf of the user. In some embodiments, the app may provide additional user interface elements, e.g., to display previous session 2156a, and provide additional customer support options (e.g., VerifyChat 2157a).

[00142] In some embodiments, the MMCDB may utilize a text challenge procedure to verify the authenticity of the user, e.g., 2151b. For example, the MMCDB may communicate with the user via text chat, SMS messages, electronic mail, Facebook® messages, Twitter™ tweets, and/or the like. The MMCDB may pose a challenge question, e.g., 2152b, for the user. The app may provide a user input interface element(s) (e.g., virtual keyboard 2153b) to answer the challenge question posed by the MMCDB. In some embodiments, the challenge question may randomly selected by the MMCDB automatically; in some embodiments, a customer service representative 2155b may manually communicate with the user. In some embodiments, the user may not have initiated the transaction, e.g., the transaction is fraudulent. In such embodiments, the user may cancel, e.g., 2158b, the text challenge. The MMCDB may then cancel the transaction, and/or initiate fraud investigation procedures on behalf of the user. In some embodiments, the app may provide additional user interface elements, e.g., to display previous session 2156b, and provide additional customer support options (e.g., VerifyChat 2157b).

Merchant Analytics Platform

[00143] FIGURE 22 shows a block diagram illustrating example aspects of a merchant analytics platform in first set of embodiments of the MMCDB. In some implementations, a user, e.g., 2201, may desire to purchase products from a merchant. For example, the user may utilize a card (e.g., a credit card, debit, card, prepaid card, charge card, etc.) to purchase products, services, and/or other offerings ("products") from a merchant 2202. In some implementations, the user may exhibit consumption patterns. For example, the user may often buy a similar set of products simultaneously each time the user shops. In some implementations, the purchasing patterns of the user may be reflected in the card transactions conducted by the user. For example, the consumption patterns may reflect in card transaction records of the transactions conducted by the user, which may be mined by a card company, e.g., 2203. In some implementations, information as to the general preferences of the user, purchasing preferences of the user, cost-sensitivities of the user, etc. may be gleaned from studying the aggregated card transaction records pertaining to the user. For example, analysis of the aggregated user card transaction records may indicate a preference for shopping within a particular geographical area, at particular times, with particular merchants, for particular products types, categories, brand names, quantities, and/or the like. As another example, analysis of the aggregated card transaction records may indicate correlations between purchases of the user. For example, the analysis may provide the ability to predict (with a known confidence level) that a user may purchase product B given that the user has purchased (or intends to purchase) product A (or products A, and/or C, and/or D, etc.). Thus, in some implementations, analysis of the aggregated card transaction records of a user may allow the MMCDB to provide suggestions to the merchant and/or user as to products that the user is likely to be interested in purchasing. For example, a user may desire suggestions as to what products, services, offerings, deals that user may be interested in, e.g., 2204. In some implementations, the MMCDB may provide such suggestions, e.g., 106, to the user on a real-time basis (e.g., as the user is scanning products at a point-of-sale terminal, as the user is performing a price check, as the user is paying for a purchase, etc., as the user walks by a merchant where the MMCDB determines that products of interest to the user are available, etc.).

In some implementations, a merchant, e.g., 2202, may desire to understand customer behavior better so that the merchant may determine which products to provide for customers to generate maximum retail sales, generate customer loyalty, etc, e.g., 2205. In some implementations, the MMCDB may provide merchant analytics reports to the merchant including recommendations of product, service, discount, Groupon® offers, and/or other offers that the merchant can make to the user based on the user's behavioral patterns, e.g., 2206.

[00144] FIGURES 23A-B show data flow diagrams illustrating an example procedure to provide a user and/or merchant offers for products, services and/or the like, using user behavior patterns derived from card-based transaction data in some embodiments of the MMCDB. In some implementations, a user, e.g., 2301, may desire to purchase a product, service, offering, and/or the like ("product"), from a merchant. The user may communicate with a merchant server, e.g., 2303, via a client such as, but not limited to: a personal computer, mobile device, television, point-of-sale terminal, kiosk, ATM, pharmacy store, store counter, and/or the like (e.g., client 2302). For example, the user may provide user input, e.g., purchase input 2311, into the client indicating the user's desire to purchase the product. In various implementations, the user input may include, but not be limited to: keyboard entry, card swipe, activating a RFID/NFC enabled hardware device (e.g., electronic card having multiple accounts, smartphone, tablet, etc.), mouse clicks, depressing buttons on a joystick/game console, voice commands, single/multi-touch gestures on a touch-sensitive interface, touching user interface elements on a touch-sensitive display, and/or the like. For example, the user may direct a browser application executing on the client device to a website of the merchant, and may select a product from the website via clicking on a hyperlink presented to the user via the website. As another example, the client may obtain track 1 data from the user's card (e.g., credit card, debit card, prepaid card, charge card, etc.), such as the example track 1 data provided below:

```
%B123456789012345^PUBLIC/J.Q.^99011200000000000000**901******?*
(wherein '123456789012345' is the card number of 'J.Q. Public'
and has a CVV number of 901. '990112' is a service code, and ***
represents decimal digits which change randomly each time the
card is used.)
```

[00145] In some implementations, the client may generate a purchase order message, e.g., 2312, and provide, e.g., 2313, the generated purchase order message to the merchant server, e.g., 2303. For example, a browser application executing on the client may provide, on behalf of the user, a (Secure) Hypertext Transfer Protocol ("HTTP(S)") GET message including the product order details for the merchant server in the form of data formatted according to the eXtensible Markup Language ("XML"). Below is an example HTTP(S) GET message including an XML-formatted purchase order message for the merchant server:

```
GET /purchase.php HTTP/1.1
Host: www.merchant.com
Content-Type: Application/XML
Content-Length: 1306
<?XML version = "1.0" encoding = "UTF-8"?>
<purchase_order>
        <order_ID>4NFU4RG94</order_ID>
        <timestamp>2011-02-22 15:22:43</timestamp>
        <user_ID>john.q.public@gmail.com</user_ID>
        <client_details>
            <client_IP>192.168.23.126</client_IP>
            <client_type>smartphone</client_type>
            <client_model>HTC Hero</client_model>
            <OS>Android 2.2</OS>
            <app_installed_flag>true</app_installed_flag>
        </client_details>
        <purchase_details>
            <num_products>1</num_products>
            <product>
                <product_type>book</product_type>
                <product_params>
                    <product_title>XML for
dummies</product_title>
                    <ISBN>938-2-14-168710-0</ISBN>
                    <edition>2nd ed.</edition>
                    <cover>hardbound</cover>
                    <seller>bestbuybooks</seller>
```

```
                    </product_params>
                    <quantity>1</quantity>
               </product>
          </purchase_details>
          <account_params>
               <account_name>John Q. Public</account_name>
               <account_type>credit</account_type>
               <account_num>123456789012345</account_num>
               <billing_address>123 Green St., Norman, OK
98765</billing_address>
               <phone>123-456-7809</phone>
               <sign>/jqp/</sign>
               <confirm_type>email</confirm_type>

          <contact_info>john.q.public@gmail.com</contact_info>
          </account_params>
          <shipping_info>
               <shipping_adress>same as billing</shipping_address>
               <ship_type>expedited</ship_type>
               <ship_carrier>FedEx</ship_carrier>
               <ship_account>123-45-678</ship_account>
               <tracking_flag>true</tracking_flag>
               <sign_flag>false</sign_flag>
          </shipping_info>
</purchase_order>
```

[00146] In some implementations, the merchant server may, in response to receiving the purchase order message from the client, generate, e.g., 2314, a request for merchant analytics from a pay network server, e.g., 2305, so that the merchant may provide product offerings for the user. For illustration, in the example above, the merchant server may add an XML-encoded data structure to the body of the purchase order message, and forward the message to the pay network server. An example XML-encoded data snippet that the merchant server may add to the body of the purchase order message before forwarding to the pay network server is provided below:

```
<analytics_request>
```

```
<request_ID>NEUI4BGF9</request_ID>
<details>
    <type>products OR services OR discounts</type>
    <deliver_to>user AND merchant</deliver_to>
    <timeframe>realtime</timeframe>
    <user_priority>high</user_priority>
    <data_source>appended</data_source>
</details>
<merchant_params>
    <merchant_ID>3FBCR4INC</merchant_id>
    <merchant_name>Books & Things, Inc.</merchant_name>

<merchant_auth_key>1NNF484MCP59CHB27365</merchant_auth_ke
y>
</merchant_params>
</analytics_request>
```

[00147] The merchant server may provide the merchant analytics request, e.g., 2315, to the pay network server. In some implementations, the pay network server may extract the merchant and user profile information from the merchant analytics request. For illustration, the pay network server may extract values of the 'merchant_ID' and 'user_ID' fields from the merchant analytics request in the examples above. Using the merchant and user profile information, the pay network server may determine whether the merchant and/or user are enrolled in the merchant analytics program. In some implementations, the pay network server may provide the results of merchant analytics only to those entities that are enrolled in the merchant analytics program. For example, the server may query a database, e.g., pay network database 2307, to determine whether the user and/or merchant are enrolled in the merchant analytics program. In some implementations, the pay network server may generate a query the database for user behavior patterns of the user for merchant analytics, e.g., 2317. For example, the database may be a relational database responsive to Structured Query Language ("SQL") commands. The pay network server may execute a hypertext preprocessor ("PHP") script including SQL commands to query the database for user behavior patterns of the

user. An example PHP/SQL command listing, illustrating substantive aspects of querying the database, is provided below:

```
<?PHP
header('Content-Type: text/plain');
mysql_connect("254.93.179.112",$DBserver,$password); // access
database server
mysql_select_db("USERS.SQL"); // select database table to search
//create query for issuer server data
$query = "SELECT behavior_profile_XML FROM UserBehaviorTable
WHERE userid LIKE '%' $user_id";
$result = mysql_query($query); // perform the search query
mysql_close("USERS.SQL"); // close database access
?>
```

[00148] In response to obtaining the issuer server query, e.g., 2317, the pay network database may provide, e.g., 2318, the requested behavior patterns data to the pay network server. For example, the user behavior patterns data may comprise pair-wise correlations of various variables to each other, and/or raw user transaction patterns. An example XML-encoded user behavoir pattern data file is provided below:

```
<?XML version = "1.0" encoding = "UTF-8"?>
<last_updated>2011-02-22 15:22:43</timestamp>
<user_ID>john.q.public@gmail.com</user_ID>
<pair_correlation_data>
        <pair><time>AM</time><pdt>A</pdt><confidence>0.65</confid
ence></pair>
        <pair><pdt>B</pdt><pdt>A</pdt><confidence>0.95</confidenc
e></pair>
        <pair><zip>98456</zip><pdt>A</pdt><confidence>0.25</confi
dence></pair>
        <pair><time>PM</time><zip>98465</zip><confidence>0.45</co
nfidence></pair>
</pair_correlation_data>
<raw_data>
        <transaction>
```

```
        <timestamp>2011-02-21 15:32:01</timestamp>
        <product>
            <product_type>book</product_type>
            <product_params>
                <product_title>XML for dummies</product_title>
                <ISBN>938-2-14-168710-0</ISBN>
                <edition>2nd ed.</edition>
                <cover>hardbound</cover>
                <seller>bestbuybooks</seller>
            </product_params>
            <quantity>1</quantity>
        </transaction>
        .
        .
        .
        <transaction>
            . . .
        </transaction>
</raw_data>
```

[00149] In some implementations, the pay network server may identify products, services and/or other offerings likely desired by the user based on pre-generated user behavioral pattern analysis and user profile, e.g., 2319. The pay network server may generate a query, e.g., 2320, for merchants that may be able to provide the identified products, services, and/or offerings for the user. For example, the pay network server may generate a query based on the GPS coordinates of the user (e.g., obtained from the user's smartphone), the merchant store in which the user currently is present, etc., for merchants in the vicinity of the user who may have products included within the identified products likely desired by the user. In some implementations, the pay network server may also generate a query for offers (e.g., discount offers, Groupon® offers, etc.) that the merchant may be able to offer for the users. For example, the pay network server may utilize PHP/SQL commands similar to those provided above to query a database. In response, the database may provide, e.g., 2321, the requested merchant and/or offer data to the pay network server. In some implementations, the

pay network server may generate a real-time merchant analytics report for the merchant, e.g., 2322. In some implementations, the pay network server may generate a real-time geo-sensitive product offer packet for the user, e.g., including such items as (but not limited to): merchant names, location, directions, offers, discounts, interactive online purchase options, instant mobile wallet purchase ability, order hold placing features (e.g., to hold the items for pick up so as to prevent the items going out of stock, e.g., during seasonal shopping times), and/or the like. In some implementations, the pay network server may provide the merchant analytics report, e.g., 2324, to the merchant server, and may provide the real-time geo-sensitive product offer packet, e.g., 2327, to the client. In some implementations, the merchant server may utilize the pay network server's merchant analytics report to generate, e.g., 2325, offer(s) for the user. The merchant server may provide the generated offer(s), e.g., 2326, to the user. In some implementations, the client may render and display, e.g., 2328, the real-time geo-sensitive product offer packet from the pay network server and/or purchase offer(s) from the merchant to the user.

[00150] FIGURE 24 shows a logic flow diagram illustrating example aspects of providing a user and/or merchant offers for products, services and/or the like, using user behavior patterns derived from card-based transaction data in some embodiments of the MMCDB, e.g., a Merchant Analytics ("MA") component. In some implementations, the MMCDB may obtain a trigger to perform merchant analytics. For example a user may desire to purchase a product, service, offering, and/or the like ("product"), from a merchant (e.g., start scanning products in the checkout counter of the merchant's store), or may initiate a purchase transaction (e.g., attempt to pay for products purchased at the merchant store). In some implementations, the MMCDB may extract, e.g., 2402, the merchant and user profile information from the merchant analytics request. For example, the MMCDB may extract fields such as, but not limited to: user_ID, user_name, timestamp, merchant_ID, merchant_name, merchant_type, and/or the like. Using the merchant and/or user profile information, the MMCDB may generate a query the database for user behavior patterns, e.g., 2403, of the user for merchant analytics. In some implementations, the MMCDB may identify products, services and/or other offerings likely desired by the user based on pre-generated user

behavioral pattern analysis and user profile, e.g., 2404. The MMCDB may identify, e.g., 2405, merchants that may be able to provide the identified products, services, and/or offerings for the user. For example, the MMCDB may generate a query based on the GPS coordinates of the user (e.g., obtained from the user's smartphone), the merchant store in which the user currently is present, etc., for merchants in the vicinity of the user who may have products included within the identified products likely desired by the user. In some implementations, the pay network server may also determine offers (e.g., discount offers, Groupon® offers, etc.), e.g., 2406, that the merchant may be able to offer for the users. In some implementations, the MMCDB may generate a real-time merchant analytics report for the merchant, e.g., 2407. In some implementations, the MMCDB may generate, e.g., 2408, a real-time geo-sensitive product offer packet for the user, e.g., including such items as (but not limited to): merchant names, location, directions, offers, discounts, interactive online purchase options, instant mobile wallet purchase ability, order hold placing features (e.g., to hold the items for pick up so as to prevent the items going out of stock, e.g., during seasonal shopping times), and/or the like. In some implementations, the MMCDB may provide the merchant analytics report to the merchant server, and may provide the real-time geo-sensitive product offer packet to the client, e.g., 2409.

[00151] FIGURE 25 shows a logic flow diagram illustrating example aspects of generating a user behavior pattern analysis in some embodiments of the MMCDB, e.g., a User Behavioral Pattern Analytics ("UBPA") component. In some implementations, the MMCDB may select, e.g., 2501, a user (e.g., via user ID) for behavioral pattern analysis. The MMCDB may store, e.g., 2502, card-based transaction data records for each card-based transaction performed by the user, e.g., via a Card-Based Transaction Execution component. The MMCDB may aggregate such card-based transaction data records of the user, e.g., 2503. For example, the MMCDB may utilize a Transaction Data Aggregation component such as that described above with reference to FIGS. 8-9. In various implementations, the MMCDB may aggregate card transaction records of the user according to criteria including, but not limited to: geographical location of card use, time of card use, type of purchase, quantity of purchase, transaction value, merchant type, merchant name, spending category (e.g., such as the North American Industry

Classification System (NAICS) codes for spending categories), and/or the like. The MMCDB may analyze the aggregated card transaction data, e.g., 2504, to determine user behavioral patterns, e.g., via a User Pattern Identification ("UPI") component such as described below with reference to FIGURE 26. In some implementations, the MMCDB may provide user behavioral patterns obtained from the analysis for use by other MMCDB components and/or affiliated entities, e.g., 2505.

[00152] FIGURE 26 shows a logic flow diagram illustrating example aspects of identifying user behavioral patterns from aggregated card-based transaction data in some embodiments of the MMCDB, e.g., a User Patten Identification ("UPI") component. In some implementations, a pay network server ("server") may obtain a user ID of a user for whom the server is required to generate user behavioral patterns, e.g., 2601. The server may query a database, e.g., a pay network database, for aggregated card transaction data records of the user, e.g., 2602. The server may also query, e.g., 2603, the pay network database for all possible field value that can be taken by each of the field values (e.g., AM/PM, zipcode, merchant_ID, merchant_name, transaction cost brackets, etc.). Using the field values of all the fields in the transaction data records, the server may generate field value pairs, for performing a correlation analysis on the field value pairs, e.g., 2604. An example field value pair is: 'time' is 'AM' and 'merchant' is 'Walmart'. The server may then generate probability estimates for each field value pair occurring in the aggregated transaction data records. For example, the server may select a field value pair, e.g., 2605. The server may determine the number of records within the aggregated transaction data records where the field value pair occurs, e.g., 2606. The server may then calculate a probability quotient for the field value pair by dividing the number determined for the occurrences of the field value pair by the total number of aggregate transaction data records, e.g., 2607. The server may also assign a confidence level for the probability quotient based on the sample size, e.g., total number of records in the aggregated transaction data records, e.g., 2608. The server may generate and store an XML snippet, such as described above with reference to FIGURES 23A-B, including the field value pair, the probability quotient, and the confidence level associated with the probability quotient, e.g., 2609. The server may perform such a computation for each field value pair (see 2610) generated in 2604.

[00153] FIGURES 27A-B show block diagrams illustrating example aspects of merchant analytics in a second set of embodiments of the MMCDB. With reference to FIGURE 27A, in some implementations, the MMCDB may provide merchant analytics reports to various users in order to facilitate their making calculated investment decisions. For example, a stock investor may desire business analytics to determine which stocks the investor should invest in, how the investor should modify the investor's portfolio, and/or the like, e.g., 2701. In another example, a retailer may desire to understand customer behavior better so that the retailer may determine which products to provide for customers to generate maximum retail sales, e.g., 2702. In another example, a serviceperson providing services to customers may desire to understand which services the customer tend to prefer, and/or a paying for in the marketplace, e.g., 2703. In another example, a service provider may desire to understand the geographical areas where business for the serviceperson is likely to be concentrated, e.g., 2704. In some implementations, a credit card company may have access to a large database of card-based transactions. The card-based transaction may have distributed among them information on customer behavior, demand, geographical distribution, industry sector preferences, and/or the like, which may be mined in order to provide investors, retailer, service personnel and/or other users business analytics information based on analyzing the card-based transaction data. In some implementations, the MMCDB may take specific measures in order to ensure the anonymity of users whose card-based transaction data are analyzed for providing business analytics information for users. For example, the MMCDB may perform business analytics on anonymized card-based transaction data to provide solutions to questions such as illustrated in 2701-2704.

[00154] With reference to FIGURE 27B, in some implementations, the MMCDB may obtain an investment strategy to be analyzed, e.g., 2711, for example, from a user. The MMCDB may determine, e.g., 2712 the scope of the investment strategy analysis (e.g., geographic scope, amount of data required, industry segments to analyze, type of analysis to be generated, time-resolution of the analysis (e.g., minute, hour, day, month, year, etc.), geographic resolution (e.g., street, block, zipcode, metropolitan area, city, state, country, inter-continental, etc.). The MMCDB may aggregate card-based transaction data in accordance with the determined scope of analysis, e.g., 2713. The

MMCDB may normalized aggregated card-based transaction data records for uniform processing, e.g., 2714. In some implementations, the MMCDB may apply classification labels to card-based transaction data records, e.g., 2715, for investment strategy analysis. The MMCDB may filter the card-based transaction data records to include only those records as relevant to the analysis, e.g., 2716. For example, the MMCDB may utilize the classification labels corresponding to the transaction data records to determine which records are relevant to the analysis. In some implementations, the MMCDB may anonymize transaction data records for consumer privacy protection prior to investment strategy analysis, e.g., 2717. The MMCDB may perform econometrical investment strategy analysis, e.g., 2718, and generate an investment strategy analysis report based on the investment strategy analysis, e.g., 2719. The MMCDB may provide the investment strategy analysis report for the user requesting the investment strategy analysis.

