A dynamic information gateway management system is provided to translate messages between heterogeneous computer applications. The information management system listens for transmitted messages of predetermined types and when a message is found, takes it into the system and performs a translation using stored metadata and configuration information. Each transaction is graphically modeled and the model is stored in the configuration information. The substance of the message is maintained such that the form and structure of the message is translated but the meaning of the message remains the same.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Albania</td>
<td>ES</td>
<td>Spain</td>
<td>LS</td>
<td>Lesotho</td>
<td>SI</td>
<td>Slovenia</td>
</tr>
<tr>
<td>AM</td>
<td>Armenia</td>
<td>FI</td>
<td>Finland</td>
<td>LT</td>
<td>Lithuania</td>
<td>SK</td>
<td>Slovakia</td>
</tr>
<tr>
<td>AT</td>
<td>Austria</td>
<td>FR</td>
<td>France</td>
<td>LU</td>
<td>Luxembourg</td>
<td>SN</td>
<td>Senegal</td>
</tr>
<tr>
<td>AU</td>
<td>Australia</td>
<td>GA</td>
<td>Gabon</td>
<td>LV</td>
<td>Latvia</td>
<td>SZ</td>
<td>Swaziland</td>
</tr>
<tr>
<td>AZ</td>
<td>Azerbaijan</td>
<td>GB</td>
<td>United Kingdom</td>
<td>MC</td>
<td>Monaco</td>
<td>TD</td>
<td>Chad</td>
</tr>
<tr>
<td>BA</td>
<td>Bosnia and Herzegovina</td>
<td>GE</td>
<td>Georgia</td>
<td>MD</td>
<td>Republic of Moldova</td>
<td>TG</td>
<td>Togo</td>
</tr>
<tr>
<td>BB</td>
<td>Barbados</td>
<td>GH</td>
<td>Ghana</td>
<td>MG</td>
<td>Madagascar</td>
<td>TJ</td>
<td>Tajikistan</td>
</tr>
<tr>
<td>BE</td>
<td>Belgium</td>
<td>GN</td>
<td>Guinea</td>
<td>MK</td>
<td>The former Yugoslavia</td>
<td>TM</td>
<td>Turkmenistan</td>
</tr>
<tr>
<td>BF</td>
<td>Burkina Faso</td>
<td>GR</td>
<td>Greece</td>
<td>ML</td>
<td>Mali</td>
<td>TR</td>
<td>Turkey</td>
</tr>
<tr>
<td>BG</td>
<td>Bulgaria</td>
<td>HU</td>
<td>Hungary</td>
<td>MN</td>
<td>Mongolia</td>
<td>TT</td>
<td>Trinidad and Tobago</td>
</tr>
<tr>
<td>BJ</td>
<td>Benin</td>
<td>IE</td>
<td>Ireland</td>
<td>MR</td>
<td>Mauritania</td>
<td>UA</td>
<td>Ukraine</td>
</tr>
<tr>
<td>BR</td>
<td>Brazil</td>
<td>IL</td>
<td>Israel</td>
<td>MW</td>
<td>Malawi</td>
<td>UG</td>
<td>Uganda</td>
</tr>
<tr>
<td>BY</td>
<td>Belarus</td>
<td>IS</td>
<td>Iceland</td>
<td>ML</td>
<td>Mali</td>
<td>US</td>
<td>United States of America</td>
</tr>
<tr>
<td>CA</td>
<td>Canada</td>
<td>IT</td>
<td>Italy</td>
<td>NE</td>
<td>Niger</td>
<td>UZ</td>
<td>Uzbekistan</td>
</tr>
<tr>
<td>CF</td>
<td>Central African Republic</td>
<td>JP</td>
<td>Japan</td>
<td>NL</td>
<td>Netherlands</td>
<td>VN</td>
<td>Viet Nam</td>
</tr>
<tr>
<td>CG</td>
<td>Congo</td>
<td>KE</td>
<td>Kenya</td>
<td>NO</td>
<td>Norway</td>
<td>YU</td>
<td>Yugoslavia</td>
</tr>
<tr>
<td>CH</td>
<td>Switzerland</td>
<td>KG</td>
<td>Kyrgyzstan</td>
<td>NZ</td>
<td>New Zealand</td>
<td>ZW</td>
<td>Zimbabwe</td>
</tr>
<tr>
<td>CI</td>
<td>Côte d’Ivoire</td>
<td>KP</td>
<td>Democratic People’s</td>
<td>PL</td>
<td>Poland</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CM</td>
<td>Cameroon</td>
<td>KR</td>
<td>Republic of Korea</td>
<td>PT</td>
<td>Portugal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CN</td>
<td>China</td>
<td>KZ</td>
<td>Kazakhstan</td>
<td>RU</td>
<td>Russian Federation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CU</td>
<td>Cuba</td>
<td>LC</td>
<td>Saint Lucia</td>
<td>SD</td>
<td>Sudan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CZ</td>
<td>Czech Republic</td>
<td>LI</td>
<td>Liechtenstein</td>
<td>SE</td>
<td>Sweden</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DE</td>
<td>Germany</td>
<td>LK</td>
<td>Sri Lanka</td>
<td>SG</td>
<td>Singapore</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DK</td>
<td>Denmark</td>
<td>LR</td>
<td>Liberia</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EE</td>
<td>Estonia</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DYNAMIC INFORMATION GATEWAY MANAGEMENT SYSTEM