[00155] FIGURES 28A-C show data flow diagrams illustrating an example procedure for econometrical analysis of a proposed investment strategy based on card-based transaction data in some embodiments of the MMCDB. With reference to FIGURE 28A, in some implementations, a user, e.g., 2801, may desire to obtain an analysis of an investment strategy. For example, the user may be a merchant, a retailer, an investor, a serviceperson, and/or the like provider or products, services, and/or other offerings. The user may communicate with a pay network server, e.g., 2805a, to obtain an investment strategy analysis. For example, the user may provide user input, e.g., analysis request input 2811, into a client, e.g., 2802, indicating the user's desire to request an investment strategy analysis. In various implementations, the user input may include, but not be limited to: keyboard entry, mouse clicks, depressing buttons on a joystick/game console, voice commands, single/multi-touch gestures on a touch-sensitive interface, touching user interface elements on a touch-sensitive display, and/or the like. In some implementations, the client may generate an investment strategy analysis request, e.g., 2812, and provide, e.g., 2813, the generated investment strategy analysis request to the pay network server. For example, a browser application executing on the client may provide, on behalf of the user, a (Secure) Hypertext Transfer Protocol ("HTTP(S)") GET message including the investment strategy analysis request

in the form of XML-formatted data. Below is an example HTTP(S) GET message including an XML-formatted investment strategy analysis request:

```
GET /analysisrequest.php HTTP/1.1
Host: www.paynetwork.com
Content-Type: Application/XML
Content-Length: 1306
<?XML version = "1.0" encoding = "UTF-8"?>
<analysis_request>
        <request_ID>EJ39FI1F</request_ID>
        <timestamp>2011-02-24 09:08:11</timestamp>
        <user_ID>investor@paynetwork.com</user_ID>
        <password>******</password>
        <request_details>
            <time_period>year 2011</time_period>
            <time_interval>month-to-month</time_interval>
            <area_scope>United States</area>
            <area_resolution>zipcode</area_resolution>
            <spend_sector>retail<sub>home
improvement</sub></spend_sector>
        </request_details>
        <client_details>
            <client_IP>192.168.23.126</client_IP>
            <client_type>smartphone</client_type>
            <client_model>HTC Hero</client_model>
            <OS>Android 2.2</OS>
            <app_installed_flag>true</app_installed_flag>
        </client_details>
</analysis_request>
```

[00156] In some implementations, the pay network server may parse the investment strategy analysis request, and determine the type of investment strategy analysis required, e.g., 2814. In some implementations, the pay network server may determine a scope of data aggregation required to perform the analysis. The pay network server may initiate data aggregation based on the determined scope, for example, via a Transaction Data Aggregation ("TDA") component such as described

above with reference to FIGURE 9. The pay network server may query, e.g., 2816, a pay network database, e.g., 2807, for addresses of pay network servers that may have stored transaction data within the determined scope of the data aggregation. For example, the pay network server may utilize PHP/SQL commands similar to the examples provided above. The database may provide, e.g., 2817, a list of server addresses in response to the pay network server's query. Based on the list of server addresses, the pay network server may issue transaction data requests, e.g., 2818b-n, to the other pay network servers, e.g., 2805b-n. The other the pay network servers may query their transaction databases, e.g., 2810b-n, for transaction data falling within the scope of the transaction data requests. In response to the transaction data queries, e.g., 2819b-n, the transaction databases may provide transaction data, e.g., 2820b-n, to the other pay network servers. The other pay network servers may return the transaction data obtained from the transactions databases, e.g., 2821b-n, to the pay network server making the transaction data requests, e.g., 2805a.

[00157] With reference to FIGURE 28B, the pay network server 2805a may aggregate, e.g., 2823, the obtained transaction data records, e.g. via the TDA component. The pay network server may normalize, e.g., 2824, the aggregated transaction data so that all the data share a uniform data structure format, e.g., via a Transaction Data Normalization ("TDN") component such as described below with reference to FIGURE 29. The pay network server may generate, e.g., 2825-2828, one or more classification labels for each of the transaction data records, e.g., via a Card-Based Transaction Classification ("CTC") component such as described below with reference to FIGURE 30. The pay network server may query for classification rules, e.g., 2826, a database, e.g., pay network database 2807. Upon obtaining the classification rules, e.g., 2827, the pay network server may generate, e.g., 2828, classified transaction data records using the classification rules, e.g., via the CTC component. The pay network server may filter, e.g., 2829, relevant transaction data records using the classification labels, e.g., via a Transaction Data Filtering ("TDF") component such as described below with reference to FIGURE 31. The pay network server may anonymize, e.g., 2830, the transaction data records, e.g., via a Consumer Data Anonymization ("CDA") component such as described below with reference to FIGURE 32.

[00158] With reference to FIGURE 28C, the pay network server may, in some implementations, store aggregated, normalized, classified, filtered, and/or anonymized data records, e.g., 2832, in a database, e.g., transactions database 2810a. In some implementations, the pay network server may econometrically analyze, e.g., 2833, aggregated, normalized, classified, filtered, and/or anonymized data records, e.g., via an *Econometrical Strategy Analysis ("ESA") component* such as described below with reference to FIGURE 33. The pay network server may prepare a report customized to the client used by the user. The pay network server may provide a reporting rules query to a database, e.g., pay network database 2807, for reporting rules to use in preparing the business analytics report. Upon obtaining the reporting rules, e.g., 2835, the pay network server may generate a business analytics report customized to the client, e.g., 2836, for example via a Business Analytics Reporting ("BAR") such as described below with reference to FIGURE 34. The pay network server may provide the business analytics report, e.g., 2837, to the client, e.g., 2802. The client may render and display, e.g., 2838, the business analytics report for the user.

[00159] FIGURE 29 shows a logic flow diagram illustrating example aspects of normalizing raw card-based transaction data into a standardized data format in some embodiments of the MMCDB, e.g., a Transaction Data Normalization ("TDN") component. In some implementations, a pay network server ("server") may attempt to convert any transaction data records stored in a database it has access to in a normalized data format. For example, the database may have a transaction data record template with predetermined, standard fields that may store data in pre-defined formats (e.g., long integer / double float / 4 digits of precision, etc.) in a pre-determined data structure. A sample XML transaction data record template is provided below:

```
<?XML version = "1.0" encoding = "UTF-8"?>
<transaction_record>
        <record_ID>00000000</record_ID>
        <norm_flag>false</norm_flag>
        <timestamp>yyyy-mm-dd hh:mm:ss</timestamp>
        <transaction_cost>$0,000,000,00</transaction_cost>
        <merchant_params>
              <merchant_id>00000000</merchant_id>
```

```
                <merchant_name>TBD</merchant_name>
                <merch_auth_key>0000000000000000</merch_auth_key>
        </merchant_params>
        <merchant_products>
                <num_products>000</num_products>
                <product>
                        <product_type>TBD</product_type>
                        <product_name>TBD</product_name>
                        <class_labels_list>TBD<class_labels_list>
                        <product_quantity>000</product_quantity>
                        <unit_value>$0,000,000.00</unit_value>
                        <sub_total>$0,000,000.00</sub_total>
                        <comment>normalized transaction data record
template</comment>
                </product>
        </merchant_products>
        <user_account_params>
                <account_name>JTBD</account_name>
                <account_type>TBD</account_type>
                <account_num>0000000000000000</account_num>
                <billing_line1>TBD</billing_line1>
                <billing_line2>TBD</billing_line2>
                <zipcode>TBD</zipcode>
                <state>TBD</state>
                <country>TBD</country>
                <phone>00-00-000-000-0000</phone>
                <sign>TBD</sign>
        </user_account_params>
</transaction_record>
```

[00160] In some implementations, the server may query a database for a normalized transaction data record template, e.g., 2901. The server may parse the normalized data record template, e.g., 2902. Based on parsing the normalized data record template, the server may determine the data fields included in the normalized data record template, and the format of the data stored in the fields of the data record template, e.g., 2903. The server may obtain transaction data records for normalization.

The server may query a database, e.g., 2904, for non-normalized records. For example, the server may issue PHP/SQL commands to retrieve records that do not have the 'norm_flag' field from the example template above, or those where the value of the 'norm_flag' field is 'false'. Upon obtaining the non-normalized transaction data records, the server may select one of the non-normalized transaction data records, e.g., 2905. The server may parse the non-normalized transaction data record, e.g., 2906, and determine the fields present in the non-normalized transaction data record, e.g., 2907. The server may compare the fields from the non-normalized transaction data record with the fields extracted from the normalized transaction data record template. For example, the server may determine whether the field identifiers of fields in the non-normalized transaction data record match those of the normalized transaction data record template, (e.g., via a dictionary, thesaurus, etc.), are identical, are synonymous, are related, and/or the like. Based on the comparison, the server may generate a 1:1 mapping between fields of the non-normalized transaction data record match those of the normalized transaction data record template, e.g., 2909. The server may generate a copy of the normalized transaction data record template, e.g., 2910, and populate the fields of the template using values from the non-normalized transaction data record, e.g., 2911. The server may also change the value of the 'norm_flag' field to 'true' in the example above. The server may store the populated record in a database (for example, replacing the original version), e.g., 2912. The server may repeat the above procedure for each non-normalized transaction data record (see e.g., 2913), until all the non-normalized transaction data records have been normalized.

[00161] FIGURE 30 shows a logic flow diagram illustrating example aspects of generating classification labels for card-based transactions in some embodiments of the MMCDB, e.g., a Card-Based Transaction Classification ("CTC") component. In some implementations, a server may apply one or more classification labels to each of the transaction data records. For example, the server may classify the transaction data records, according to criteria such as, but not limited to: geo-political area, luxury level of the product, industry sector, number of items purchased in the transaction, and/or the like. The server may obtain transactions from a database that are unclassified, e.g., 3001, and obtain rules and labels for classifying the records, e.g., 3002. For example,

the database may store classification rules, such as the exemplary illustrative XML-encoded classification rule provided below:

```
<rule>
        <id>NAICS44_45</id>
        <name>NAICS - Retail Trade</name>
        <inputs>merchant_id</inputs>
        <operations>
            <1>label = 'null'</1>
            <1>cat = NAICS_LOOKUP(merchant_id)</1>
            <2>IF (cat == 44 || cat ==45) label = 'retail
trade'</2>
        </operations>
        <outputs>label</outputs>
</rule>
```

[00162] The server may select an unclassified data record for processing, e.g., 3003. The server may also select a classification rule for processing the unclassified data record, e.g., 3004. The server may parse the classification rule, and determine the inputs required for the rule, e.g., 3005. Based on parsing the classification rule, the server may parse the normalized data record template, e.g., 3006, and extract the values for the fields required to be provided as inputs to the classification rule. For example, to process the rule in the example above, the server may extract the value of the field 'merchant_id' from the transaction data record. The server may parse the classification rule, and extract the operations to be performed on the inputs provided for the rule processing, e.g., 3007. Upon determining the operations to be performed, the server may perform the rule-specified operations on the inputs provided for the classification rule, e.g., 3008. In some implementations, the rule may provide threshold values. For example, the rule may specify that if the number of products in the transaction, total value of the transaction, average luxury rating of the products sold in the transaction, etc. may need to cross a threshold in order for the label(s) associated with the rule to be applied to the transaction data record. The server may parse the classification rule to extract any threshold values required for the rule to apply, e.g., 3009. The server may compare the computed values with the rule thresholds, e.g., 3010. If the rule

threshold(s) is crossed, e.g., 3011, option "Yes," the server may apply one or more labels to the transaction data record as specified by the classification rule, e.g., 3012. For example, the server may apply a classification rule to an individual product within the transaction, and/or to the transaction as a whole. In some implementations, the server may process the transaction data record using each rule (see, e.g., 3013). Once all classification rules have been processed for the transaction record, e.g., 3013, option "No," the server may store the transaction data record in a database, e.g., 3014. The server may perform such processing for each transaction data record until all transaction data records have been classified (see, e.g., 3015).

[00163] FIGURE 31 shows a logic flow diagram illustrating example aspects of filtering card-based transaction data for econometrical investment strategy analysis in some embodiments of the MMCDB, e.g., a Transaction Data Filtering ("TDF") component. In some implementations, a server may filter transaction data records prior to econometrical investment strategy analysis based on classification labels applied to the transaction data records. For example, the server may filter the transaction data records, according to criteria such as, but not limited to: geo-political area, luxury level of the product, industry sector, number of items purchased in the transaction, and/or the like. The server may obtain transactions from a database that are classified, e.g., 3101, and investment strategy analysis parameters, e.g., 3102. Based on the analysis parameters, the server may generate filter rules for the transaction data records, e.g., 3103. The server may select a classified data record for processing, e.g., 3104. The server may also select a filter rule for processing the classified data record, e.g., 3105. The server may parse the filter rule, and determine the classification labels required for the rule, e.g., 3106. Based on parsing the classification rule, the server may parse the classified data record, e.g., 3107, and extract values for the classification labels (e.g., true/false) required to process the filter rule. The server may apply the classification labels values to the filter rule, e.g., 3108, and determine whether the transaction data record passes the filter rule, e.g., 3109. If the data record is admissible in view of the filter rule, e.g., 3110, option "Yes," the server may store the transaction data record for further analysis, e.g., 3112. If the data record is not admissible in view of the filter rule, e.g., 3110, option "No," the server may select another filter rule to process the

transaction data record. In some implementations, the server may process the transaction data record using each rule (see, e.g., 3111) until all rules are exhausted. The server may perform such processing for each transaction data record until all transaction data records have been filtered (see, e.g., 3113).

[00164] FIGURE 32 shows a logic flow diagram illustrating example aspects of anonymizing consumer data from card-based transactions for econometrical investment strategy analysis in some embodiments of the MMCDB, e.g., a Consumer Data Anonymization ("CDA") component. In some implementations, a server may remove personal information relating to the user (e.g., those fields that are not required for econometrical investment strategy analysis) and/or merchant from the transaction data records. For example, the server may truncate the transaction data records, fill randomly generated values in the fields comprising personal information, and/or the like. The server may obtain transactions from a database that are to be anonymized, e.g., 3201, and investment strategy analysis parameters, e.g., 3202. Based on the analysis parameters, the server may determine the fields that are necessary for econometrical investment strategy analysis, e.g., 3203. The server may select a transaction data record for processing, e.g., 3204. The server may parse the transaction data record, e.g., 3205, and extract the data fields in the transactions data records. The server may compare the data fields of the transaction data record with the fields determined to be necessary for the investment strategy analysis, e.g., 3206. Based on the comparison, the server may remove any data fields from the transaction data record, e.g., those that are not necessary for the investment strategy analysis, and generate an anonymized transaction data record, e.g., 3207. The server may store the anonymized transaction data record in a database, e.g., 3208. In some implementations, the server may process each transaction data record (see, e.g., 3209) until all the transaction data records have been anonymized.

[00165] FIGURES 33A-B show logic flow diagrams illustrating example aspects of econometrically analyzing a proposed investment strategy based on card-based transaction data in some embodiments of the MMCDB, e.g., an Econometrical Strategy Analysis ("ESA") component. With reference to FIGURE 33A, in some implementations, the server may obtain spending categories (e.g., spending categories

as specified by the North American Industry Classification System ("NAICS")) for which to generate estimates, e.g., 3301. The server may also obtain the type of forecast (e.g., month-to-month, same-month-prior-year, yearly, etc.) to be generated from the econometrical investment strategy analysis, e.g., 3302. In some implementations, the server may obtain the transaction data records using which the server may perform econometrical investment strategy analysis, e.g., 3303. For example, the server may select a spending category (e.g., from the obtained list of spending categories) for which to generate the forecast, e.g., 3304. For example, the forecast series may be several aggregate series (described below) and the 12 spending categories in the North American Industry Classification System (NAICS) such as department stores, gasoline, and so on, that may be reported by the Department of Commerce (DOC).

[00166] To generate the forecast, the server may utilize a random sample of transaction data (e.g., approximately 6% of all transaction data within the network of pay servers), and regression analysis to generate model equations for calculating the forecast from the sample data. For example, the server may utilize distributed computing algorithms such as Google MapReduce. Four elements may be considered in the estimation and forecast methodologies: (a) rolling regressions; (b) selection of the data sample ("window") for the regressions; (c) definition of explanatory variables (selection of accounts used to calculate spending growth rates); and (d) inclusion of the explanatory variables in the regression equation ("candidate" regressions) that may be investigated for forecasting accuracy. The dependent variable may be, e.g., the growth rate calculated from DOC revised sales estimates published periodically. Rolling regressions may be used as a stable and reliable forecasting methodology. A rolling regression is a regression equation estimated with a fixed length data sample that is updated with new (e.g., monthly) data as they become available. When a new data observation is added to the sample, the oldest observation is dropped, causing the total number of observations to remain unchanged. The equation may be estimated with the most recent data, and may be re-estimated periodically (e.g., monthly). The equation may then be used to generate a one-month ahead forecast for year-over-year or month over month sales growth.

[00167] Thus, in some implementations, the server may generate N window lengths (e.g., 18 mo, 24 mo, 36 mo) for rolling regression analysis, e.g., 3305. For each of the candidate regressions (described below), various window lengths may be tested to determine which would systemically provide the most accurate forecasts. For example, the server may select a window length may be tested for rolling regression analysis, e.g., 3306. The server may generate candidate regression equations using series generated from data included in the selected window, e.g., 3307. For example, the server may generate various series, such as, but not limited to:

[00168] Series (1): Number of accounts that have a transaction in the selected spending category in the current period (e.g., month) and in the prior period (e.g., previous month / same month last year);

[00169] Series (2): Number of accounts that have a transaction in the selected spending category in the either the current period (e.g., month), and/or in the prior period (e.g., previous month / same month last year);

[00170] Series (3): Number of accounts that have a transaction in the selected spending category in the either the current period (e.g., month), or in the prior period (e.g., previous month / same month last year), but not both;

[00171] Series (4): Series (1) + overall retail sales in any spending category from accounts that have transactions in both the current and prior period;

[00172] Series (5): Series (1) + Series (2) + overall retail sales in any spending category from accounts that have transactions in both the current and prior period; and

[00173] Series (6): Series (1) + Series (2) + Series (3) + overall retail sales in any spending category from accounts that have transactions in both the current and prior period.

[00174] With reference to FIGURE 33B, in some implementations, the server may calculate several (e.g., six) candidate regression equations for each of the series. For example, the server may calculate the coefficients for each of the candidate regression equations. The server may calculate a value of goodness of fit to the data for each candidate regression equations, e.g., 3308. For example, two measures of goodness of

fit may be used: (1) out-of-sample (simple) correlation; and (2) minimum absolute deviation of the forecast from revised DOC estimates. In some implementations, various measures of goodness of fit may be combined to create a score. In some implementations, candidate regression equations may be generated using rolling regression analysis with each of the N generated window lengths (see, e.g., 3309). In some implementations, upon generation of all the candidate regression equations and their corresponding goodness of fit scores, the equation(s) with the best score is chosen as the model equation for forecasting, e.g., 3310. In some implementations, the equation(s) with the highest score is then re-estimated using latest retail data available, e.g., from the DOC, e.g., 3311. The rerun equations may be tested for auto correlated errors. If the auto correlation test is statistically significant then the forecasts may include an autoregressive error component, which may be offset based on the autocorrelation test.

[00175] In some implementations, the server may generate a forecast for a specified forecast period using the selected window length and the candidate regression equation, e.g., 3312. The server may create final estimates for the forecast using DOC estimates for prior period(s), e.g., 3313. For example, the final estimates (e.g., F_t^Y - year-over-year growth, F_t^M - month-over-month growth) may be calculated by averaging month-over-month and year-over-year estimates, as follows:

$$D_t^Y = \left(1+G_t^Y\right) \bullet R_{t-12}$$

$$D_t^M = \left(1+G_t^M\right) \bullet A_{t-1}$$

$$D_t = Mean\left(D_t^M, D_t^Y\right)$$

$$B_{t-1}^Y = \left(1+G_{t-1}^Y\right) \bullet R_{t-13}$$

$$B_{t-1}^M = A_{t-1}$$

$$B_{t-1} = Mean\left(B_{t-1}^M, B_{t-1}^Y\right)$$

$$F_t^Y = D_t / R_{t-12} - 1$$

$$F_t^M = D_t / B_{t-1} - 1$$

[00176] Here, G represents the growth rates estimated by the regressions for year (superscript Y) or month (superscript M), subscripts refer to the estimate period, t is the current forecasting period); R represents the DOC revised dollar sales estimate; A represents the DOC advance dollar estimate; D is a server-generated dollar estimate, B is a base dollar estimate for the previous period used to calculate the monthly growth forecast.

[00177] In some implementations, the server may perform a seasonal adjustment to the final estimates to account for seasonal variations, e.g., 3314. For example, the server may utilize the X-12 ARIMA statistical program used by the DOC for seasonal adjustment. The server may then provide the finalized forecast for the selected spending category, e.g., 3315. Candidate regressions may be similarly run for each spending category of interest (see, e.g., 3316).

[00178] FIGURE 34 shows a logic flow diagram illustrating example aspects of reporting business analytics derived from an econometrical analysis based on card-based transaction data in some embodiments of the MMCDB, e.g., a Business Analytics Reporting ("BAR") component. In some implementations, the server may customize a business analytics report to the attributes of a client of the user requesting the investment strategy analysis, e.g., 3401. The server may obtain an investment strategy analysis request from a client. The request may include details about the client such as, but not limited to: client_type, client_IP, client_model, client_OS, app_installed_flag, and/or the like. The server may parse the request, e.g., 3402, and determine the type of client (e.g., desktop computer, mobile device, smartphone, etc.). Based on the type of client, the server may determine attributes of the business analytics report, including but not limited to: report size; report resolution, media format, and/or the like, e.g., 3403. The server may generate the business analytics report according to the determined attributes, e.g., 3404. The server may compile the report into a media format according to the attributes of the client, e.g., 3405, and provide the business analytics report for the client, e.g., 3406. Optionally, the server may initiate actions

(e.g., generate a market data feed, trigger an investment action, trigger a wholesale purchase of goods for a retailer, etc.) based on the business analytics report and/or data utilized in preparing the business analytics report, e.g., 3407.

Analytical Model Sharing

[00179] Thus, as seen from the discussion above, in various embodiments, the MMCDB facilitates the creation of analytical models using which the data aggregated by the Centralized Personal Information Platform of the MMCDB may be utilized to provide business or other intelligence to the various users of the MMCDB. Examples of analytical models include the components discussed above in the discussion with reference to FIGURES 24 and 33A-B. In some implementations, the MMCDB may facilitate the sharing of such analytical models among various users and/or other entities or components associated with the MMCDB. For example, a developer of an analytical model such as the real-time offer merchant analytics report-generating component of FIGURE 24 may distribute the analytical model to other users of the MMCDB. Optionally, the model may be described according to an encryptmatics XML format, as discussed in detail further below in this disclosure. In some embodiments, the analytical model may be provide without providing the model data sets based on which the model was developed, so as to protect the privacy of the consumers whose data were included in the model data set. In alternate embodiments, the MMCDB may utilize a consumer data anonymization component such as that described above with reference to FIGURE 32, before sharing the model data set along with the analytical model.

[00180] FIGURE 35 shows a logic flow diagram illustrating example aspects of sharing an analytical model generated using data acquired using the centralized personal information platform in some embodiments of the MMCDB, e.g., an Analytical Model Sharing ("AMS") component. In some embodiments, the MMCDB may obtain an analytical model provided for sharing with other users, e.g., 3501. The MMCDB may parse the analytical model, e.g., using one of the parsers described below with reference to FIGURE 49. The MMCDB may, based on the parsing of the model, identify any model data set used to develop the analytical model, that is included in the model

provided for sharing, e.g., 3502. The MMCDB may determine, if such a model dataset is included, whether the model dataset includes private data that may be shared on an open exchange, e.g., personally identifiable information, e.g., 3503. If the data is allowed to be shared, e.g., 3504, option "No," the MMCDB may provide the analytical model for sharing, e.g., to a model exchange website, without any further processing, e.g., 3505. If however, the model dataset include data that may not be shared, e.g., 3504, option "Yes," the MMCDB may determine whether the model dataset may be excluded from the model, while still allowing the model to be properly used, e.g., 3506. If the model dataset is excludable, e.g., 3507, option "Yes," the MMCDB may delete the model dataset from the analytical model, e.g., 3508, and provide the modified analytical model for sharing, e.g., 3505. If however, the model dataset is not excludable, e.g., 3507, option "No," the MMCDB may determine whether the dataset can be anonymized while still allowing the analytical model to function properly. If the dataset can be anonymized, e.g., 3509, option "Yes," the MMCDB may anonymize the model dataset, e.g., using the CDA component discussed above with reference to FIGURE 32, e.g., 3510. Then, the MMCDB may provide the modified analytical model including the anonymized dataset for sharing, e.g., 3505. If however, the model dataset cannot be anonymized, e.g., 3509, option "No," the MMCDB may generate and provide a sharing rejection message to the provider of the analytical model for sharing, e.g., 3511.

Encryptmatics XML Data Converter

[00181] In some embodiments, the MMCDB may utilize metadata (e.g., easily configurable data) to drive an analytics and rule engine that may convert any structured data obtained via the centralized personal information platform, discussed above in this disclosure, into a standardized XML format ("encryptmatics" XML). See FIGURE 14B for additional description. The encryptmatics XML may then be processed by an encryptmatics engine of the MMCDB that is capable of parsing, transforming and analyzing data to generate decisions based on the results of the analysis. Accordingly, in some embodiments, the MMCDB may implement a metadata-based interpretation engine that parses structured data, including, but not limited to: web content, graph databases, micro blogs, images or software code, and convert the structured data into

commands in the encryptmatics XML file format. The structured data may include, without limitation, software code, images, free text, relational database queries, graph queries, sensory inputs, and/or the like. As an example, the MMCDB may convert software code written in SAS integrated system of software products provided by SAS Institute, Inc., into a standard encryptmatics XML structure. The example below shows *SAS code and encryptmatics XML that serves a similar purpose to the SAS code* - defining a module's inputs, outputs, and function calls:

```
// SAS
filename myFTL "my.378.FTL";
data MyFile;
     length yyddd $5. ;
     infile myFTL lrecl=50000;
     input @21 yyddd  ;
run;

// Encryptmatics XML
<lock name="F: Transaction Date : yyddd"
  inkeyid="0" inkeystart="21" inkeystop="25"
  outkeyid="31" outkeyindex="1"
  function="INSTANT"
  type="STRING"
/>
```

[00182] In the encryptmatics XML examples herein, a "key" represents a collection of data values. A "tumblar" represents a hash lookup table that may also allow wild card searches. A "lock" represents a definition including one or more input data sources, data types for the input sources, one or more data output storage variables, and functions/modules that may be called to process the input data from the input data sources. A "door" may refer to a collection of locks, and a vault may represent a model package defining the input, output, feature generation rules and analytical models. Thus, the encryptmatics XML may be thought of as a framework for calling functions (e.g., INSTANT - returns the raw value, LAG - return a key from a prior transaction, ADD - add two keys together, OCCURRENCE - returns the number of times a key value

occurred in prior transactions, AVG - returns an average of past and current key values, etc.) and data lookups with a shared storage space to process a grouped data stream.

[00183] In some embodiments, a metadata based interpretation engine may populate a data/command object (e.g., an encryptmatics XML data structure defining a "vault") based on a given data record, using configurable metadata. The configurable metadata may define an action for a given glyph or keyword contained within a data record. The MMCDB may obtain the structured data, and perform a standardization routine using the structured data as input (e.g., including script commands, for illustration). For example, the MMCDB may remove extra line breaks, spaces, tab spaces, etc. from the structured data. The MMCDB may determine and load a metadata library, using which the MMCDB may parse subroutines or functions within the script, based on the metadata. In some embodiments, the MMCDB may pre-parse conditional statements based on the metadata. The MMCDB may also parse data to populate a data/command object based on the metadata and prior parsing. Upon finalizing the data/command object, the MMCDB may export the data/command object as XML in standardized encryptmatics format. For example, the engine may process the object to export its data structure as a collection of encryptmatics vaults in a standard encryptmatics XML file format. The encryptmatics XML file may then be processed to provide various features by an encryptmatics engine.

[00184] As an example, using such a metadata based interpretation engine, the MMCDB can generate the encryptmatics XML code, provided below, from its equivalent SAS code, provided beneath the encryptmatics XML code generated from it:

```
// SAS function code
myInput    = filename("../data/30x.raw", "fixed",
"../metaData/ftl_302.meta");

data myout;
 set myInput;
  auth_amt    = float(myInput.auth_amt);
  auth_amt2   = log(auth_amt);
 run;
```

```
proc freq data = myout;
   tables auth_amt2  ;
 run;


// Equivalent encryptmatics XML function code
<init>
loop=mainLoop
  <input>
     keyname=myinput
     file=../data/30x.raw
     format=fixed
      meta_data= ../metaData/ftl_302.meta
  </input>
  <output>
    keyname=myout
    file=VARRAY
    format=VARRAY
    meta_data= {'auth_amt2': (1, 0, 'String'), 'auth_amt': (0, 0,
'String')}
  </output>
</init>
<vault>
  <door>
   <lock>
     outkey=myout
     outkeyname=auth_amt
     inkey=myinput
     inkeyname=auth_amt
     function=float
     type=String
   </lock>
   <lock>
     outkey=myout
     outkeyname=auth_amt2
     inkey=myout
     inkeyname=auth_amt
```

```
      function=log
      type=String
    </lock>
  </door>
</vault>
 <init>
summary_level=2
loop=mainLoop
  <input>
     keyname=myout
     file=VARRAY:myout
     format=array
      meta_data= {'auth_amt2': (1, 0, 'String'), 'auth_amt': (0,
0, 'String')}
  </input>
  <output>
    keyname=_output
    file=stdout
    format=VARRAY
    meta_data= {'_agg1': (0, 0, 'object')}
  </output>
</init>
<vault>
  <door>
   <lock>
     outkey=agg1
     outkeyname=auth_amt2
     inkey=myout
     inkeyname=auth_amt2
     function=instant
     type=String
   </lock>
  </door>
  <door>
   <lock>
     outkey=_output
     outkeyname=_agg1
```

```
      function=aggfreq
      type=object
      parser=noparse
      groupkeyname=agg1
    </lock>
  </door>
</vault>
```

[00185] As another example, using such a metadata based interpretation engine, the MMCDB can generate the encryptmatics XML code, provided below, from its equivalent SAS code, provided beneath the encryptmatics XML code generated from it:

```
// SAS function code
myInput            = filename("../data/vnd.test.json", "JSON",
"../metaData/enrollment.meta");
 myTumblar = tumblarname("../tumblars/enrollment.exp.tumblar");

 data myOut;
   set myInput;
     customer_ipaddresstmp = tumble(myInput.customer_ipaddress  ,
customer_ipaddress   );
     customer_ipaddress    =   myOut.customer_ipaddresstmp/1000;
     cvv_resulttmp = tumble(myInput.cv_result   ,   cv_result );
     cv_result    =   myOut.cvv_resulttmp/1000;
 keep customer_ipaddress cv_result;
 run;

 proc model data = myOut out=Scored;
  features = customer_ipaddress cv_result ;
  weights  = 1,1 ;
  type     = 'bayes'  ;
 run;

 proc print data = Scored;
 run;
```

```
// Equivalent encryptmatics XML function code
<init>
loop=mainLoop
  <input>
     keyname=myinput
     file=../data/vnd.test.json
     format=JSON
      meta_data= ../metaData/ftl_302.meta  <br/>
      </input>
  <output>
    keyname=myout
    file=VARRAY
    format=VARRAY
    meta_data= {'cv_result': (1, 0, 'String'),
'customer_ipaddress': (0, 0, 'String')}
  </output>
  <constant>
     indexname=_constant_1000
     value=1000
     type=float
  </constant>
</init>
<vault>
  <door>
   <lock>
     outkey=_old_myout
     outkeyname=customer_ipaddresstmp
     inkey=myinput
     inkeyname=customer_ipaddress
     function=INSTANT
     type=String
     tumblar=customer_ipaddress
   </lock>
   <lock>
     outkey=_old_myout
     outkeyname=customer_ipaddress
     inkey=_old_myout
```

```
    inkeyname=customer_ipaddresstmp
    inkey2=_constants
    inkey2name=_constant_1000
    function=divide
    type=String
  </lock>
  <lock>
    outkey=_old_myout
    outkeyname=cvv_resulttmp
    inkey=myinput
    inkeyname=cv_result
    function=INSTANT
    type=String
    tumblar=cv_result
  </lock>
  <lock>
    outkey=_old_myout
    outkeyname=cv_result
    inkey=_old_myout
    inkeyname=cvv_resulttmp
    inkey2=_constants
    inkey2name=_constant_1000
    function=divide
    type=String
  </lock>
  <lock>
    outkey=myout
    outkeyname=customer_ipaddress
    inkey=_old_myout
    inkeyname=customer_ipaddress
    function=instant
    type=String
  </lock>
  <lock>
    outkey=myout
    outkeyname=cv_result
    inkey=_old_myout
```

```
      inkeyname=cv_result
      function=instant
      type=String
    </lock>
  </door>
</vault>
 <init>
loop=mainLoop
  <input>
     keyname=myout
     file=VARRAY:myout
     format=array
      meta_data= {'cv_result': (1, 0, 'String'),
'customer_ipaddress': (0, 0, 'String')}
  </input>
  <output>
    keyname=scored
    file=VARRAY
    format=VARRAY
    meta_data= {'_mdl1': (0, 0, 'float')}
  </output>
</init>
<vault>
  <door>
   <lock>
     outkey=mdl1
     outkeyname=customer_ipaddress
     inkey=myout
     inkeyname=customer_ipaddress
     function=instant
     type=String
   </lock>
   <lock>
     outkey=mdl1
     outkeyname=cv_result
     inkey=myout
     inkeyname=cv_result
```

```
      function=instant
      type=String
    </lock>
  </door>
  <door>
    <lock>
      outkey=scored
      outkeyname=_mdl1
      function=_mdl1
      type=float
      fnc-weights=1.0,1.0
      function=SUMPROB
      model=bayes
      parser=noparse
      groupkeyname=mdl1
    </lock>
  </door>
</vault>
 <init>
outputall=True
loop=print
  <input>
     keyname=scored
     file=VARRAY:scored
     format=array
      meta_data= {'_mdl1': (0, 0, 'float')}
  </input>
  <output>
    keyname=_output
    file=stdout
    format=VARRAY
    meta_data= {'_mdl1': (0, 0, 'String')}
  </output>
</init>
<vault>
  <door>
   <lock>
```

```
        outkey=_output
        outkeyname=_mdl1
        inkey=scored
        inkeyname=_mdl1
        function=instant
        type=String
      </lock>
    </door>
  </vault>
```

[00186] FIGURE 36 shows a logic flow diagram illustrating example aspects of a metadata based interpretation engine of the MMCDB that generates standardized encryptmatics XML from structured data obtained from various sources via the centralized personal information platform, e.g., an Encryptmatics XML Converter ("EXC") component. In some embodiments, the MMCDB may obtain a structured data object for conversion into encryptmatics XML format, e.g., 3601. The MMCDB may parse the structured data, e.g., 3602. For example, the MMCDB may utilize parsers such as the example parsers discussed below with reference to FIGURE 49. In some embodiments, the MMCDB may determine and load a metadata library, using which the MMCDB may parse subroutines or functions within the script, based on the metadata. In some embodiments, the MMCDB may pre-parse conditional statements based on the metadata. The MMCDB may also parse data to populate a data/command object based on the metadata and prior parsing. The MMCDB may obtain the structured data, and perform a standardization routine using the structured data as input (e.g., including script commands, for illustration). For example, the MMCDB may optionally eliminate superfluous characters, e.g., extra line breaks, spaces, tabs, etc., to generate a modified structured data object, e.g., 3603. The MMCDB may extract a glyph or keywords from the modified structured data, e.g., 3604. The MMCDB may, using the metadata library, lookup a database (e.g., a metadata library) for an encryptmatics XML metadata code snippet corresponding to the extracted glyph or keyword, e.g., 3605, and append the retrieved encryptmatics XML metadata to a metadata object, e.g., 3606. The MMCDB may perform such a routine until all glyphs or keywords are extracted and processed from the modified structured data, see e.g., 3607. Then, the MMCDB may, upon

finalizing the data/command object, export the data/command object as XML in standardized encryptmatics file format, e.g., 3608. For example, the engine may process the object to export its data structure as a collection of encryptmatics vaults in a standard encryptmatics XML file format. In some embodiments, the MMCDB may execute an application defined by the exported encryptmatics file, e.g., on other structured data available in the centralized personal information platform, e.g., 3609.

[00187] Thus, in some embodiments, the MMCDB may gradually convert the entire centralized personal information platform from structured data into standardized encryptmatics XML format. The MMCDB may also generate structured data as an output from the execution of the standardized encryptmatics XML application, and add the structured data to the centralized personal information platform databases, e.g., 3610. In some embodiments, the MMCDB may recursively provides structured data generated as a result of execution of the encryptmatics XML application as input into the EXC component, e.g. 3611.

[00188] FIGURE 37 shows a data flow diagram illustrating an example email data aggregation procedure in some embodiments of the MMCDB. In some implementations, the pay network server may obtain a trigger to extract one or more monitorable user email addresseses and generate an email access API query in order to monitor a user's email activity and aggregate the content thereof. For example, the pay network server may periodically perform an update of its aggregated database, e.g., 3710a, with new information available from the user's email account activity operating on the Internet. In one embodiment, the information aggregated is the raw content of email messages, including header information containing the server delivery path through which the message has passed. As another example, a request for email data aggregation update may be obtained as a result of a user wishing to enroll in a service, for which the pay network server may facilitate data entry by providing an automated web form filling system using information about the user obtained from the email data aggregation update. In some implementations, the pay network server may parse the trigger to extract access credentials with which to perform an email data aggregation update. The pay network server may generate a query for application programming interface (API) templates for various email provider services (e.g., Gmail™, Yahoo

Mail™, etc.) from which to collect email data for aggregation. In some embodiments, the aggregation templates will be configured to provide access to the user's email account at the email service provider. In other embodiments, the aggregation templates will be configured to provide a mechanism to parse retrieved user email into a more suitable format for processing. In still other embodiments, the templates may indicate that an email transfer protocol (such as POP, IMAP, and/or the like) should be employed. In some instances, the email transfer protocol may be used over a secondary secured connection (such as SSH, PGP, and/or the like).

[00189] In one embodiment, the pay network server may query, e.g., 3712, a pay network database, e.g., 3707, for email aggregation API templates for the email provider services. For example, the pay network server may utilize PHP/SQL commands similar to the examples provided above. The database may provide, e.g., 3713, a list of email access API templates in response. Based on the list of API templates, the pay network server may generate email aggregation requests, e.g., 3714. The pay network server may issue the generated email aggregation requests, e.g., 3715a-c, to the email network servers, e.g., 3701a-c. For example, the pay network server may issue PHP commands to request the email provider servers for email data. An example listing of commands to issue email aggregation data requests 3715a-c, substantially in the form of PHP commands, is provided below:

```
<?php

$aggregated_email = "";
$mail = imap_open('{server.com:110/pop3}',
                  $user,
                  $password);
$headers = imap_headers($mail);

for ($i=1; $i<=count($headers); $i++) {
        $structure = imap_fetchstructure($mail, $i);
        $structure_parts = $structure->parts;
        $number_parts = count($structure_parts);
        for ($j=0; $j<=$number_parts; $j++) {
             $text = imap_fetchbody($mail,$i,$j);
```

```
                $aggregated_email .=
                    nl2br(htmlspecialchars($text))."<br>";
    }
}

?>
```

[00190] In some embodiments, the email provider servers may query, e.g., 3717a-c, their databases, e.g., 3710a-c, for email aggregation results falling within the scope of the email aggregation request. In response to the queries, the databases may provide email data, e.g., 3718a-c, to the email provider servers. The email provider servers may return the email data obtained from the databases, e.g., 3719a-c, to the pay network server making the email aggregation requests. An example listing of email data 3719a-c, substantially in the form of JavaScript Object Notation (JSON)-formatted data, is provided below:

```
["data":[
  {"headers": "Delivered-To: MrSmith@gmail.com
Received: by 10.36.81.3 with SMTP1 id e3cs239nzb; Tue, 5 Mar 2020
15:11:47 -0800 (PST)
Return-Path:
Received: from mail.emailprovider.com (mail.emailprovider.com
[111.111.11.111]) by mx.gmail.com with SMTP id
h19si826631rnb.2005.03.29.15.11.46; Tue, 5 Mar 2020 15:11:47 -
0800 (PST)
Message-ID: <20050329231145.62086.mail@mail.emailprovider.com>
Received: from [11.11.111.111] by mail.emailprovider.com via
HTTP; Tue, 5 Mar 2020 15:11:45 PST
Date: Tue, 5 Mar 2020 15:11:45 -0800 (PST)
From: Mr Jones
Subject: Dinner at Patsy's this weekend?
To: Mr Smith",
    "from_addr": "MrJones@gmail.com",
    "from_name": "Mr Jones",
    "to_addr": "MrSmith@gmail.com",
    "subject": "Dinner at Patsy's this weekend?"
```

```
    "date": "Tue, 5 Mar 2020 15:11:45 -0800 (PST)",
    "msg_content":  "Hey Frank,\n\nWould you like to meet at
Patsy's for dinner on Saturday night?  Their chicken parm is as
good as my mom's (and that's pretty good!).\n\nRafael"
},
{
 ...
}
]]
```

[00191] In some embodiments, the pay network server may store the aggregated email data results, e.g., 3720, in an aggregated database, e.g., 3710a.

[00192] FIGURE 38 is a block diagram illustrating an example structure of a distributed linking node mesh, in one embodiment of the MMCDB. In one embodiment, the linking node mesh may be represented as a modified graph data structure that contains nodes for entities and edges that represent the associations between the nodes. In one embodiment, the nodes are actual observable entities, such as a user 3801, a business 3803, an item 3807, a review on an online web site, e.g., 3813, 3816, and/or the like. The graph mesh may also contain deduced entities that have been inserted by the MMCDB as a result of the aggregation described herein, such as the aggregation of data from email, search queries, location aggregation, and/or the like. Non-limiting examples of deduced entity nodes that may be inserted into the graph mesh include a deduced item, i.e., 3810 or a deduced opportunity, i.e., 3805. A deduced item may be an item that the mesh determines exists based on scanning emails, e.g., 3809, to determine that a concept that occurs with some frequency in emails is associated with a concept that is not yet linked into the mesh graph. An example deduced item may be a user's mother's chicken parmesan 3810. In one embodiment, there may also be deduced opportunities added to the mesh graph, e.g. 3805. A deduced opportunity may be an opportunity that is determined based on aggregated transaction data, e.g., 3804. For example, through the use of aggregated transaction data it may be determined by the MMCDB that the price of a given set of items declines at a restaurant, e.g., 3803, during the week, e.g., 3805. This decline in pricing may then allow the MMCDB to determine that there exists a special weekday menu with lower

prices. In so doing, an opportunity for use by the mesh in providing recommendations or information to users may be created, e.g., 3805, in order to facilitate searching the mesh.

[00193] In one embodiment, the mesh graph may also contain service items, e.g., 3807, such as a restaurants chicken parmesan or other menu item. The service item and its link to the business 3803, e.g., 3806, 3808, may be determined using a forward web crawl (such as by crawling from a business home page to its menu pages), or by a reverse web crawl, such as by crawling using an Optical Character Recognition scanned menu forwarded through an email exchange and aggregated by an email aggregating component of the MMCDB.