FIELD OF THE INVENTION

This invention relates to exchanging data among heterogeneous computer systems and more particularly to exchanging messages across networks having heterogeneous computer systems using different data formats, communication protocols, and database technologies.

BACKGROUND OF THE INVENTION

The movement of data across the Babel of computing platforms and applications that comprise the typical enterprise IT infrastructure is difficult, and over time it has caused the creation of spaghetti-like computing architectures interconnecting "stove pipe" legacy and client/server applications. This complex architecture significantly inhibits the ability of the organizations to share information among all decisions makers.

An important element in industry and Government is that these "stove pipe" legacy systems and computing infrastructure, represent several hundred of billions of dollars in investment of public or private funds which neither Government or industry can discard. On world-wide basis this investment may well exceed trillion of dollars.

It is estimated that over 60% of all application development activities are attributed to specialized work to make systems interface and to migrate new functionality into "open" software architectures using client/server technologies. Ten plus years after the onset of the client/server revolution, "the corporate mainframe" is still in place and is the repository of key business data that exists nowhere else in the enterprise.

Even if all legacy systems suddenly were to disappear, the "open, client-server" world has not turned out the
panacea that was initially promised. The protracted and still very incomplete "open, client/server" revolution has left most large organizations with a welter of computing platforms and interconnecting systems which work together only with difficulty and at substantial programming costs.

The problem is only compounded when an organization attempts, for sound business reasons, to communicate between internal systems and those of external firms such as customers or vendors, in which systems must reliably exchange messages whose content is critical to the core business.

There is a movement toward standardized IT messages (such as S.W.I.F.T. for international data exchange of financial transactions and X.12 for data exchange relating to integrated logistics), and towards open mechanisms for the guaranteed transport of data among heterogeneous systems. This can sound like a complete and elegant solution until one realizes that the interfacing system at both ends of data exchange process must have custom code to enable the data exchange. The "interface" custom code of the interfacing applications must provide for the translation and transformation of the input data (meaning and structure) into data forms and content that the receiving system can process. This approach is relatively straightforward the first time. The problems begin when message formats change or new messages are added to the mix, with the result that the transacting applications on each end of the interface cycle, is in a continual state of costly changes. Further complicating this problem is the fact that the engineers who developed the original interface code may no longer be employees of the organization, and that system documentation is misplaced or out of date.

Global competition is also forcing organizations to reexamine all aspects of how they use computing technologies
to accomplish their goals. Reduction in GATT tariffs and the onset of Global commerce enabled by Internet, also act as multipliers to the demand for increased exchanges of business information among geographically dispersed organizations. This ever increasing demand for "data exchange" among organizations and geographically dispersed heterogeneous systems make the task for "data interoperability" especially important. It remains desirable to have efficient data exchange between heterogeneous systems in a networked computing environment.

It is an object of the present invention to provide a method and apparatus to translate and transform data among normally incompatible applications and database management systems.

It is another object of the present invention to provide a method and apparatus to provide a graphical user interface that makes such translation easy to define and to perform.

It is another object of the present invention to provide a method and apparatus providing a Visual Language and a Palette of visual operators in a single authoring tool, and providing visual mapping mechanisms to define in a single logical step all required data manipulation and business process functions (i.e., data translation algorithms, internetworking of applications, and encapsulation of business rules).

It is another object of the present invention to provide a method and apparatus that presents comprehensive views for remote monitoring and management of the data exchange activities.

It is another object of the present invention to provide a method and apparatus enabling interoperability without the need to modify the programs of the interfacing systems.
It is another object of the present invention to provide a method and apparatus for the exchange of data among incompatible networks and data communication protocols without the need to write custom code to achieve network connectivity.

SUMMARY OF THE INVENTION

The problems of data exchange among heterogeneous systems in a networked computing environment are solved by the present invention of a dynamic information gateway management system (DIGMAS).

The dynamic information gateway management system addresses the major problem besetting the Information Technology (IT) functions of today which is making data available to decision makers at all levels of the organization in a convenient form and at the time the data is needed.