[00194] In one embodiment, the mesh graph may additionally contain meta concepts, e.g., 3810, 3812, 3815. Meta-concepts are conceptual nodes added to the graph by MMCDB that define not a specific entity (such as a user or a business) nor a specific deduced entity (such as a deduced item or a deduced opportunity), but rather indicate an abstract concept to which many more nodes may relate. For example, through web crawling, e.g., 3814, or email aggregation, e.g., 3817, user reviews may be imported as nodes within the mesh graph, e.g., 3813, 3816. Nodes may be anonymous, e.g., 3813, linked to a specific user's friend (such as to provide specific user recommendations based on a social graph link), e.g., 3816, and/or the like. These reviews may be analyzed for positive concepts or words such as "delightful meal" or "highly recommended" and thereafter be determined by the MMCDB to be a positive review and linked to a mesh meta-concept of the kind positive review, e.g., 3815. In so doing, the MMCDB allows disparate aggregated inputs such as email aggregation data, location aggregation data, web crawling or searching aggregated data, and/or the like to be used to roll up concepts into conceptual units.

[00195] In one embodiment, these conceptual meta concepts, e.g., 3815, may be further linked to actual items, e.g., 3807. In so doing connections can be formed between real world entities such as actual reviews of items, to meta-concepts such as a positive review or beneficial location, and further linked to actual items as a location. Further meta-concepts may include activities such as dinner, e.g., 3812, a non-entity specific item (e.g., not a restaurant's chicken parmesan and not a mother's chicken

parmesan, but chicken parmesan as a concept), e.g., 3811. The connection of actual entity nodes with deduced entity nodes and meta-concept nodes allows the mesh to answer a virtually limitless number of questions regarding a given nodes connections and probable outcomes of a decision.

[00196] In one embodiment, nodes within the mesh graph are connected by edges that have no magnitude. In another embodiment, the edges themselves may have meta-data associated with them that enable faster or better querying of the mesh. Example meta data that may be stored at a graph edge include a relative magnitude of connection between nodes, path information regarding other nodes available from the edge, and/or the like. In still other embodiments, intermediate or link nodes, e.g., 3804, 3806, 3808, 3814, 3817, 3809, may be inserted by the MMCDB into the mesh graph. These intermediate nodes may function as the equivalent of an edge, in that they may describe a relationship between two nodes. In one embodiment, the link nodes may contain information about the nodes that they connect to. In so doing, the number of nodes in the graph that need to be searched in order to find a given type, magnitude or value of connection may be reduced logarithmically. Additionally, the link nodes may contain data about how the relationship between the nodes it links was established, such as by indicating the connection was established via search aggregation, email aggregation, and/or the like.

[00197] In one embodiment, the distributed linking node mesh may be stored in a modified open source database such as Neo4j, OrientDB, HyperGraphDB, and/or the like. An example structure substantially in the form of XML suitable for storing a distributed linking node mesh is:

```
<mesh>
  <nodes>
    <node id="1" kind="entity" type="user">
      <name="John Doe">
    </node>
    <node id="2" kind="entity" type="item">
      <name="iPhone" />
    </node>
    <node id="3" kind="deduced_entity" type="business">
```

```
        <name="Apple Computer" />
        <attribute type="keyword" value="iPhone" />
        <deduced_from value="frequency_keyword" />
      </node>
      <node>
        ...
      </node>
    </nodes>
    <link_nodes>
      <linknode id="78" in_node="1" out_node="3" value="55" />
      <linknode id="97" in_node="1" out_node="2" value="124" />
      ...
    </link_nodes>
    <edges>
      <edge from_node="1" to_node="78" />
      <edge from_node="78" to_node="3" />
      ...
    </edges>
  </mesh>
```

[00198] An example query suitable for querying a distributed linking node mesh is:

```
START user=node(5,4,1,2,3)
MATCH user-[:affinity]->"iphone"
WHERE entity.manufacturer =~ 'Apple.*', link.strength >= 40
RETURN user, user.affinity
```

[00199] In another embodiment, an example query suitable for querying a distributed linking node mesh is:

```
##MODEL QUERY Language (JSON FORMAT)
{
  1: {'LOWER': 100, 'BASETYPE': ['MODEL_001_001_00',
'MODEL_002_001_00', 'MODEL_003_001_00', 'MODEL_004_001_00'] ,
'attribute': 'WEIGHT', 'rule': 'NEAR', 'OP': 'PROX', 'type':
'TOKENENTITY', 'HIGHER': 100}
, 2: {'type': ['USER', 'MCC'], 'rule': 'FOLLOW'}
```

```
, 3: {'rule': 'RESTRICTSUBTYPE', 'BASETYPE': ['MODEL_001_001_00',
'MODEL_002_001_00', 'MODEL_003_001_00', 'MODEL_004_001_00']}}
}
```

[00200] FIGURES 39A-E are an example block diagram illustrating a distributed linking node mesh search, in one embodiment, of the MMCDB. The graph presented in Fig. 39A-E is similar to the graph presented in Fig. 38 but nodes of different type are represented similarly for ease of understanding. In one embodiment, a user 3901 may desire to find a good deal on dinner with friends at a restaurant nearby. The MMCDB may be configured with a capability to extract sub-concepts from a natural form query question, such as by natural language processing. Example tools suitable for this type of processing may include OpenNLP, Graph Expression, FreeLing, and/or the like.

[00201] In one embodiment, the query portion relating to finding a good deal is performed as a MMCDB search to arrive arrive at a result of a deduced opportunity for lower prices during weekdays, e.g., 3902. The search may then progress to extract the concept of a good deal merged with a restaurant nearby. Using an integrated location capability of a user's device, the user's current location may additionally be provided to the MMCDB for use in this portion of the query process, to produce a result containing a deduced opportunity for lower prices (e.g., a "good deal") at a business nearby wherein the lower prices are linked to the business nearby with a certain degree of weight, e.g., 3903. In one embodiment, the search may progress to find results for the concept of a dinner (e.g., meta-concept dinner 3904), which is itself linked through intermedia nodes to the business found in the previous portion of the search, e.g., 3905. In one embodiment, the search may then progress to find connections that indicate that the user 3901 will like the restaurant, e.g., 3906, and that the user's friends will similarly like the restaurant, e.g., 3907. The intermediate searches performed may be then merged to produce a unitary result, e.g., 3908, for a restaurant meeting the full criteria. In cases where no single entity meets all the criteria, the most important criteria to a user may be first determined using its own MMCDB search, such as a search that determines that a user 3901 has never traveled to a nearby popular location area for dinner and therefore concluding that location is very important to the user. In one

embodiment, multiple results 3908 may be returned and ranked for acceptability to both the user and his/her friends, enabling the user to then choose a preferred location.

[00202] FIGURE 39F shows an alternative embodiment of a distributed linking node mesh search. Here, mesh user 3901 wants to determine the probability that a user will buy Korean BBQ before a certain time, e.g., 3909. The distributed linking mesh may be queried. For example, the user's previous purchases of Korean BBQ (e.g., 3910, 3911), may be linked to a meta-concept that indicates an affinity for Korean BBQ, e.g., 3911. The affinity may be similarly linked to an entity indicating a purchase frequency for Korean BBQ, e.g., 3912. Similarly, by aggregating data from the user's email correspondence (i.e., calendar updates, and/or the like), the mesh may have an entity representing the user's schedule including free time, e.g., 3913. Both the purchase frequency 3912 and the user schedule 3913 may be linked to a mesh meta-concept of free time, e.g., 3914, which may indicate that the entities are related when the user has free time (e.g., the individual may be more likely to go for Korean BBQ when she is not working). By querying the distributed linking node mesh for interrelations between entities built from aggregation techniques (and deduced or input entities), a profile of the user's future behavior may be similarly built. For example, if the user schedule indicates that the user is free on both Wednesday and Thursday afternoons, and the aggregated purchase history indicates an affinity to purchase Korean BBQ both on those days (based on the purchase transaction entities) and when the user is free (based on the meta-concept of free time), then a mesh search can return a probability based on the respective weights of the constituent entity relationships with respect to the user.

[00203] FIGURES 40A-C are an example block diagram illustrating index creation in a distributed linking node mesh, in one embodiment of the MMCDB. In one embodiment, a non-indexed graph is exploded to form a chain of relationships from a single entity's perspective, e.g., 4001. A furthest node graph traversal is then performed, e.g., 4002, whereby the linking nodes are sequentially removed and replaced with a single edge which has a magnitude of the connection between the two nodes, e.g., 4002a, 4002b. A nearest node graph traversal may then be performed, e.g., 4003, whereby the magnitude of links further from the nearest node is modified by a factor of previous links. Modification proceed from nearest to furthest nodes, e.g., 4003a. In the

example illustrated, a modification is made to the second edge encountered to make its value as a relation of magnitude with User X incident on both the relationship of User X to Business Y and of Business Y to Deduced Opportunity L. This procedure may produce a flattened graph from a single entity's perspective, e.g., 4004. The graph may then be further modified to a single perspective indexed graph, e.g., 4005, to reduce the number of hops in the graph from a given entity to any other entity within the indexed graph, so as to advantageously speed searching of the graph from the entity's perspective, e.g., 4005a. In one embodiment, the output of similar indexes performed from other entity perspectives, e.g., 4006b, may be linked 4006c to the generated perspective 4006a. In so doing, the index may form a graph that simultaneously allows easy searching from the perspective of a single entity while maintaining connection between entities of different perspectives.

[00204] FIGURE 41 is an example block diagram illustrating aspects of an Encryptmatics XML converter component. In one embodiment, models may be input in a number of disparate language, e.g., 4101. Languages may include interpreted languages such as Python, PHP, and/or the like, e.g., 4102, intermediate compiled languages such as .Net, Java, and/or the like, e.g., 4103, compiled languages such as C++, COBOL, and/or the like, e.g., 4104. A user defined language may also be input, e.g., 4105. In one embodiment, the user defined language will be input with a language mapper, e.g., 4106 that defines a mapping of the user defined language's functions, methods, supported types, and/or the like to a language known by the MMCDB. In still other embodiments, a native meta-data language, e.g., 4107, may be input.

[00205] In one embodiment, languages other than a native meta-data language are passed to a meta-data language conversion component 4108, such as an Encryptmatics XML converter. The converter may convert the language to a meta-data language 4109. In one embodiment, the meta data language may describe data sources 4110 including a private data store (not available to the provided model), an anonymized data store that is based on the private data store (available to the provided model), and/or a public data store. In one embodiment, the meta-data language may be deconverted 4111 to produce data queries and model logic 4112 that is parseable by the MMCDB interpreter.

[00206] FIGURE 42 is an example logic flow illustrating input language loading by an Encryptmatics XML converter component, in one embodiment of a MMCDB. In one embodiment, an input language definition is received, e.g., 4201. The language definition may be a file containing information about the functions available within the input language. In one embodiment, the language definition is source code for the language to be loaded into the MMCDB. In still other embodiments, the language definition is an executable binary file suitable for the MMCDB to issue sample commands and receive output. In one embodiment, the current mesh language definition may be retrieved 4202. The mesh language may be an XML based meta-data language that allows the description of data sources, data source manipulations (both visible and not visible to the input model) and a model to be run against the data sources. A model may be a series of logic commands describing manipulations or conditional logic to apply to input or source data sources in order to reach a result. In one embodiment, language loading may facilitate the user providing the description of data sources, data source manipulations, the model, and/or the like in a language with which the user is already familiar. The MMCDB may then used a loaded language definition to convert the language to a common meta-data based (e.g., XML based, JSON based, and/or the like) language with which to then parse and execute commands from.

[00207] In one embodiment, the first unprocessed mesh language operation is extracted from the mesh language definition. An example operation may be "TRIM", which may strip whitespace from the beginning and end of an input string. A determination is made if the mesh operation has an equivalent operation in the input language, e.g., 4204. Such a determination may be made by executing a sample command against the input binary and observing the output to determine if an error occurred. In other embodiments, a publically available language definition web site may be crawled to determine which function(s) within an input language likely map to the mesh operation equivalent(s). In some instances, there will be a one-to-one mapping between the input language and the meta-data based mesh language. If there is not a one-to-one equivalence, e.g., 4205, a determination is made (using a procedure similar to that employed above) to determine if a combination of input language functions may

equate to a mesh language operation, e.g., 4206. For example, an input language that supports both a left-trim (strip space to left of string) and a right-trim operation (strip space to right of string) may be considered to support a mesh TRIM through a combination applying both the left-trim and right-trim operations, producing a substantially equivalent output result.

[00208] In one embodiment, if no matching combination is found, e.g., 4207, the mesh operation may be marked as unavailable for the input language, e.g., 4208 and the next unprocessed mesh operation may then be considered. If a matching combination is found, e.g., 4207, an upper bound test may be employed to test the upper bound behavior of the input language operation and compare that to the upper bound behavior of an equivalent mesh operation, e.g., 4209. For example, some languages may perform floating point rounding to a different degree of precision at upper bounds of input. By testing this case, a determination may be made if the equivalent input language function will produce output equivalent to the mesh operation at upper bounds. In one embodiment, a lower bound test may be employed to test the lower bound behavior of the input language operation and compare that to the lower bound behavior of an equivalent mesh operation, e.g., 4210. For example, some languages may perform floating point rounding to a different degree of precision at lower bounds of input. By testing this case, a determination may be made if the equivalent input language function will produce output equivalent to the mesh operation at upper bounds. In one embodiment, other custom tests may then be performed that may be dependent on the mesh operation or the input language operation(s), e.g., 4211. If the results of the test cases above produce output that is different than the expected output for the equivalent mesh operation, e.g., 4212, an offset spanning function may be generated to span the difference between the languages. For example, in the example above if the rounding function in the input language is determined to produce different behavior than the equivalent mesh operation at a lower bound, a function may be provided in the input or mesh language to modify any output of the given input language operations to create an equivalent mesh language operation output. For example, a given floating point number may be rounded to a given level of significant digits to produce equivalent behavior.

[00209] In one embodiment, the offset spanning function may not be capable of completely mapping the input language operation(s) to the mesh language operation, e.g., 4214. In one embodiment, previous versions of the mesh language definition, e.g., 4215, may then be tested using a procedure substantially similar to that described above to determine if they may completely map the input language, e.g., 4216. If the previous version of the mesh language definition completely maps the input language, the mesh language definition version for the input language may be set to the previous version, e.g., 4217. For example, a previous version of the mesh language definition may contain different capabilities or function behaviors that allow it to completely map to an input language. If previous versions of the mesh input language do not completely map to the input language, language clipping parameters may be generated, e.g., 4218. Language clipping parameters are input limitations that are placed on an input language such that any inputs within the input limitations range will produce compliant mesh operation output. Inputs outside that range may generate an error. In one embodiment, language clipping parameters may include limits to the upper bound or lower bound of acceptable input. Such limits may be determined by iteratively testing increasing or decreasing inputs in order to find an input range that maps completely to the mesh operation.

[00210] In one embodiment, the current mesh operation, input language operation(s) any spanning functions or language clipping parameters, the mesh language version, and/or the like may be stored in an input language definition database, e.g., 4219. If there are more unprocessed mesh language operations, e.g., 4220, the procedure may repeat.

[00211] FIGURES 43A-B show an example logic flow for input model conversion, in one embodiment of an MMCDB. In one embodiment, a language command file is received, e.g., 4301. The language command file may contain instructions in any language which has been loaded into the MMCDB (e.g., Fig. 42). The input language command file may contain instructions that may describe a set of manipulations that may be performed on a data set (e.g., a data set that is input as part of the input language command file, a data set that is loaded from an external data source, and/or the like). In one embodiment, input language definitions corresponding to the language of the input language command file is retrieved, e.g., 4302. A mesh language definition,

which may specify operations that are available within the mesh language, may also be retrieved, e.g., 4303. Non-conditional logic flow blocks in the input language command file may be determined, e.g., 4304. A non-conditional logic block represents the outermost logic contained within an input language command file. For example, if a file contains no conditional logic (i.e., no if/than/else blocks, and/or the like), then the outermost logic may be the complete set of input language commands themselves. In one embodiment, a run block is created for each outermost non-conditional logic flow block. The metadata run blocks are then populated with logic commands further in the procedure. In one embodiment, any variables that are initialized within the logic block corresponding to the run block are determined, e.g., 4306. A variable initialization template may then be determined, e.g., 4307. In one embodiment, the input language definition is used to determine if an equivalent meta-data based variable type is available in the mesh language definition for each of the variables initialized in the input language command file, e.g., 4308. If all variable types are not available, a model input error may be raised, e.g., 4309.

[00212] In one embodiment, the variable initialization template and the input language definition are used to create a constants block based on the variable initialization template, e.g., 4310. Within the constants block, any constants that were included in the input language command file may be stored as structured XML. An example constants block, substantially the form of XML is as follows:

```
<constant>
  indexname="0"
  value='row by row'
  Type="string"
</constant>
```

[00213] In one embodiment, there may be multiple constant blocks defined corresponding to multiple constant values in the input language command file. In other embodiments, constants may be collapsed to one block.

[00214] In one embodiment, the input datasources may then be determined based on the input language command file, e.g., 4311. For example, an input datasource may

be defined directly in the input language command file (such as by declaring a variable as an array to values in the input language command file). In other embodiments, the inputs may be external to the input language command file, such as a third party library or loaded from an external source file (such as a comma delimited file, via a SQL query to an ODBC compliant database, and/or the like). A mesh language input datasource template may then be retrieved, e.g., 4312, to provide a structure to the MMCDB to use in formatting the inputs as meta-data. The datasources may be scanned to determine if they are available to the model (such as by executing "ls –l" on a POSIX compliant Unix system), e.g., 4313. If the datasources are available to the model, then a meta data language input block may be created using the input datasource template, the language definition, and the input language command file, e.g., 4314. An example input block substantially in the form of XML is:

```
<input>
  keyname="test_by"
  file="<ecyptmatics install>/test_by.egd"
  format="ecdataformat"
  meta_data={'col8': (7, 0, 'string'), '__header': True, 'col_2':
(1,0,'int'), 'col_3': (2,0,'int'), 'col_1': (0,0,'int'), 'col_6':
(5,0,'julian'), 'col_7': (6,0,'float'), 'col_4':
(3,0,'ordinaldate'), 'col_5': (4,0,'date')}
</input>
```

[00215] In one embodiment, a mesh language output template is determined, e.g., 4315 and an output block is created using a procedure substantially similar to that described above with respect to the constant and input blocks, e.g., 4316. An example output block, substantially in the form of XML is:

```
    <output>
    keyname="myout"
    file="stdout"
    format="deliminated"
    meta_data={'col_2': (2, 0, 'String'), 'col_2_1': (4, 0,
'String'), 'test': (0, 0, 'String'), 'col_3': (3, 0, 'String'),
'col_1': (1, 0, 'String'), 'sum_col_7': (5, 0, 'String')}
```

```
    deliminator= "csv"
    </output>
```

[00216] In one embodiment, the constant block, input block, and output block are added to a newly created initialization block and the initialization block is added to the current run block, e.g., 4317. An example run block with a complete initialization block included therein, substantially in the form of XML is as follows:

```
<run>
<init>
processor=process
    <input>
     keyname="test_by"
     file="<ecryptmatics install>/test/data/test_by.egd"
     format="ecdataformat"
     deliminator="csv"
      meta_data={'col_8': (7, 0, 'string'), '__header': True,
'col_2': (1, 0, 'int'), 'col_3': (2, 0, 'int'), 'col_1': (0, 0,
'int'), 'col_6': (5, 0, 'julian'), 'col_7': (6, 0, 'float'),
'col_4': (3, 0, 'ordinaldate'), 'col_5': (4, 0, 'date')}
    </input>
    <output>
    keyname="myout"
    file="stdout"
    format="deliminated"
    meta_data={'col_2': (2, 0, 'String'), 'col_2_1': (4, 0,
'String'), 'test': (0, 0, 'String'), 'col_3': (3, 0, 'String'),
'col_1': (1, 0, 'String'), 'sum_col_7': (5, 0, 'String')}
    deliminator= "csv"
    </output>
    <constant>
     indexname="0"
     value='row by row'
     type="string"
    </constant>
</init>
</run>
```

[00217] In one embodiment, a vault block will then be created, e.g., 4318. A logic command block will be extracted from the input logic command file, e.g., 4319. A logic command block is a logic block that is a non-outermost non-conditional logic flow. A door block may then be added to the vault block, e.g., 4320. A logic command, representing a discrete logic operation, may then be extracted from the logic command block, e.g., 4321. The logic command may be a tumbler, e.g., 4322, in which case a tumbler key may be looked up in a tumbler database and the tumbler may be processed, e.g., 4323. Further detail with respect to tumbler processing may be found with respect to Figs. 44-45. The logic command may then be mapped to a mesh language equivalent by using the language definition file, e.g., 4324. A mesh template logic command template, containing formatting information for a logic command, may be retrieved, e.g., 4325. In one embodiment, a lock block may be created using the mesh language definition, the language definition, and the logic command, e.g., 4326. The created lock block may be added to the current door block, e.g., 4327. In one embodiment, if there are more logic commands, e.g., 4328, the procedure may continue. If there are more logic command blocks, e.g., 4329, the procedure may similarly continue. In one embodiment, if there are more outermost non-conditional logic flow blocks in the input language command file, e.g., 4330, the procedure may continue with respect to Fig. 43A.

[00218] FIGURE 44 is an example block diagram illustrating a tumbler data source manipulation and anonymization component, e.g., a TDS Component. In one embodiment, a user model may call a tumbler as part of a logic command block processing (e.g., in order to perform a hash table lookup, to provide third-party data, to import anonymized transaction data, and/or the like). In one embodiment, portions of the data manipulation may not be visible, e.g., 4401, to the user model in order to maintain privacy for the record owners, to preserve business secrets, and/or the like. In one embodiment, the data source to be anonymized is loaded into a key/value table, e.g., 4402. The entire matrix may be considered as a tumblar key. In other embodiments, a single cell within the matrix may be a tumblar key. In still other embodiments, the matrix may take the form of a n x n matrix of arbitrary size (e.g., a 4x4x4x4 matrix, and/or the like) In one embodiment, the keys or values may be pointers to underlying data records. In another embodiment, the keys or values may themselves be the data for

manipulation. Commands may be read from the tumblar file (which may, in some embodiments, have a format substantially similar to an input language command file, e.g., 4301). The commands may change some values in the matrix to other values, such as may be done to anonymize user payment card information, e.g., 4403. In other embodiments, data may be removed from the matrix and replaced with other data values, e.g., 4404. When indicated by the tumblar file, when a set number such as 5 anonymization operations have been performed, or when the tumblar key has reached a certain value, the tumblar key may be considered visible to the user model, e.g., 4405. In so doing, the current keychain may be visible to the user model, e.g., 4407. Additional operations may then be performed on the key, extending the keychain, e.g., 4408. A keychain is a representation of current and past values of a key/value store. In one embodiment, the keychain 4409 may be returned. The keychain may contain an n x n sized matrix (i.e., a single 2D matrix, a 3D collection of 2D matrix, a 4D matrix, and/or the like), e.g., 4409a, 4409b.

[00219] In one embodiment, a tumblar file may be substantially in the form of XML as follows:

```
<xml>
<run>
<init>
processor=process
tumblar_name=None
tumblar_path=c:\1m\Ecryptmatics\ecryptmatics\test\tumblars\flare
tumblarkey=flare
    <input>
     keyname="flares"
     file="<ecryptmatics install>/test/data/flare.datal"
     format="deliminated"
     deliminator=" "
      meta_data={'evolution': (4, 0, 'Evolution'),
'prev_activity': (5, 0, 'Previous 24 hour flare activity code'),
'area': (8, 0, 'Area'), 'are_largest_spot': (9, 0, 'Area of the
largest spot'), 'histocially_complex': (6, 0, 'Historically-
complex'), 'complex': (7, 0, "Did region become historically
```

```
complex  on this pass across the sun's disk"), 'class_cd': (0, 0,
'Code for class (modified Zurich class)'), 'activity': (3, 0,
'Activity'), 'spot_dict_cd': (2, 0, 'Code for spot
distribution'), 'largets_spot_cd': (1, 0, 'Code for largest spot
size'), '__header': False}
    </input>
    <output>
    keyname="myout"
    file="stdout"
    format="deliminated"
    meta_data={'evolution': (3, 0, 'String'), 'area': (1, 0,
'String'), 'complex': (2, 0, 'String'), 'activity': (0, 0,
'String')}
    deliminator= "csv"
    </output>
</init>
<vault>
    <door>
        <lock>
     outkey="myout"
     outkeyname="activity"
     inkey="flares"
     inkeyname="activity"
     function="tumble"
     type="String"
     tumblar-masks="*"
     fnc-tumblar-key-table="flare.activity"
     tumblar-default="None"
        </lock>
        <lock>
     outkey="myout"
     outkeyname="area"
     inkey="flares"
     inkeyname="area"
     function="tumble"
     type="String"
     tumblar-masks="*"
```

```
        fnc-tumblar-key-table="flare.area"
        tumblar-default="None"
           </lock>
           <lock>
        outkey="myout"
        outkeyname="complex"
        inkey="flares"
        inkeyname="complex"
        function="tumble"
        type="String"
        tumblar-masks="*"
        fnc-tumblar-key-table="flare.complex"
        tumblar-default="None"
           </lock>
           <lock>
        outkey="myout"
        outkeyname="evolution"
        inkey="flares"
        inkeyname="evolution"
        function="tumble"
        type="String"
        tumblar-masks="*"
        fnc-tumblar-key-table="flare.evolution"
        tumblar-default="None"
           </lock>
      </door>
  </vault>
  </run>
  </xml>
```

[00220] FIGURE 45 is an example logic flow showing a tumblar data source anonymization component, e.g., a TDS component, in one embodiment of a MMCDB. In one embodiment, a user unaccessible data source request and a user generated model containing tumblar data source manipulations may be received, e.g., 4501. In one embodiment, a tumblar key may be extracted, e.g., 4502. A tumblar definition corresponding to the tumblar key may be retrieved from a tumblar database, e.g., 4503.

A tumblar definition may contain manipulations (e.g., functions, methods, and/or the like) that may be performed on a given source file before the data is made available for use in a user model. In one embodiment, an input/starting key name may be determined (e.g., by inspecting an init block or by inspecting the input key values in the first lock of the first door of the first vault in the first run block in the tumblar file), e.g., 4505. An unprocessed internal tumblar data operation may be extracted including an input and an output key, e.g., 4506. An internal tumblar operation may be an operation that is performed before a user model has access to the data store, such as data manipulations that anonymize data. Manipulation operations may include bit shifting, replacing or masking certain field values in the data set, swapping data values between records (such as may be done to maintain a total of all values or the average of all values while not revealing to the user model the underlying data). In one embodiment, the current map located at the input key may be duplicated and stored, e.g., 4507. The operation may then be performed on the data copy, e.g., 4508. In so doing, a chain (e.g., a key chain) of values may be created for a single data point. If the current output key is visible to the user model (such as if the output key is > a given value such as 31, the output has undergone a given number of operations, and/or the like), e.g., 4509, then any maps equal to or greater than the current map may be marked as visible to the user model, e.g., 4510. Manipulation operations may continue on the data and an unprocessed external tumblar data operation (e.g., an operation visible to the user model) may be extracted, e.g., 4511. The current map may be duplicated, e.g., 4512, and stored as a new map also visible to the user model, e.g., 4512. In one embodiment, the external tumblar data operation may then be applied to the copied map, e.g., 4513. If there are no more processed external tumblar data operations, e.g., 4514, the user model visible portion of the keychain may be returned, e.g., 4515.

[00221] FIGURE 46 is an example data flow illustrating mesh aggregation and cluster querying, in one embodiment of a MMCDB. In one embodiment, a firehose server 4601 provides firehose input, e.g., 4602 to a mesh server 4603. A firehose server may be a server capable of accepting input from one or more data sources (e.g., Twitter or Facebook posts, transaction records, email data, and/or the like) at a high relative flow rate. In one embodiment, the firehose server may perform some manipulations on

the received data before it is input to the mesh server 4603. An example firehose input 4602, substantially in the form of XML formatted data is:

```
<firehose_input>
  <input type="email" id="1">
    <dictionary_entry>
        {id: "1h65323765gtyuf#uy76355",
         type: email,
         category: {cat1: "food", cat2: "dinner"},
         from_addr: "john.doe@gmail.com",
         to_addr: "jane.doe@gmail.com",
         subject: "Korean BBQ this weekend?",
         dictionary_keywords: "Korean, dinner, nyc",
         content_hash: "7m865323476feeaniiji"}
    </dictionary_entry>
    <datetime>Jan 20, 2020 15:23:43 UTC</datetime>
    <from_addr>john.doe@gmail.com</from_addr>
    <to_addr>jane.doe@gmail.com</to_addr>
    <subject>Korean BBQ this weekend?</subject>
    <content>
Received: by 10.36.81.3 with SMTP1 id e3cs239nzb; Tue, 5 Mar 2020
15:11:47 -0800 (PST)
Return-Path:
Received: from mail.emailprovider.com (mail.emailprovider.com
[111.111.11.111]) by mx.gmail.com with SMTP id
h19si826631rnb.2005.03.29.15.11.46; Tue, 5 Mar 2020 15:11:47 -
0800 (PST)
Message-ID: <20050329231145.62086.mail@mail.emailprovider.com>
Received: from [11.11.111.111] by mail.emailprovider.com via
HTTP; Tue, 5 Mar 2020 15:11:45 PST
Date: Tue, 5 Mar 2020 15:11:45 -0800 (PST)
From: John Doe <john.doe@gmail.com>
Subject: Korean BBQ this weekend?
To: Jane Doe <jane.doe@gmail.com>
Hi Jane,
Would you like to meet up in New York city this weekend for
Korean BBQ?  I know this great place down on Spring Street.
```

```
John
      </content>
    </input>
    <input type="tweet" id="2">
      ...
    </input>
    <input type="purchase_transaction" id="3">
      ...
    </input>
    <input type="web_search" id="4">
      ...
    </input>
    <input id="n">
      ...
    </input>
  </firehost_input>
```

[00222] In one embodiment, the mesh structure may then be updated, e.g., 4604. Further detail regarding updating the mesh structure can be found throughout this specification, drawing and claims, and particularly with reference to Figs. 15-19. In one embodiment, a clustering node 4605 may send a cluster categories request 4606 to the mesh server. A cluster categories request may contain a category or deduced concept that is to be added to the mesh. In one embodiment, the category may have no pre-existing associations in the mesh (e.g., the category to be added may be an orphan category). An example cluster categories request 4606, substantially in the form of an HTTP(S) POST message including XML is:

```
POST /cluster_categories.php HTTP/1.1
Host: www.meshserver.com
Content-Type: Application/XML
Content-Length: 667
<?XML version = "1.0" encoding = "UTF-8"?>
<cluster_categories_request>
  <cluster operation="add">
    <concept value="iphone" />
    <concept_related_concept value="apple" />
```

```
      <concept_keyword value="64GB" />
      <concept_keyword value="Steve Jobs" />
    </cluster>
    <cluster>
      ...
    </cluster>
  </cluster_categories_request>
```

[00223] In an alternative embodiment, an example cluster categories request 4606, substantially in the form of an HTTP(S) POST message including XML is:

```
POST /cluster_categories.php HTTP/1.1
Host: www.meshserver.com
Content-Type: Application/XML
Content-Length: 667
<?XML version = "1.0" encoding = "UTF-8"?>
<cluster_categories_request>
  <cluster operation="add">
    <concept value="portable music player" />
    <manufacturer>Apple Computer</manufacturer>
    <model>iPod Touch 32GB</model>
    <size>32GB</size>
  </cluster>
  <cluster>
    ...
  </cluster>
</cluster_categories_request>
```

[00224] In one embodiment, the cluster categories request above may be modified by the MMCDB as a result of aggregated data. For example, a request to create a cluster for an iPod of a given size may be supplemented with alternative models / sizes. In so doing, the mesh may expand a recommendation, graph entity, and/or the like to emcompass concepts that are connected with the primary concept. In one embodiment, this modified cluster may take the form a the form of XML substantially similar to:

```
  <cluster>
    <concept value="portable music player" />
```

```
      <manufacturer>Apple Computer</manufacturer>
      <model>
        <1>iPod Touch 32GB</1>
        <2>iPod Touch 64GB</2>
        <3>iPod Touch 128GB</3>
        <4>iPhone 32GB</4>
        <5>iPhone 64GB</5>
        <6>iPhone 128GB</6>
      </model>
      <size>32GB OR 64GB OR 128GB</size>
    </cluster>
```

[00225] In one embodiment, the mesh structure may be updated in response to the cluster categories request, e.g., 4604. In one embodiment, a user 4607 may use his/her mobile device to indicate that they wish to purchase an item based on cluster concepts, e.g., a user bid/buy input 4608. For example, a user may query "I want the TV that AV Geeks thinks is best and I'll pay $1,500 for it". In one embodiment, the query may be substantially in the form of a language input such as the above, which may be parsed using natural language processing packages such as FreeLing, LingPipe, OpenNLP, and/or the like. In other embodiments, the user may be presented with a structured query interface on their mobile device that allows a restricted set of options and values from which to build a bid/buy input 4608. For example, a user may be given a list of categories (such as may be built by querying a categories database as described with respect to Fig. 49) from which to choose when making a bid/buy input. In one embodiment, a clustering server 4609 may receive the user bid/buy input 4608 and generate a consumer cluster based bid request, e.g., 4610 and provide same to a clustering node 4605. An example consumer cluster based bid request 4610, substantially in the form of an HTTP(S) POST message including XML is:

```
POST /consumer_bid_request.php HTTP/1.1
Host: www.meshserver.com
Content-Type: Application/XML
Content-Length: 667
<?XML version = "1.0" encoding = "UTF-8"?>
<consumer_cluster_based_bid_request>
```

```
    <datetime>Jan 21, 2020 5:34:09 UTC</datetime>
    <user_id>43246</user_id>
    <request>
      <type>bid</type>
      <item>
        <item_query>LCD Television</item_query>
        <type_desired value="best" />
        <cluster_source value="AV Geeks.com" />
        <cluster_min_expertise_level value="top2prct" />
        <max_price value="1500.00" currency="USD" />
        <expire_request value="30days" />
        <payment type="credit">
          <card_type>VISA</card_type>
          <card_num>9876543598766732</card_num>
          <card_exp>0525</card_exp>
        </payment>
        <shipping_address>
          <addr1>100 Main St.</addr1>
          <city>Anytown</city>
          <state>CA</state>
          <zip>90145</zip>
        </shipping_address>
      </item>
    </request>
  </consumer_cluster_based_bid_request>
```

[00226] In an alternative embodiment, the consumer cluster based bid request may be generated using the user interface described herein and with respect to Fig. 48A-B. In one embodiment, the consumer cluster based bid request 4610, generated using the interface may be substantially in the form of an HTTP(S) POST message including XML:

```
  POST /consumer_bid_request.php HTTP/1.1
  Host: www.meshserver.com
  Content-Type: Application/XML
  Content-Length: 667
  <?XML version = "1.0" encoding = "UTF-8"?>
```

```
<consumer_cluster_based_bid_request>
  <datetime>Jan 21, 2020 5:34:09 UTC</datetime>
  <user_id>43246</user_id>
  <request>
    <type>bid</type>
    <item>
      <item_query>headphones</item_query>
      <quantity value="2" />
      <requirement value="rated_top_3" />
      <cluster_source value="consumerreports.com" />
      <max_price value="249.95" currency="USD" />
      <expire_request value="January 15, 2020" />
      <payment type="credit">
        <card_type>VISA</card_type>
        <card_num>9876543659876732</card_num>
        <card_exp>0525</card_exp>
      </payment>
      <shipping_address>
        <addr1>100 Main St.</addr1>
        <city>Anytown</city>
        <state>CA</state>
        <zip>90145</zip>
      </shipping_address>
    </item>
  </request>
</consumer_cluster_based_bid_request>
```

[00227] In one embodiment, in response to the consumer cluster based bid request 4610, the clustering node 4605 may generate a cluster request 4611. A cluster request may be a request to search the mesh in order to find results (e.g., items matching a cluster's buying habits, merchants offering an item, alternative items for purchase, friends that have already purchased items, items the user already owns – based on, for example, past purchase transactions – that may satisfy the request, and/or the like). An example query suitable for querying a distributed linking node mesh is:

```
START user=node(5,4,1,2,3)
```

```
MATCH entity-[:affinity]->"consumer_reports"
WHERE entity.recommended >= '3',
   entity.recommendation.item.type ~= "headphones"
RETURN entity.recommendation.item.name,
   entity.recommendation.item.model,
   entity.recommendation.item.averageprice
```

[00228] In one embodiment, the mesh server may provide a cluster request response 4612. An example cluster request response 4612 substantially in the form of an HTTP(S) POST message including XML is:

```
POST /cluster_request_response.php HTTP/1.1
Host: www.clusteringnode.com
Content-Type: Application/XML
Content-Length: 667
<?XML version = "1.0" encoding = "UTF-8"?>
<cluster_request_response>
  <requested_item>
    <item_query>LCD Television</item_query>
    <type_desired value="best" />
    <cluster_source value="AV Geeks.com" />
    <cluster_min_expertise_level value="top2prct" />
    <max_price value="1500.00" currency="USD" />
  </requested_item>
  <cluster_results>
    <num_users_meeting_cluster value="2541" />
    <average_user_feedback_ranking value="94%" />
    <cluster_user_purchases>
      <item rank="1">
        <desc>Sony Bravada 50" LCD 645</desc>
        <model>KDL50EX645</model>
      </item>
      <item rank="2">
        <desc>Sony Bravada 50" LCD 655</desc>
        <model>KDL50EX655</model>
      </item>
      <item>
```

```
            ...
          </item>
        </cluster_user_purchases>
      </cluster_results>
    </cluster_request_response>
```

[00229] In an alterative embodiment, an example cluster request response 4612 substantially in the form of an HTTP(S) POST message including XML is:

```
POST /cluster_request_response.php HTTP/1.1
Host: www.clusteringnode.com
Content-Type: Application/XML
Content-Length: 667
<?XML version = "1.0" encoding = "UTF-8"?>
<cluster_request_response>
  <requested_item>
      <item_query>headphones</item_query>
      <quantity value="2" />
      <requirement value="rated_top_3" />
      <cluster_source value="consumerreports.com" />
      <max_price value="249.95" currency="USD" />
      <expire_request value="January 15, 2020" />
  </requested_item>
  <cluster_results>
    <cluster_consumer_reports_ranking>
      <item consumer_reports_rank="1">
        <desc>Panasonic Technics Pro DJ</desc>
        <model>RP-DH1250</model>
        <avg_price>$235.55</avg_price>
      </item>
      <item consumer_reports_rank="2">
        <desc>Coby In Ear Headphones</desc>
        <model>CVEM76PNK</model>
        <avg_price>$245.55</avg_price>
      </item>
      <item consumer_reports_rank="3">
        <desc>Shure E2c-n Sound Isolating Earphones</desc>
```

```
            <model>SHE2CN</model>
            <avg_price>$249.95</avg_price>
          </item>
        </cluster_consumer_reports_ranking>
      </cluster_results>
    </cluster_request_response>
```

[00230] In one embodiment, the clustering node 4605 may then process the cluster response and create transaction triggers. Further details regarding cluster request response 4612 processing may be found throughout the specification, drawings and claims and particularly with reference to **Fig. 47**, e.g., a **CRA Component**.

[00231] In one embodiment, a lead cluster order request may be generated for merchants that were identified as a result of the cluster response analysis, e.g., 4613. In other embodiments, a default list of merchants may be used. A lead cluster order request may contain information relating to the identified purchase that the user 4607 wishes to engage in. In the example above, for example, the analysis may have determined that based on the aggregated AV Geeks user expert preference information, the user should purchase Sony television model KDL50EX645 or KDL50EX655. The analysis may also have determined that a given merchant sells those models of television (such as by using aggregated sales transaction data as described herein). A request may then be sent to the merchant indicating a purchase item, a user lead that may execute the purchase and a price the user is willing to pay. In one embodiment, the user identity is not provided or is anonymized such that the merchant does not have information sufficient to determine the actual identity of the user but may determine if they wish to execute the sale to the user. An example lead cluster order request 4614, substantially in the form of an HTTP(S) POST message containing XML data:

```
POST /lead_cluster_order_request.php HTTP/1.1
Host: www.merchantserver.com
Content-Type: Application/XML
Content-Length: 667
<?XML version = "1.0" encoding = "UTF-8"?>
<lead_cluster_order_request>
  <lead validFor="30_days">
```

```
      <type>television</type>
      <items join="OR">
        <item model="KDL50EX645" />
        <item model="KDL50EX655" />
      </items>
      <user_information>
        <name>John Doe</name>
        <email>john.doe@gmail.com</email>
        <phone>865-765-3465</phone>
      </user_information>
      <payment type="credit">
        <card_type>VISA</card_type>
        <card_num>98765436598766732</card_num>
        <card_exp>0525</card_exp>
      </payment>
      <shipping_address>
        <addr1>100 Main St.</addr1>
        <city>Anytown</city>
        <state>CA</state>
        <zip>90145</zip>
      </shipping_address>
    </lead>
    <lead>
      ...
    </lead>
  </lead_cluster_order_request>
```

[00232] In one embodiment, a merchant may accept the order and generate a lead cluster order accept/reject response. In other embodiments, the merchant may indicate that they wish to hold the lead opportunity open and may accept at a later time if no other merchant has filled the lead cluster order request. In still other embodiments, the merchant response may contain a counteroffer for the user (e.g., $1600), which the user may then accept or decline. In one embodiment, the user receives an order acceptance confirmation 4617 indicating that their order has been fulfilled.

[00233] In one embodiment, a user may cancel a cluster based bid request prior to the merchant fulfilling the order. For example, a user may transmit a user cancel input 4618 to clustering server 4609. The clustering server may forward the cancel message to the clustering node 4605, e.g., 4619, which may in turn forward the cancel message to the merchant(s) server 4615, e.g., 4620.

[00234] FIGURE 47 is an example logic flow illustrating cluster response analysis and transaction triggering, e.g., a CRA component, in one embodiment of a MMCDB. In one embodiment, a cluster request response is received, e.g., 4701. Cluster criteria (i.e., user requesting cluster, the criteria for the cluster, payment/shipping information for the user purchase bid, and/or the like) may be extracted from the cluster request response, e.g., 4702. In one embodiment, the cluster criteria is examined to determined if it meets the minimum cluster criteria, e.g., 4703. Examples of minimum cluster criteria include minimum feedback ranking of users in cluster, minimum years of expertise of users in cluster, median value of items returned, and/or the like. If the cluster criteria is not greater than the minimum cluster criteria, the user may be prompted to adjust the minimum criteria and a search may be re-run, e.g., 4704. In other embodiments, the criteria may be adjusted automatically by the MMCDB or a third-party database may be queried to determine new minimum criteria (e.g., a user expertise ranking service, a user review site, and/or the like).

[00235] In one embodiment, candidate purchase items may be extracted from the cluster request response, e.g., 4705. A merchant database may be queried to determine merchants selling the candidate purchase items. An example merchant database query, substantially in the form of PHP/SQL commands is provided below:

```
<?PHP
header('Content-Type: text/plain');
mysql_connect("localhost",$DBserver,$password);
mysql_select_db("merchants.sql");
$query = "SELECT merchant_id, merchant_name, price,
quantity_on_hand FROM merchants WHERE merchant_item_id LIKE '%'
$cluster_returned_model_num";
$result = mysql_query($query); // perform the search query
mysql_close("merchants.sql"); // close database access
```

?>

[00236] In one embodiment, a maximum price the user is willing to pay is determined, e.g., 4707. An average selling price of the candidate purchase items may be determine (such as by querying a merchant table containing price history, querying a price history table, performing a live crawl of a merchant's web site, and/or the like). If the user's maximum price is not within a given range of the average merchant item price, e.g., 4709, a price trend database may be queried, e.g., 4710. A price trend database may contain historical information relating to the price of an item over time. If the price trend (i.e., the linear extrapolation of the historical prices, and/or the like) shows that the average price of the item will be within 40% of the user's maximum price before the user purchase bid expires, e.g., 4711, the user purchase bid request may be held, e.g., 4712, so that the cluster response analysis may be re-run again before the bid expires. In another embodiment, even if the user's price will not be within a range of the average price of an item at the queried merchants, the user procedure may continue if the user has been marked as a high priority bid user (e.g., a frequent bidder, a new bidder, and/or the like), e.g., 4713. In one embodiment, the first merchant that has stock of the item may be selected, e.g., 4714. If the merchant has received greater than a set amount of bids in a time period, e.g., 4715, another merchant may be selected. In so doing, one merchant may not be overwhelmed with bids. In one embodiment, a lead cluster order request is created and transmitted to the merchant, e.g., 4716.

[00237] FIGURES 48A-C show user interface diagrams illustrating example aspects of a discovery shopping mode of a virtual wallet application in some embodiments of the MMCDB. In some embodiments, the virtual wallet application may provide a 'discovery shopping' mode for the user. For example, the virtual wallet application may obtain information on aggregate purchasing behavior of a sample of a population relevant to the user, and may provide statistical/aggregate information on the purchasing behavior for the user as a guide to facilitate the user's shopping. For example, with reference to FIGURE 48A, the discovery shopping mode 4801 may provide a view of aggregate consumer behavior, divided based on product category (see 4802). Thus, the virtual wallet application may provide visualization of the magnitude of

consumer expenditure in particular market segment, and generate visual depictions representative of those magnitudes of consumer expenditure (see 4803-4806). In some embodiments, the virtual wallet application may also provide an indicator (see 4809) of the relative expenditure of the user of the virtual wallet application (see blue bars); thus the user may be able to visualize the differences between the user's purchasing behavior and consumer behavior in the aggregate. The user may be able to turn off the user's purchasing behavior indicator (see 4810). In some embodiments, the virtual wallet application may allow the user to zoom in to and out of the visualization, so that the user may obtain a view with the appropriate amount of granularity as per the user's desire (see 4807-4808). At any time, the user may be able to reset the visualization to a default perspective (see 4811).

[00238] Similarly, the discovery shopping mode 4821 may provide a view of aggregate consumer response to opinions of experts, divided based on opinions of experts aggregated form across the web (see 4802). Thus, the virtual wallet application may provide visualizations of how well consumers tend to agree with various expert opinion on various product categories, and whose opinions matter to consumers in the aggregate (see 4823-4826). In some embodiments, the virtual wallet application may also provide an indicator (see 4829) of the relative expenditure of the user of the virtual wallet application (see blue bars); thus the user may be able to visualize the differences between the user's purchasing behavior and consumer behavior in the aggregate. The user may be able to turn off the user's purchasing behavior indicator (see 4830). In some embodiments, the virtual wallet application may allow the user to zoom in to and out of the visualization, so that the user may obtain a view with the appropriate amount of granularity as per the user's desire (see 4827-4828). At any time, the user may be able to reset the visualization to a default perspective (see 4831).

[00239] With reference to FIGURE 48B, in some implementations, the virtual wallet application may allow users to create targeted shopping rules for purchasing (see FIGURE 48A, 4812, 4822). For example, the user may utilize the consumer aggregate behavior and the expert opinion data to craft rules on when to initiate purchases automatically. As an example, rule 4841 specifies that the virtual wallet should sell the users iPad2 if its consumer reports rating falls below 3.75/5.0, before March 1, provided

a sale price of $399 can be obtained. As another example, rule 4842 specifies that the virtual wallet should buy an iPad3 if rule 4841 succeeds before February 15. As another example, rule 4843 specifies that the wallet should buy a Moto Droid Razr from the Android Market for less than $349.99 if its Slashdot rating is greater than 3.75 before February 1. Similarly, numerous rules with a wide variety of variations and dependencies may be generated for targeted shopping in the discovery mode. In some implementations, the virtual wallet user may allow the user to modify a rule. For example, the wallet may provide the user with an interface similar to 4846 or 4847. The user may utilize tools available in the rule editor toolbox to design the rule according to the user's desires. In some implementations, the wallet may also provide a market status for the items that are subject to the targeted shopping rules.

[00240] With reference to FIGURE 48C, in some implementations, the virtual wallet application may provide a market watch feature, wherein the trends associated with items subject to targeted shopping rules may be tracked and visually represented for the user. For example, the visualization may take, in some implementations, the form of a ticker table, wherein against each item 4851(A)-(E) are listed a product category or cluster of expert opinions to which the product is related 4852, pricing indicators, including, but not limited to: price at the time of rule creation 4852, price at the time of viewing the market watch screen 4853, and a target price for the items (A)-(E). Based on the prices, the market watch screen may provide a trending symbol (e.g., up, down, no change, etc.) for each item that is subject to a targeted shopping rule. Where an item satisfied the targeted rule (see item (E)), the virtual wallet may automatically initiate a purchase transaction for that item once the target price is satisfied.

MMCDB Controller

[00241] FIGURE 49 shows a block diagram illustrating embodiments of a MMCDB controller. In this embodiment, the MMCDB controller 4901 may serve to aggregate, process, store, search, serve, identify, instruct, generate, match, and/or facilitate interactions with a computer through various technologies, and/or other related data.

[00242] Typically, users, which may be people and/or other systems, may engage information technology systems (e.g., computers) to facilitate information processing. In turn, computers employ processors to process information; such processors 4903 may be referred to as central processing units (CPU). One form of processor is referred to as a microprocessor. CPUs use communicative circuits to pass binary encoded signals acting as instructions to enable various operations. These instructions may be operational and/or data instructions containing and/or referencing other instructions and data in various processor accessible and operable areas of memory 4929 (e.g., registers, cache memory, random access memory, etc.). Such communicative instructions may be stored and/or transmitted in batches (e.g., batches of instructions) as programs and/or data components to facilitate desired operations. These stored instruction codes, e.g., programs, may engage the CPU circuit components and other motherboard and/or system components to perform desired operations. One type of program is a computer operating system, which, may be executed by CPU on a computer; the operating system enables and facilitates users to access and operate computer information technology and resources. Some resources that may be employed in information technology systems include: input and output mechanisms through which data may pass into and out of a computer; memory storage into which data may be saved; and processors by which information may be processed. These information technology systems may be used to collect data for later retrieval, analysis, and manipulation, which may be facilitated through a database program. These information technology systems provide interfaces that allow users to access and operate various system components.

[00243] In one embodiment, the MMCDB controller 4901 may be connected to and/or communicate with entities such as, but not limited to: one or more users from user input devices 4911; peripheral devices 4912; an optional cryptographic processor device 4928; and/or a communications network 4913.

[00244] Networks are commonly thought to comprise the interconnection and interoperation of clients, servers, and intermediary nodes in a graph topology. It should be noted that the term "server" as used throughout this application refers generally to a computer, other device, program, or combination thereof that processes and responds to

the requests of remote users across a communications network. Servers serve their information to requesting "clients." The term "client" as used herein refers generally to a computer, program, other device, user and/or combination thereof that is capable of processing and making requests and obtaining and processing any responses from servers across a communications network. A computer, other device, program, or combination thereof that facilitates, processes information and requests, and/or furthers the passage of information from a source user to a destination user is commonly referred to as a "node." Networks are generally thought to facilitate the transfer of information from source points to destinations. A node specifically tasked with furthering the passage of information from a source to a destination is commonly called a "router." There are many forms of networks such as Local Area Networks (LANs), Pico networks, Wide Area Networks (WANs), Wireless Networks (WLANs), etc. For example, the Internet is generally accepted as being an interconnection of a multitude of networks whereby remote clients and servers may access and interoperate with one another.

[00245] The MMCDB controller 4901 may be based on computer systems that may comprise, but are not limited to, components such as: a computer systemization 4902 connected to memory 4929.

Computer Systemization

[00246] A computer systemization 4902 may comprise a clock 4930, central processing unit ("CPU(s)" and/or "processor(s)" (these terms are used interchangeable throughout the disclosure unless noted to the contrary)) 4903, a memory 4929 (e.g., a read only memory (ROM) 4906, a random access memory (RAM) 4905, etc.), and/or an interface bus 4907, and most frequently, although not necessarily, are all interconnected and/or communicating through a system bus 4904 on one or more (mother)board(s) 4902 having conductive and/or otherwise transportive circuit pathways through which instructions (e.g., binary encoded signals) may travel to effectuate communications, operations, storage, etc. The computer systemization may be connected to a power source 4986; e.g., optionally the power source may be internal. Optionally, a cryptographic processor 4926 and/or transceivers (e.g., ICs) 4974 may be connected to

the system bus. In another embodiment, the cryptographic processor and/or transceivers may be connected as either internal and/or external peripheral devices 4912 via the interface bus I/O. In turn, the transceivers may be connected to antenna(s) 4975, thereby effectuating wireless transmission and reception of various communication and/or sensor protocols; for example the antenna(s) may connect to: a Texas Instruments WiLink WL1283 transceiver chip (e.g., providing 802.11n, Bluetooth 3.0, FM, global positioning system (GPS) (thereby allowing MMCDB controller to determine its location)); Broadcom BCM4329FKUBG transceiver chip (e.g., providing 802.11n, Bluetooth 2.1 + EDR, FM, etc.); a Broadcom BCM4750IUB8 receiver chip (e.g., GPS); an Infineon Technologies X-Gold 618-PMB9800 (e.g., providing 2G/3G HSDPA/HSUPA communications); and/or the like. The system clock typically has a crystal oscillator and generates a base signal through the computer systemization's circuit pathways. The clock is typically coupled to the system bus and various clock multipliers that will increase or decrease the base operating frequency for other components interconnected in the computer systemization. The clock and various components in a computer systemization drive signals embodying information throughout the system. Such transmission and reception of instructions embodying information throughout a computer systemization may be commonly referred to as communications. These communicative instructions may further be transmitted, received, and the cause of return and/or reply communications beyond the instant computer systemization to: communications networks, input devices, other computer systemizations, peripheral devices, and/or the like. It should be understood that in alternative embodiments, any of the above components may be connected directly to one another, connected to the CPU, and/or organized in numerous variations employed as exemplified by various computer systems.

[00247] The CPU comprises at least one high-speed data processor adequate to execute program components for executing user and/or system-generated requests. Often, the processors themselves will incorporate various specialized processing units, such as, but not limited to: integrated system (bus) controllers, memory management control units, floating point units, and even specialized processing sub-units like graphics processing units, digital signal processing units, and/or the like. Additionally,

processors may include internal fast access addressable memory, and be capable of mapping and addressing memory 4929 beyond the processor itself; internal memory may include, but is not limited to: fast registers, various levels of cache memory (e.g., level 1, 2, 3, etc.), RAM, etc. The processor may access this memory through the use of a memory address space that is accessible via instruction address, which the processor can construct and decode allowing it to access a circuit path to a specific memory address space having a memory state. The CPU may be a microprocessor such as: AMD's Athlon, Duron and/or Opteron; ARM's application, embedded and secure processors; IBM and/or Motorola's DragonBall and PowerPC; IBM's and Sony's Cell processor; Intel's Celeron, Core (2) Duo, Itanium, Pentium, Xeon, and/or XScale; and/or the like processor(s). The CPU interacts with memory through instruction passing through conductive and/or transportive conduits (e.g., (printed) electronic and/or optic circuits) to execute stored instructions (i.e., program code) according to conventional data processing techniques. Such instruction passing facilitates communication within the MMCDB controller and beyond through various interfaces. Should processing requirements dictate a greater amount speed and/or capacity, distributed processors (e.g., Distributed MMCDB), mainframe, multi-core, parallel, and/or super-computer architectures may similarly be employed. Alternatively, should deployment requirements dictate greater portability, smaller Personal Digital Assistants (PDAs) may be employed.

[00248] Depending on the particular implementation, features of the MMCDB may be achieved by implementing a microcontroller such as CAST's R8051XC2 microcontroller; Intel's MCS 51 (i.e., 8051 microcontroller); and/or the like. Also, to implement certain features of the MMCDB, some feature implementations may rely on embedded components, such as: Application-Specific Integrated Circuit ("ASIC"), Digital Signal Processing ("DSP"), Field Programmable Gate Array ("FPGA"), and/or the like embedded technology. For example, any of the MMCDB component collection (distributed or otherwise) and/or features may be implemented via the microprocessor and/or via embedded components; e.g., via ASIC, coprocessor, DSP, FPGA, and/or the like. Alternately, some implementations of the MMCDB may be implemented with

embedded components that are configured and used to achieve a variety of features or signal processing.

[00249] Depending on the particular implementation, the embedded components may include software solutions, hardware solutions, and/or some combination of both hardware/software solutions. For example, MMCDB features discussed herein may be achieved through implementing FPGAs, which are a semiconductor devices containing programmable logic components called "logic blocks", and programmable interconnects, such as the high performance FPGA Virtex series and/or the low cost Spartan series manufactured by Xilinx. Logic blocks and interconnects can be programmed by the customer or designer, after the FPGA is manufactured, to implement any of the MMCDB features. A hierarchy of programmable interconnects allow logic blocks to be interconnected as needed by the MMCDB system designer/administrator, somewhat like a one-chip programmable breadboard. An FPGA's logic blocks can be programmed to perform the operation of basic logic gates such as AND, and XOR, or more complex combinational operators such as decoders or mathematical operations. In most FPGAs, the logic blocks also include memory elements, which may be circuit flip-flops or more complete blocks of memory. In some circumstances, the MMCDB may be developed on regular FPGAs and then migrated into a fixed version that more resembles ASIC implementations. Alternate or coordinating implementations may migrate MMCDB controller features to a final ASIC instead of or in addition to FPGAs. Depending on the implementation all of the aforementioned embedded components and microprocessors may be considered the "CPU" and/or "processor" for the MMCDB.

Power Source

[00250] The power source 4986 may be of any standard form for powering small electronic circuit board devices such as the following power cells: alkaline, lithium hydride, lithium ion, lithium polymer, nickel cadmium, solar cells, and/or the like. Other types of AC or DC power sources may be used as well. In the case of solar cells, in one embodiment, the case provides an aperture through which the solar cell may capture photonic energy. The power cell 4986 is connected to at least one of the

interconnected subsequent components of the MMCDB thereby providing an electric current to all subsequent components. In one example, the power source 4986 is connected to the system bus component 4904. In an alternative embodiment, an outside power source 4986 is provided through a connection across the I/O 4908 interface. For example, a USB and/or IEEE 1394 connection carries both data and power across the connection and is therefore a suitable source of power.

Interface Adapters

[00251] Interface bus(ses) 4907 may accept, connect, and/or communicate to a number of interface adapters, conventionally although not necessarily in the form of adapter cards, such as but not limited to: input output interfaces (I/O) 4908, storage interfaces 4909, network interfaces 4910, and/or the like. Optionally, cryptographic processor interfaces 4927 similarly may be connected to the interface bus. The interface bus provides for the communications of interface adapters with one another as well as with other components of the computer systemization. Interface adapters are adapted for a compatible interface bus. Interface adapters conventionally connect to the interface bus via a slot architecture. Conventional slot architectures may be employed, such as, but not limited to: Accelerated Graphics Port (AGP), Card Bus, (Extended) Industry Standard Architecture ((E)ISA), Micro Channel Architecture (MCA), NuBus, Peripheral Component Interconnect (Extended) (PCI(X)), PCI Express, Personal Computer Memory Card International Association (PCMCIA), and/or the like.

[00252] Storage interfaces 4909 may accept, communicate, and/or connect to a number of storage devices such as, but not limited to: storage devices 4914, removable disc devices, and/or the like. Storage interfaces may employ connection protocols such as, but not limited to: (Ultra) (Serial) Advanced Technology Attachment (Packet Interface) ((Ultra) (Serial) ATA(PI)), (Enhanced) Integrated Drive Electronics ((E)IDE), Institute of Electrical and Electronics Engineers (IEEE) 1394, fiber channel, Small Computer Systems Interface (SCSI), Universal Serial Bus (USB), and/or the like.

[00253] Network interfaces 4910 may accept, communicate, and/or connect to a communications network 4913. Through a communications network 4913, the MMCDB controller is accessible through remote clients 4933b (e.g., computers with web

browsers) by users 4933a. Network interfaces may employ connection protocols such as, but not limited to: direct connect, Ethernet (thick, thin, twisted pair 10/100/1000 Base T, and/or the like), Token Ring, wireless connection such as IEEE 802.11a-x, and/or the like. Should processing requirements dictate a greater amount speed and/or capacity, distributed network controllers (e.g., Distributed MMCDB), architectures may similarly be employed to pool, load balance, and/or otherwise increase the communicative bandwidth required by the MMCDB controller. A communications network may be any one and/or the combination of the following: a direct interconnection; the Internet; a Local Area Network (LAN); a Metropolitan Area Network (MAN); an Operating Missions as Nodes on the Internet (OMNI); a secured custom connection; a Wide Area Network (WAN); a wireless network (e.g., employing protocols such as, but not limited to a Wireless Application Protocol (WAP), I-mode, and/or the like); and/or the like. A network interface may be regarded as a specialized form of an input output interface. Further, multiple network interfaces 4910 may be used to engage with various communications network types 4913. For example, multiple network interfaces may be employed to allow for the communication over broadcast, multicast, and/or unicast networks.

[00254] Input Output interfaces (I/O) 4908 may accept, communicate, and/or connect to user input devices 4911, peripheral devices 4912, cryptographic processor devices 4928, and/or the like. I/O may employ connection protocols such as, but not limited to: audio: analog, digital, monaural, RCA, stereo, and/or the like; data: Apple Desktop Bus (ADB), IEEE 1394a-b, serial, universal serial bus (USB); infrared; joystick; keyboard; midi; optical; PC AT; PS/2; parallel; radio; video interface: Apple Desktop Connector (ADC), BNC, coaxial, component, composite, digital, Digital Visual Interface (DVI), high-definition multimedia interface (HDMI), RCA, RF antennae, S-Video, VGA, and/or the like; wireless transceivers: 802.11a/b/g/n/x; Bluetooth; cellular (e.g., code division multiple access (CDMA), high speed packet access (HSPA(+)), high-speed downlink packet access (HSDPA), global system for mobile communications (GSM), long term evolution (LTE), WiMax, etc.); and/or the like. One typical output device may include a video display, which typically comprises a Cathode Ray Tube (CRT) or Liquid Crystal Display (LCD) based monitor with an interface (e.g., DVI circuitry and cable)

that accepts signals from a video interface, may be used. The video interface composites information generated by a computer systemization and generates video signals based on the composited information in a video memory frame. Another output device is a television set, which accepts signals from a video interface. Typically, the video interface provides the composited video information through a video connection interface that accepts a video display interface (e.g., an RCA composite video connector accepting an RCA composite video cable; a DVI connector accepting a DVI display cable, etc.).

[00255] User input devices 4911 often are a type of peripheral device 512 (see below) and may include: card readers, dongles, finger print readers, gloves, graphics tablets, joysticks, keyboards, microphones, mouse (mice), remote controls, retina readers, touch screens (e.g., capacitive, resistive, etc.), trackballs, trackpads, sensors (e.g., accelerometers, ambient light, GPS, gyroscopes, proximity, etc.), styluses, and/or the like.

[00256] Peripheral devices 4912 may be connected and/or communicate to I/O and/or other facilities of the like such as network interfaces, storage interfaces, directly to the interface bus, system bus, the CPU, and/or the like. Peripheral devices may be external, internal and/or part of the MMCDB controller. Peripheral devices may include: antenna, audio devices (e.g., line-in, line-out, microphone input, speakers, etc.), cameras (e.g., still, video, webcam, etc.), dongles (e.g., for copy protection, ensuring secure transactions with a digital signature, and/or the like), external processors (for added capabilities; e.g., crypto devices 528), force-feedback devices (e.g., vibrating motors), network interfaces, printers, scanners, storage devices, transceivers (e.g., cellular, GPS, etc.), video devices (e.g., goggles, monitors, etc.), video sources, visors, and/or the like. Peripheral devices often include types of input devices (e.g., cameras).

[00257] It should be noted that although user input devices and peripheral devices may be employed, the MMCDB controller may be embodied as an embedded, dedicated, and/or monitor-less (i.e., headless) device, wherein access would be provided over a network interface connection.

[00258] Cryptographic units such as, but not limited to, microcontrollers, processors 4926, interfaces 4927, and/or devices 4928 may be attached, and/or

communicate with the MMCDB controller. A MC68HC16 microcontroller, manufactured by Motorola Inc., may be used for and/or within cryptographic units. The MC68HC16 microcontroller utilizes a 16-bit multiply-and-accumulate instruction in the 16 MHz configuration and requires less than one second to perform a 512-bit RSA private key operation. Cryptographic units support the authentication of communications from interacting agents, as well as allowing for anonymous transactions. Cryptographic units may also be configured as part of the CPU. Equivalent microcontrollers and/or processors may also be used. Other commercially available specialized cryptographic processors include: Broadcom's CryptoNetX and other Security Processors; nCipher's nShield; SafeNet's Luna PCI (e.g., 7100) series; Semaphore Communications' 40 MHz Roadrunner 184; Sun's Cryptographic Accelerators (e.g., Accelerator 6000 PCIe Board, Accelerator 500 Daughtercard); Via Nano Processor (e.g., L2100, L2200, U2400) line, which is capable of performing 500+ MB/s of cryptographic instructions; VLSI Technology's 33 MHz 6868; and/or the like.

Memory

[00259] Generally, any mechanization and/or embodiment allowing a processor to affect the storage and/or retrieval of information is regarded as memory 4929. However, memory is a fungible technology and resource, thus, any number of memory embodiments may be employed in lieu of or in concert with one another. It is to be understood that the MMCDB controller and/or a computer systemization may employ various forms of memory 4929. For example, a computer systemization may be configured wherein the operation of on-chip CPU memory (e.g., registers), RAM, ROM, and any other storage devices are provided by a paper punch tape or paper punch card mechanism; however, such an embodiment would result in an extremely slow rate of operation. In a typical configuration, memory 4929 will include ROM 4906, RAM 4905, and a storage device 4914. A storage device 4914 may be any conventional computer system storage. Storage devices may include a drum; a (fixed and/or removable) magnetic disk drive; a magneto-optical drive; an optical drive (i.e., Blueray, CD ROM/RAM/Recordable (R)/ReWritable (RW), DVD R/RW, HD DVD R/RW etc.); an array of devices (e.g., Redundant Array of Independent Disks (RAID)); solid state

memory devices (USB memory, solid state drives (SSD), etc.); other processor-readable storage mediums; and/or other devices of the like. Thus, a computer systemization generally requires and makes use of memory.

Component Collection

[00260] The memory 4929 may contain a collection of program and/or database components and/or data such as, but not limited to: operating system component(s) 4915 (operating system); information server component(s) 4916 (information server); user interface component(s) 4917 (user interface); Web browser component(s) 4918 (Web browser); database(s) 4919; mail server component(s) 4921; mail client component(s) 4922; cryptographic server component(s) 4920 (cryptographic server); the MMCDB component(s) 4935; and/or the like (i.e., collectively a component collection). These components may be stored and accessed from the storage devices and/or from storage devices accessible through an interface bus. Although non-conventional program components such as those in the component collection, typically, are stored in a local storage device 4914, they may also be loaded and/or stored in memory such as: peripheral devices, RAM, remote storage facilities through a communications network, ROM, various forms of memory, and/or the like.

Operating System

[00261] The operating system component 4915 is an executable program component facilitating the operation of the MMCDB controller. Typically, the operating system facilitates access of I/O, network interfaces, peripheral devices, storage devices, and/or the like. The operating system may be a highly fault tolerant, scalable, and secure system such as: Apple Macintosh OS X (Server); AT&T Plan 9; Be OS; Unix and Unix-like system distributions (such as AT&T's UNIX; Berkley Software Distribution (BSD) variations such as FreeBSD, NetBSD, OpenBSD, and/or the like; Linux distributions such as Red Hat, Ubuntu, and/or the like); and/or the like operating systems. However, more limited and/or less secure operating systems also may be employed such as Apple Macintosh OS, IBM OS/2, Microsoft DOS, Microsoft Windows 2000/2003/3.1/95/98/CE/Millenium/NT/Vista/XP/Win7 (Server), Palm OS, and/or

the like. An operating system may communicate to and/or with other components in a component collection, including itself, and/or the like. Most frequently, the operating system communicates with other program components, user interfaces, and/or the like. For example, the operating system may contain, communicate, generate, obtain, and/or provide program component, system, user, and/or data communications, requests, and/or responses. The operating system, once executed by the CPU, may enable the interaction with communications networks, data, I/O, peripheral devices, program components, memory, user input devices, and/or the like. The operating system may provide communications protocols that allow the MMCDB controller to communicate with other entities through a communications network 4913. Various communication protocols may be used by the MMCDB controller as a subcarrier transport mechanism for interaction, such as, but not limited to: multicast, TCP/IP, UDP, unicast, and/or the like.

Information Server

[00262] An information server component 4916 is a stored program component that is executed by a CPU. The information server may be a conventional Internet information server such as, but not limited to Apache Software Foundation's Apache, Microsoft's Internet Information Server, and/or the like. The information server may allow for the execution of program components through facilities such as Active Server Page (ASP), ActiveX, (ANSI) (Objective-) C (++), C# and/or .NET, Common Gateway Interface (CGI) scripts, dynamic (D) hypertext markup language (HTML), FLASH, Java, JavaScript, Practical Extraction Report Language (PERL), Hypertext Pre-Processor (PHP), pipes, Python, wireless application protocol (WAP), WebObjects, and/or the like. The information server may support secure communications protocols such as, but not limited to, File Transfer Protocol (FTP); HyperText Transfer Protocol (HTTP); Secure Hypertext Transfer Protocol (HTTPS), Secure Socket Layer (SSL), messaging protocols (e.g., America Online (AOL) Instant Messenger (AIM), Application Exchange (APEX), ICQ, Internet Relay Chat (IRC), Microsoft Network (MSN) Messenger Service, Presence and Instant Messaging Protocol (PRIM), Internet Engineering Task Force's (IETF's) Session Initiation Protocol (SIP), SIP for Instant Messaging and Presence Leveraging

Extensions (SIMPLE), open XML-based Extensible Messaging and Presence Protocol (XMPP) (i.e., Jabber or Open Mobile Alliance's (OMA's) Instant Messaging and Presence Service (IMPS)), Yahoo! Instant Messenger Service, and/or the like. The information server provides results in the form of Web pages to Web browsers, and allows for the manipulated generation of the Web pages through interaction with other program components. After a Domain Name System (DNS) resolution portion of an HTTP request is resolved to a particular information server, the information server resolves requests for information at specified locations on the MMCDB controller based on the remainder of the HTTP request. For example, a request such as http://123.124.125.126/myInformation.html might have the IP portion of the request "123.124.125.126" resolved by a DNS server to an information server at that IP address; that information server might in turn further parse the http request for the "/myInformation.html" portion of the request and resolve it to a location in memory containing the information "myInformation.html." Additionally, other information serving protocols may be employed across various ports, e.g., FTP communications across port 21, and/or the like. An information server may communicate to and/or with other components in a component collection, including itself, and/or facilities of the like. Most frequently, the information server communicates with the MMCDB database 4919, operating systems, other program components, user interfaces, Web browsers, and/or the like.

[00263] Access to the MMCDB database may be achieved through a number of database bridge mechanisms such as through scripting languages as enumerated below (e.g., CGI) and through inter-application communication channels as enumerated below (e.g., CORBA, WebObjects, etc.). Any data requests through a Web browser are parsed through the bridge mechanism into appropriate grammars as required by the MMCDB. In one embodiment, the information server would provide a Web form accessible by a Web browser. Entries made into supplied fields in the Web form are tagged as having been entered into the particular fields, and parsed as such. The entered terms are then passed along with the field tags, which act to instruct the parser to generate queries directed to appropriate tables and/or fields. In one embodiment, the parser may generate queries in standard SQL by instantiating a search string with the proper

join/select commands based on the tagged text entries, wherein the resulting command is provided over the bridge mechanism to the MMCDB as a query. Upon generating query results from the query, the results are passed over the bridge mechanism, and may be parsed for formatting and generation of a new results Web page by the bridge mechanism. Such a new results Web page is then provided to the information server, which may supply it to the requesting Web browser.

[00264] Also, an information server may contain, communicate, generate, obtain, and/or provide program component, system, user, and/or data communications, requests, and/or responses.

User Interface

[00265] Computer interfaces in some respects are similar to automobile operation interfaces. Automobile operation interface elements such as steering wheels, gearshifts, and speedometers facilitate the access, operation, and display of automobile resources, and status. Computer interaction interface elements such as check boxes, cursors, menus, scrollers, and windows (collectively and commonly referred to as widgets) similarly facilitate the access, capabilities, operation, and display of data and computer hardware and operating system resources, and status. Operation interfaces are commonly called user interfaces. Graphical user interfaces (GUIs) such as the Apple Macintosh Operating System's Aqua, IBM's OS/2, Microsoft's Windows 2000/2003/3.1/95/98/CE/Millenium/NT/XP/Vista/7 (i.e., Aero), Unix's X-Windows (e.g., which may include additional Unix graphic interface libraries and layers such as K Desktop Environment (KDE), mythTV and GNU Network Object Model Environment (GNOME)), web interface libraries (e.g., ActiveX, AJAX, (D)HTML, FLASH, Java, JavaScript, etc. interface libraries such as, but not limited to, Dojo, jQuery UI, MooTools, Prototype, script.aculo.us, SWFObject, Yahoo! User Interface, any of which may be used and provide a baseline and means of accessing and displaying information graphically to users.

[00266] A user interface component 4917 is a stored program component that is executed by a CPU. The user interface may be a conventional graphic user interface as provided by, with, and/or atop operating systems and/or operating environments such

as already discussed. The user interface may allow for the display, execution, interaction, manipulation, and/or operation of program components and/or system facilities through textual and/or graphical facilities. The user interface provides a facility through which users may affect, interact, and/or operate a computer system. A user interface may communicate to and/or with other components in a component collection, including itself, and/or facilities of the like. Most frequently, the user interface communicates with operating systems, other program components, and/or the like. The user interface may contain, communicate, generate, obtain, and/or provide program component, system, user, and/or data communications, requests, and/or responses.

Web Browser

[00267] A Web browser component 4918 is a stored program component that is executed by a CPU. The Web browser may be a conventional hypertext viewing application such as Microsoft Internet Explorer or Netscape Navigator. Secure Web browsing may be supplied with 128bit (or greater) encryption by way of HTTPS, SSL, and/or the like. Web browsers allowing for the execution of program components through facilities such as ActiveX, AJAX, (D)HTML, FLASH, Java, JavaScript, web browser plug-in APIs (e.g., Firefox, Safari Plug-in, and/or the like APIs), and/or the like. Web browsers and like information access tools may be integrated into PDAs, cellular telephones, and/or other mobile devices. A Web browser may communicate to and/or with other components in a component collection, including itself, and/or facilities of the like. Most frequently, the Web browser communicates with information servers, operating systems, integrated program components (e.g., plug-ins), and/or the like; e.g., it may contain, communicate, generate, obtain, and/or provide program component, system, user, and/or data communications, requests, and/or responses. Also, in place of a Web browser and information server, a combined application may be developed to perform similar operations of both. The combined application would similarly affect the obtaining and the provision of information to users, user agents, and/or the like from the MMCDB enabled nodes. The combined application may be nugatory on systems employing standard Web browsers.

Mail Server

[00268] A mail server component 4921 is a stored program component that is executed by a CPU 4903. The mail server may be a conventional Internet mail server such as, but not limited to sendmail, Microsoft Exchange, and/or the like. The mail server may allow for the execution of program components through facilities such as ASP, ActiveX, (ANSI) (Objective-) C (++), C# and/or .NET, CGI scripts, Java, JavaScript, PERL, PHP, pipes, Python, WebObjects, and/or the like. The mail server may support communications protocols such as, but not limited to: Internet message access protocol (IMAP), Messaging Application Programming Interface (MAPI)/Microsoft Exchange, post office protocol (POP3), simple mail transfer protocol (SMTP), and/or the like. The mail server can route, forward, and process incoming and outgoing mail messages that have been sent, relayed and/or otherwise traversing through and/or to the MMCDB.

[00269] Access to the MMCDB mail may be achieved through a number of APIs offered by the individual Web server components and/or the operating system.

[00270] Also, a mail server may contain, communicate, generate, obtain, and/or provide program component, system, user, and/or data communications, requests, information, and/or responses.

Mail Client

[00271] A mail client component 4922 is a stored program component that is executed by a CPU 4903. The mail client may be a conventional mail viewing application such as Apple Mail, Microsoft Entourage, Microsoft Outlook, Microsoft Outlook Express, Mozilla, Thunderbird, and/or the like. Mail clients may support a number of transfer protocols, such as: IMAP, Microsoft Exchange, POP3, SMTP, and/or the like. A mail client may communicate to and/or with other components in a component collection, including itself, and/or facilities of the like. Most frequently, the mail client communicates with mail servers, operating systems, other mail clients, and/or the like; e.g., it may contain, communicate, generate, obtain, and/or provide program component, system, user, and/or data communications, requests, information, and/or

responses. Generally, the mail client provides a facility to compose and transmit electronic mail messages.

Cryptographic Server

[00272] A cryptographic server component 4920 is a stored program component that is executed by a CPU 4903, cryptographic processor 4926, cryptographic processor interface 4927, cryptographic processor device 4928, and/or the like. Cryptographic processor interfaces will allow for expedition of encryption and/or decryption requests by the cryptographic component; however, the cryptographic component, alternatively, may run on a conventional CPU. The cryptographic component allows for the encryption and/or decryption of provided data. The cryptographic component allows for both symmetric and asymmetric (e.g., Pretty Good Protection (PGP)) encryption and/or decryption. The cryptographic component may employ cryptographic techniques such as, but not limited to: digital certificates (e.g., X.509 authentication framework), digital signatures, dual signatures, enveloping, password access protection, public key management, and/or the like. The cryptographic component will facilitate numerous (encryption and/or decryption) security protocols such as, but not limited to: checksum, Data Encryption Standard (DES), Elliptical Curve Encryption (ECC), International Data Encryption Algorithm (IDEA), Message Digest 5 (MD5, which is a one way hash operation), passwords, Rivest Cipher (RC5), Rijndael, RSA (which is an Internet encryption and authentication system that uses an algorithm developed in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman), Secure Hash Algorithm (SHA), Secure Socket Layer (SSL), Secure Hypertext Transfer Protocol (HTTPS), and/or the like. Employing such encryption security protocols, the MMCDB may encrypt all incoming and/or outgoing communications and may serve as node within a virtual private network (VPN) with a wider communications network. The cryptographic component facilitates the process of "security authorization" whereby access to a resource is inhibited by a security protocol wherein the cryptographic component effects authorized access to the secured resource. In addition, the cryptographic component may provide unique identifiers of content, e.g., employing and MD5 hash to obtain a unique signature for an digital audio file. A cryptographic component may communicate to

and/or with other components in a component collection, including itself, and/or facilities of the like. The cryptographic component supports encryption schemes allowing for the secure transmission of information across a communications network to enable the MMCDB component to engage in secure transactions if so desired. The cryptographic component facilitates the secure accessing of resources on the MMCDB and facilitates the access of secured resources on remote systems; i.e., it may act as a client and/or server of secured resources. Most frequently, the cryptographic component communicates with information servers, operating systems, other program components, and/or the like. The cryptographic component may contain, communicate, generate, obtain, and/or provide program component, system, user, and/or data communications, requests, and/or responses.

The MMCDB Database

[00273] The MMCDB database component 4919 may be embodied in a database and its stored data. The database is a stored program component, which is executed by the CPU; the stored program component portion configuring the CPU to process the stored data. The database may be a conventional, fault tolerant, relational, scalable, secure database such as Oracle or Sybase. Relational databases are an extension of a flat file. Relational databases consist of a series of related tables. The tables are interconnected via a key field. Use of the key field allows the combination of the tables by indexing against the key field; i.e., the key fields act as dimensional pivot points for combining information from various tables. Relationships generally identify links maintained between tables by matching primary keys. Primary keys represent fields that uniquely identify the rows of a table in a relational database. More precisely, they uniquely identify rows of a table on the "one" side of a one-to-many relationship.

[00274] Alternatively, the MMCDB database may be implemented using various standard data-structures, such as an array, hash, (linked) list, struct, structured text file (e.g., XML), table, and/or the like. Such data-structures may be stored in memory and/or in (structured) files. In another alternative, an object-oriented database may be used, such as Frontier, ObjectStore, Poet, Zope, and/or the like. Object databases can include a number of object collections that are grouped and/or linked together by

common attributes; they may be related to other object collections by some common attributes. Object-oriented databases perform similarly to relational databases with the exception that objects are not just pieces of data but may have other types of capabilities encapsulated within a given object. If the MMCDB database is implemented as a data-structure, the use of the MMCDB database 4919 may be integrated into another component such as the MMCDB component 4935. Also, the database may be implemented as a mix of data structures, objects, and relational structures. Databases may be consolidated and/or distributed in countless variations through standard data processing techniques. Portions of databases, e.g., tables, may be exported and/or imported and thus decentralized and/or integrated.

[00275] In one embodiment, the database component 4919 includes several tables 4919a-w. A Users table 4919a may include fields such as, but not limited to: user_id, ssn, dob, first_name, last_name, age, state, address_firstline, address_secondline, zipcode, devices_list, contact_info, contact_type, alt_contact_info, alt_contact_type, and/or the like. The Users table may support and/or track multiple entity accounts on a MMCDB. A Devices table 4919b may include fields such as, but not limited to: device_id, user_id, client_ip, client_type, client_model, operating_system, os_version, app_installed_flag, and/or the like. An Apps table 4919c may include fields such as, but not limited to: app_id, app_name, app_type, OS_compatibilities_list, version, timestamp, developer_id, and/or the like. An Accounts table 4919d may include fields such as, but not limited to: account_id, account_firstname, account_lastname, account_type, account_num, account_balance_list, billingaddress_line1, billingaddress_line2, billing_zipcode, billing_state, shipping_preferences, shippingaddress_line1, shippingaddress_line2, shipping_ zipcode, shipping_state, and/or the like. A Merchants table 4919e may include fields such as, but not limited to: merchant_id, merchant_name, provi merchant_address, ip_address, mac_address, auth_key, port_num, security_settings_list, and/or the like. An Issuers table 4919f may include fields such as, but not limited to: issuer_id, issuer_name, issuer_address, ip_address, mac_address, auth_key, port_num, security_settings_list, and/or the like. An Acquirers table 4919g may include fields such as, but not limited to: acquirer_id, account_firstname, account_lastname, account_type, account_num,

account_balance_list, billingaddress_line1, billingaddress_line2, billing_zipcode, billing_state, shipping_preferences, shippingaddress_line1, shippingaddress_line2, shipping_ zipcode, shipping_state, and/or the like. A Gateways table 4919h may include fields such as, but not limited to: gateway_id, gateway_name, merchant_id, issuer_id, acquirer_id, user_id, and/or the like. A Transactions table 4919i may include fields such as, but not limited to: transaction_id, order_id, user_id, timestamp, transaction_cost, purchase_details_list, num_products, products_ list, product_type, product_params _list, product_title, product_summary, quantity, user_id, client_id, client_ip, client_type, client_model, operating_system, os_version, app_installed_flag, user_id, account_firstname, account_lastname, account_type, account_num, billingaddress_ line1, billingaddress_line2, billing_ zipcode, billing_state, shipping_preferences, shippingaddress_line1, shippingaddress_ line2, shipping_zipcode, shipping_state, merchant_id, merchant_name, merchant_auth_ key, and/or the like. A Batches table 4919j may include fields such as, but not limited to: batch_id, parent_batch_id, transaction_id, account_id, user_id, app_id, batch_rules, and/or the like. A Ledgers table 4919k may include fields such as, but not limited to: ledger_id, transaction_id, user_id, merchant_id, issuer_id, acquirer_id, aggregation_id, ledger_name, ledger_value, and/or the like. A Products table 4919l may include fields such as, but not limited to: product_id, product_name, sku, price, inventory_level, stores_carrying, unit_of_measure, and/or the like. A Offers table 4919m may include fields such as, but not limited to: offer_id, merchant_id, offered_to_user_id, offer_type, offer_description, start_date, end_date, num_times_redeemed, and/or the like. A Behavior table 4919n may include fields such as, but not limited to: behavior_id, user_id, behavior_description, behavior_type, behavior_value, date_time_behavior, and/or the like. An Analytics table 49190 may include fields such as, but not limited to: analytics_id, batch_id, user_id, transaction_id, generated_graph, generated_results_set, generated_results_set_json, input_data_set, date_time_generated, and/or the like. A Market Data table 4919p may include fields such as, but not limited to: market_data_id, index_name, index_value, last_updated_index_datetime, and/or the like. An Input Languages table 4919q may include fields such as, but not limited to: input_language_id, function_name, function_definition, parent_input_language_id, mesh_language_id, user_id,

tumbler_id, aggregation_id, and/or the like. A Mesh Language table 4919r may include fields such as, but not limited to: mesh_language_id, operation_name, operation_min_test_case, operation_max_test_case, operation_custom_test_case, mesh_language_version, mesh_language_updated_date, and/or the like. A Tumblars table 4919s may include fields such as, but not limited to: tumbler_id, user_visible_model_commands, non_user_visible_model_commands, input_key, output_key, and/or the like. An Aggregation table 4919t may include fields such as, but not limited to: aggregation_id, aggregation_data_source, key, value, parent_aggregation_id, and/or the like. A Category table 4919u may include fields such as, but not limited to: category_id, mesh_id, user_id, category_name, category_type, entity_name, is_present_in_mesh, and/or the like. A Mesh table 4919v may include fields such as, but not limited to: mesh_id, mesh_node, mesh_node_value, mesh_edge, mesh_edge_value, mesh_link, mesh_link_value, attributes, tags, keywords, and/or the like. A Price Trends table 4919w may include fields such as, but not limited to: price_trends_id, merchant_id, date_price_observed, number_observations, observed_price, next_check_date, inventory_quantity, and/or the like.

[00276] In one embodiment, the MMCDB database may interact with other database systems. For example, employing a distributed database system, queries and data access by search MMCDB component may treat the combination of the MMCDB database, an integrated data security layer database as a single database entity.

[00277] In one embodiment, user programs may contain various user interface primitives, which may serve to update the MMCDB. Also, various accounts may require custom database tables depending upon the environments and the types of clients the MMCDB may need to serve. It should be noted that any unique fields may be designated as a key field throughout. In an alternative embodiment, these tables have been decentralized into their own databases and their respective database controllers (i.e., individual database controllers for each of the above tables). Employing standard data processing techniques, one may further distribute the databases over several computer systemizations and/or storage devices. Similarly, configurations of the decentralized database controllers may be varied by consolidating and/or distributing the various

database components 4919a-w. The MMCDB may be configured to keep track of various settings, inputs, and parameters via database controllers.

[00278] The MMCDB database may communicate to and/or with other components in a component collection, including itself, and/or facilities of the like. Most frequently, the MMCDB database communicates with the MMCDB component, other program components, and/or the like. The database may contain, retain, and provide information regarding other nodes and data.

The MMCDBs

[00279] The MMCDB component 4935 is a stored program component that is executed by a CPU. In one embodiment, the MMCDB component incorporates any and/or all combinations of the aspects of the MMCDB that was discussed in the previous figures. As such, the MMCDB affects accessing, obtaining and the provision of information, services, transactions, and/or the like across various communications networks. The features and embodiments of the MMCDB discussed herein increase network efficiency by reducing data transfer requirements the use of more efficient data structures and mechanisms for their transfer and storage. As a consequence, more data may be transferred in less time, and latencies with regard to transactions, are also reduced. In many cases, such reduction in storage, transfer time, bandwidth requirements, latencies, etc., will reduce the capacity and structural infrastructure requirements to support the MMCDB's features and facilities, and in many cases reduce the costs, energy consumption/requirements, and extend the life of MMCDB's underlying infrastructure; this has the added benefit of making the MMCDB more reliable. The generation of the mesh graph and dictionary entries by the MMCDB has the technical effect of allowing more transaction, search, enrollment and email data to be analyzed and queried by the MMCDB user without a corresponding increase in data storage server / processing infrastructure. For example, by utilizing the aggregated data record normalization 306, data field recognition 307, entity type classification 308, cross-entity correlation 309, and entity attribute 310 components of the MMCDB, raw aggregated data may be stored in a more efficient manner and indexed and searched in a manner requiring less physical infrastructure and supporting faster querying with

reduced latency (e.g., through the use of a distributed linking node mesh search component). Aspects of the MMCDB facilitate faster transaction processing by reducing consumer decision latency (e.g., through the provision of customized offers requiring decreased user input and therefore reduced data transfer requirements). Similarly, many of the features and mechanisms are designed to be easier for users to use and access, thereby broadening the audience that may enjoy/employ and exploit the feature sets of the MMCDB; such ease of use also helps to increase the reliability of the MMCDB. In addition, the feature sets include heightened security as noted via the Cryptographic components 4920, 4926, 4928 and throughout, making access to the features and data more reliable and secure.

[00280] The MMCDB component may transform data aggregated from various computer resources via MMCDB components into updated entity profiles and/or social graphs, and/or the like and use of the MMCDB. In one embodiment, the MMCDB component 4935 takes inputs such as aggregated data from various computer resources, and transforms the inputs via various components (e.g., SRA 4941, CTE 4942, TDA 4943 SDA 4944, VASE 4945, DFR 4946, ETC 4947, CEC 4948, EAA 4949, EPGU 4950, STG 4951, MA 4952, UBPA 4953, UPI 4954, TDN 4955, CTC 4956, TDF 4957, CDA 4958, ESA 4959, BAR 4960, AMS 4961, ADRN 4962, EXC 4963, CRA 4964, and/or the like), into outputs such as updated entity profiles and social graphs within the MMCDB.

[00281] The MMCDB component enabling access of information between nodes may be developed by employing standard development tools and languages such as, but not limited to: Apache components, Assembly, ActiveX, binary executables, (ANSI) (Objective-) C (++), C# and/or .NET, database adapters, CGI scripts, Java, JavaScript, mapping tools, procedural and object oriented development tools, PERL, PHP, Python, shell scripts, SQL commands, web application server extensions, web development environments and libraries (e.g., Microsoft's ActiveX; Adobe AIR, FLEX & FLASH; AJAX; (D)HTML; Dojo, Java; JavaScript; jQuery(UI); MooTools; Prototype; script.aculo.us; Simple Object Access Protocol (SOAP); SWFObject; Yahoo! User Interface; and/or the like), WebObjects, and/or the like. In one embodiment, the MMCDB server employs a cryptographic server to encrypt and decrypt communications. The MMCDB component may communicate to and/or with other components in a

component collection, including itself, and/or facilities of the like. Most frequently, the MMCDB component communicates with the MMCDB database, operating systems, other program components, and/or the like. The MMCDB may contain, communicate, generate, obtain, and/or provide program component, system, user, and/or data communications, requests, and/or responses.

Distributed MMCDBs

[00282] The structure and/or operation of any of the MMCDB node controller components may be combined, consolidated, and/or distributed in any number of ways to facilitate development and/or deployment. Similarly, the component collection may be combined in any number of ways to facilitate deployment and/or development. To accomplish this, one may integrate the components into a common code base or in a facility that can dynamically load the components on demand in an integrated fashion.

[00283] The component collection may be consolidated and/or distributed in countless variations through standard data processing and/or development techniques. Multiple instances of any one of the program components in the program component collection may be instantiated on a single node, and/or across numerous nodes to improve performance through load-balancing and/or data-processing techniques. Furthermore, single instances may also be distributed across multiple controllers and/or storage devices; e.g., databases. All program component instances and controllers working in concert may do so through standard data processing communication techniques.

[00284] The configuration of the MMCDB controller will depend on the context of system deployment. Factors such as, but not limited to, the budget, capacity, location, and/or use of the underlying hardware resources may affect deployment requirements and configuration. Regardless of if the configuration results in more consolidated and/or integrated program components, results in a more distributed series of program components, and/or results in some combination between a consolidated and distributed configuration, data may be communicated, obtained, and/or provided. Instances of components consolidated into a common code base from the program component collection may communicate, obtain, and/or provide data. This may be

accomplished through intra-application data processing communication techniques such as, but not limited to: data referencing (e.g., pointers), internal messaging, object instance variable communication, shared memory space, variable passing, and/or the like.

[00285] If component collection components are discrete, separate, and/or external to one another, then communicating, obtaining, and/or providing data with and/or to other component components may be accomplished through inter-application data processing communication techniques such as, but not limited to: Application Program Interfaces (API) information passage; (distributed) Component Object Model ((D)COM), (Distributed) Object Linking and Embedding ((D)OLE), and/or the like), Common Object Request Broker Architecture (CORBA), Jini local and remote application program interfaces, JavaScript Object Notation (JSON), Remote Method Invocation (RMI), SOAP, process pipes, shared files, and/or the like. Messages sent between discrete component components for inter-application communication or within memory spaces of a singular component for intra-application communication may be facilitated through the creation and parsing of a grammar. A grammar may be developed by using development tools such as lex, yacc, XML, and/or the like, which allow for grammar generation and parsing capabilities, which in turn may form the basis of communication messages within and between components.

[00286] For example, a grammar may be arranged to recognize the tokens of an HTTP post command, e.g.:

```
w3c -post http://... Value1
```

[00287] where Value1 is discerned as being a parameter because "http://" is part of the grammar syntax, and what follows is considered part of the post value. Similarly, with such a grammar, a variable "Value1" may be inserted into an "http://" post command and then sent. The grammar syntax itself may be presented as structured data that is interpreted and/or otherwise used to generate the parsing mechanism (e.g., a syntax description text file as processed by lex, yacc, etc.). Also, once the parsing mechanism is generated and/or instantiated, it itself may process and/or parse structured data such as, but not limited to: character (e.g., tab) delineated text, HTML,

structured text streams, XML, and/or the like structured data. In another embodiment, inter-application data processing protocols themselves may have integrated and/or readily available parsers (e.g., JSON, SOAP, and/or like parsers) that may be employed to parse (e.g., communications) data. Further, the parsing grammar may be used beyond message parsing, but may also be used to parse: databases, data collections, data stores, structured data, and/or the like. Again, the desired configuration will depend upon the context, environment, and requirements of system deployment.

[00288] For example, in some implementations, the MMCDB controller may be executing a PHP script implementing a Secure Sockets Layer ("SSL") socket server via the information sherver, which listens to incoming communications on a server port to which a client may send data, e.g., data encoded in JSON format. Upon identifying an incoming communication, the PHP script may read the incoming message from the client device, parse the received JSON-encoded text data to extract information from the JSON-encoded text data into PHP script variables, and store the data (e.g., client identifying information, etc.) and/or extracted information in a relational database accessible using the Structured Query Language ("SQL"). An exemplary listing, written substantially in the form of PHP/SQL commands, to accept JSON-encoded input data from a client device via a SSL connection, parse the data to extract variables, and store the data to a database, is provided below:

```
<?PHP
header('Content-Type: text/plain');

//set ip address and port to listen to for incoming data
$address = '192.168.0.100';
$port = 255;

//create a server-side SSL socket, listen
//for/accept incoming communication
$sock = socket_create(AF_INET, SOCK_STREAM, 0);
socket_bind($sock, $address, $port)
     or die('Could not bind to address');
socket_listen($sock);
$client = socket_accept($sock);
```

```
//read input data from client device in 1024 byte
//blocks until end of message
do {
        $input = "";
        $input = socket_read($client, 1024);
        $data .= $input;
} while($input != "");

// parse data to extract variables
$obj = json_decode($data, true);

// store input data in a database
mysql_connect("10.1.1.1",$srvr,$pass); // access database server
mysql_select("CLIENT_DB.SQL"); // select database to append
mysql_query("INSERT INTO UserTable (transmission) VALUES
($data)"); // add data to UserTable table in  a CLIENT database
mysql_close("CLIENT_DB.SQL"); // close connection to database
?>
```

[00289] Also, the following resources may be used to provide example embodiments regarding SOAP parser implementation:

```
http://www.xav.com/perl/site/lib/SOAP/Parser.html
http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?
topic=/com.ibm.IBMDI.doc/referenceguide295.htm
```

[00290] and other parser implementations:

```
http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?
topic=/com.ibm.IBMDI.doc/referenceguide259.htm
```

[00291] all of which are hereby expressly incorporated by reference.

[00292] In order to address various issues and advance the art, the entirety of this application for MMCDB (including the Cover Page, Title, Headings, Field, Background, Summary, Brief Description of the Drawings, Detailed Description, Claims, Abstract,

Figures, Appendices, and otherwise) shows, by way of illustration, various embodiments in which the claimed innovations may be practiced. The advantages and features of the application are of a representative sample of embodiments only, and are not exhaustive and/or exclusive. They are presented only to assist in understanding and teach the claimed principles. It should be understood that they are not representative of all claimed innovations. As such, certain aspects of the disclosure have not been discussed herein. That alternate embodiments may not have been presented for a specific portion of the innovations or that further undescribed alternate embodiments may be available for a portion is not to be considered a disclaimer of those alternate embodiments. It will be appreciated that many of those undescribed embodiments incorporate the same principles of the innovations and others are equivalent. Thus, it is to be understood that other embodiments may be utilized and functional, logical, operational, organizational, structural and/or topological modifications may be made without departing from the scope and/or spirit of the disclosure. As such, all examples and/or embodiments are deemed to be non-limiting throughout this disclosure. Also, no inference should be drawn regarding those embodiments discussed herein relative to those not discussed herein other than it is as such for purposes of reducing space and repetition. For instance, it is to be understood that the logical and/or topological structure of any combination of any program components (a component collection), other components and/or any present feature sets as described in the figures and/or throughout are not limited to a fixed operating order and/or arrangement, but rather, any disclosed order is exemplary and all equivalents, regardless of order, are contemplated by the disclosure. Furthermore, it is to be understood that such features are not limited to serial execution, but rather, any number of threads, processes, services, servers, and/or the like that may execute asynchronously, concurrently, in parallel, simultaneously, synchronously, and/or the like are contemplated by the disclosure. As such, some of these features may be mutually contradictory, in that they cannot be simultaneously present in a single embodiment. Similarly, some features are applicable to one aspect of the innovations, and inapplicable to others. In addition, the disclosure includes other innovations not presently claimed. Applicant reserves all rights in those presently unclaimed innovations including the right to claim such innovations, file additional applications, continuations, continuations in part, divisions, and/or the like thereof. As such, it

should be understood that advantages, embodiments, examples, functional, features, logical, operational, organizational, structural, topological, and/or other aspects of the disclosure are not to be considered limitations on the disclosure as defined by the claims or limitations on equivalents to the claims. It is to be understood that, depending on the particular needs and/or characteristics of a MMCDB individual and/or enterprise user, database configuration and/or relational model, data type, data transmission and/or network framework, syntax structure, and/or the like, various embodiments of the MMCDB, may be implemented that enable a great deal of flexibility and customization. For example, aspects of the MMCDB may be adapted for restaurant dining, online shopping ,brick-and-mortar shopping, secured information processing, and/or the like. While various embodiments and discussions of the MMCDB have been directed to electronic purchase transactions, however, it is to be understood that the embodiments described herein may be readily configured and/or customized for a wide variety of other applications and/or implementations.

CLAIMS

What is claimed is:

1. A centralized personal information platform processor-implemented method for enhancing transaction speed through the reduction of user input data transfer requirements, comprising:

aggregating data records including search results, purchase transaction data, service usage data, service enrollment data, email data and social data;

identifying data field types within the data records and their associated data values;

identifying an entity from the data field types and their associated data values;

generating, via a processor, correlations of the entity to other entities identifiable from the data field types and their associated data values by comparing fields from a non-normalized transaction data record with fields extracted from a normalized transaction data record template and determining whether the field identifiers of fields in the non-normalized transaction data record match any of the normalized transaction data record template, based on the comparison, wherein a map is generated between fields of the non-normalized transaction data record and normalized transaction data record template;

associating, via the processor, attributes to the entity by drawing inferences related to the entity from the data field types and their associated data values;

generating an updated profile and social graph of the entity using the generated correlations and associated attributes; and

providing the updated profile and social graph for an automated web form filling request.

2. The method of claim 1, further comprising:

generating a search query using the updated profile; and

providing the generated search query for further data aggregation.

3. The method of claim 2, wherein the search query is a web search query.

4. The method of claim 2, wherein the search query is a social search query.

5. The method of claim 2, wherein the search query is an email data aggregation query.

6. The method of claim 4, wherein the updated profile includes a social login credential; and wherein the social search query utilizes the social login credential.

7. The method of claim 1, further comprising:

generating a search query using the updated social graph; and

providing the generated search query for further data aggregation.

8. The method of claim 6, wherein the search query is a web search query.

9. The method of claim 6, wherein the search query is a social search query.

10. The method of claim 8, wherein the updated profile includes a social login credential; and wherein the social search query utilizes the social login credential.

11. The method of claim 1, wherein the entity is one of: an Internet Protocol address; an individual; a pair of associated individuals; and a household; an office space; and an organization.

12. A centralized personal information platform processor-implemented system for enhancing transaction speed through the reduction of user input data transfer requirements, comprising:

means to aggregate data records including search results, purchase transaction data, service usage data, service enrollment data, email data and social data;

means to identify data field types within the data records and their associated data values;

means to identify an entity from the data field types and their associated data values;

means to generate, via a processor, correlations of the entity to other entities identifiable from the data field types and their associated data values by comparing fields from a non-normalized transaction data record with fields extracted from a normalized transaction data record template and determining whether the field identifiers of fields in the non-normalized transaction data record match any of the normalized transaction data record template, based on the comparison, wherein a map is generated between fields of the non-normalized transaction data record and normalized transaction data record template;

means to associate, via the processor, attributes to the entity by drawing inferences related to the entity from the data field types and their associated data values;

means to generate an updated profile and social graph of the entity using the generated correlations and associated attributes; and

means to provide the updated profile and social graph for an automated web form filling request.

13. The system of claim 12, further comprising:

means to generate a search query using the updated profile; and

means to provide the generated search query for further data aggregation.

14. The system of claim 13, wherein the search query is a web search query.

15. The system of claim 13, wherein the search query is a social search query.

16. The system of claim 13, wherein the search query is an email data aggregation query.

17. The system of claim 15, wherein the updated profile includes a social login credential; and wherein the social search query utilizes the social login credential.

18. The system of claim 12, further comprising:

means to generate a search query using the updated social graph; and

means to provide the generated search query for further data aggregation.

19. The system of claim 17, wherein the search query is a web search query.

20. The system of claim 17, wherein the search query is a social search query.

21. The system of claim 19, wherein the updated profile includes a social login credential; and wherein the social search query utilizes the social login credential.

22. The system of claim 12, wherein the entity is one of: an Internet Protocol address; an individual; a pair of associated individuals; and a household; an office space; and an organization.

23. A centralized personal information platform processor-implemented apparatus for enhancing transaction speed through the reduction of user input data transfer requirements, comprising:

a memory;

a processor disposed in communication with said memory, and configured to issue a plurality of processing instructions stored in the memory, wherein the processor issues instructions to:

aggregate data records including search results, purchase transaction data, service usage data, service enrollment data, email data and social data;

identify data field types within the data records and their associated data values;

identify an entity from the data field types and their associated data values;

generate, via a processor, correlations of the entity to other entities identifiable from the data field types and their associated data values by comparing fields from a non-normalized transaction data record with fields extracted from a normalized transaction data record template and determining whether the field identifiers of fields in the non-normalized transaction data record match any of the normalized transaction data record template, based on the comparison, wherein a map is generated between fields of the non-normalized transaction data record and normalized transaction data record template;

associate, via the processor, attributes to the entity by drawing inferences related to the entity from the data field types and their associated data values;

generate an updated profile and social graph of the entity using the generated correlations and associated attributes; and

provide the updated profile and social graph for an automated web form filling request.

24. The apparatus of claim 23, further comprising instructions to:

generate a search query using the updated profile; and

provide the generated search query for further data aggregation.

25. The apparatus of claim 24, wherein the search query is a web search query.

26. The apparatus of claim 24, wherein the search query is a social search query.

27. The apparatus of claim 24, wherein the search query is an email data aggregation query.

28. The apparatus of claim 26, wherein the updated profile includes a social login credential; and wherein the social search query utilizes the social login credential.

29. The apparatus of claim 23, further comprising instructions to:

generate a search query using the updated social graph; and

provide the generated search query for further data aggregation.

30. The apparatus of claim 28, wherein the search query is a web search query.

31. The apparatus of claim 28, wherein the search query is a social search query.

32. The apparatus of claim 30, wherein the updated profile includes a social login credential; and wherein the social search query utilizes the social login credential.

33. The apparatus of claim 23, wherein the entity is one of: an Internet Protocol address; an individual; a pair of associated individuals; and a household; an office space; and an organization.

34. A non-transitory medium storing processor-issuable instructions for a centralized personal information platform processor-implemented to:

aggregate data records including search results, purchase transaction data, service usage data, service enrollment data, email data and social data;

identify data field types within the data records and their associated data values;

identify an entity from the data field types and their associated data values;

generate, via a processor, correlations of the entity to other entities identifiable from the data field types and their associated data values by comparing fields from a non-normalized transaction data record with fields extracted from a normalized transaction data record template and determining whether the field identifiers of fields in the non-normalized transaction data record match any of the normalized transaction data record template, based on the comparison, wherein a map is generated between fields of the non-normalized transaction data record and normalized transaction data record template;

associate, via the processor, attributes to the entity by drawing inferences related to the entity from the data field types and their associated data values;

generate an updated profile and social graph of the entity using the generated correlations and associated attributes; and

provide the updated profile and social graph for an automated web form filling request.

35. The medium of claim 34, further comprising instructions to:

generate a search query using the updated profile; and

provide the generated search query for further data aggregation.

36. The medium of claim 35, wherein the search query is a web search query.

37. The medium of claim 35, wherein the search query is a social search query.

38. The medium of claim 35, wherein the search query is an email data aggregation query.

39. The medium of claim 37, wherein the updated profile includes a social login credential; and wherein the social search query utilizes the social login credential.

40. The medium of claim 34, further comprising instructions to:

generate a search query using the updated social graph; and

provide the generated search query for further data aggregation.

41. The medium of claim 39, wherein the search query is a web search query.

42. The medium of claim 39, wherein the search query is a social search query.

43. The medium of claim 41, wherein the updated profile includes a social login credential; and wherein the social search query utilizes the social login credential.

44. The medium of claim 34, wherein the entity is one of: an Internet Protocol address; an individual; a pair of associated individuals; and a household; an office space; and an organization.