The present invention exchanges data among two or more computer systems that are heterogeneous. Heterogeneous systems are systems that create, store, and process dissimilar data using different computing platforms, communications and database management software. Heterogeneous systems receive and transmit incompatible data messages. The term "data message" is broadly defined as any textual or binary, fixed or variable length format message transmitted over a computer network, including, but not limited to, "messaging systems", electronic mail, and database communications. A message includes discrete strings of data, files that contain records, or discrete records and database calls and files (e.g. SQL statements for database transactions).

The invention has of three basic components: a) an integrated metadata repository and authoring tool, b) an administration and reporting component, and
c) networking and transformation engines.

The object of the integrated metadata repository and authoring subsystem is the creation and maintenance of a "metadata" repository. This subsystem incorporates a graphical user interface (GUI) apparatus that contains a visual language (VL) with its palette of visual operators. The GUI and the VL enable the user to define the following metadata: 1. the dictionary, 2. mapping, 3. the directory, and 4. the visual language (VL).

The dictionary contains descriptions of message standards, message types and instances, and database schemas of the enterprise applications; data structures of all messages (both hierarchical text or binary based, fixed or variable length format, and database transactions) exchanged among the interfacing systems and databases; parsing and validation standards that define the process and logic to parse and validate all messages from the input form into an internal representation and unpars the data from the internal representation into the desired output formats; and descriptions of data domains and standard functions involved in data transformations.

The mapping contains data "rules" (maps between source and target data elements) defining data moves and or generation of data from source data into the target data fields; initiation "rules" defining the correspondence between the target data object groups and the source data object groups; and assembly (i.e., algorithms) of all mapped rules (data and initiation) required to be applied for the data transformation. A unique assembly is required for each interface.

The directory contains a description of all applications and application instances involved in data exchange; a description of data communication protocols and the communication parameters used by each application instance
for network connectivity (network address and protocol); and a description of events and actions required for processing each message. An action is an algorithmic assembly of maps (data and business process rules) and of list of messages (both input and output). Messages may be associated with any number of actions. An event is a super assembly of the actions and of other events to be executed by the network and transformation engines. The event also includes definitions as to the method of its initiation and the destination/target receivers of the outputs to be generated as a result of the processing of the event. The directory also contains an encapsulation of “work-flow” rules that define the sequence of events and actions to be executed. The “work-flow” sequence simulates the data flows needed to service a specific business process.

The visual language and its palette are an integrated function of the invention. The object of this visual language and palette is to enable the definition of the translation and transformation logic (rules and algorithms) without having to write procedural or command-level code. The visual language and its palette instantly make visible to the user the processing logic encoded to describe a message transformation. Unlike procedural and command-based languages, the VL expresses the data manipulation functions and encapsulates business rules into processing metadata objects or group of objects in a single step using “flow charting” techniques.

The object of the administration and reporting subsystem is to provide appropriate controls and reporting to the administrator and users of the invention. The controls are needed to ensure optimum performance and reliability of the invention during operational state. This subsystem enables the user to select and package the “authored” metadata (i.e., transformation, validation, and
business rules) into configuration files. The configuration files are a collection of meta-object classes that are loaded into the smart interface during its initialization. The configuration file in effect "programs" at run-time the smart interface. There are three groups of meta-object classes: a) metadata objects, b) actual-data objects, and c) managing objects. A list of these objects can be found in Appendix A. This subsystem also facilitates the initiation, monitoring and management of the networking and transformation engines. This subsystem also facilitates the transmission of the configuration files to any number of copies of the networking and transformation engines. This subsystem also provides for the authorization and termination of users access to the repository to author/modify metadata and configuration files. This subsystem also facilitates the test and validation of authored configuration files or of rejected messages. Finally, this subsystem generates metadata repository reports including lists of the contents of the configuration files and of the repository.

The networking and transformation engines constitute a network server subsystem called the smart interface. The smart interface acts as a middleware server and provides for seamless data interchange among the interfacing systems with guaranteed delivery of messages from source to destination. The smart interface operational and processing behavior is governed by the metadata-objects contained in the configuration files. These files are transmitted to the smart interface during its initialization and its operation can be terminated or altered remotely by the administrator. The smart interface re-initialization process provides for the dynamic update of the functions of the smart interface in seconds as compared with traditional "network" gateways and "translation" engines that require off-line
reconfiguration which may take several hours to days for both re-engineering and testing.

The networking component is the “gateway” application and provides a routing function consisting of the initialization and management of inbound and outbound queues. The routing function has “listener and dispatching” subfunctions that constantly interrogate the “sockets” and internal stores for inputs and ready-to-be-transmitted output messages. This function will pass the input data and initiate the process of an event, or it will dispatch the generated output(s) from events to the server’s operating system for transmission. The networking component provides for back-up and recovery which provide data integrity and automated recovery of the smart interface operations. The networking function also provides an event manager that coordinates the concurrent processing of events. The smart interface is a multithreading application. The number of concurrently processing threads is limited only by the computer’s operating system. The event manager monitors the progress of each event as it is processed, and at different stages of processing (e.g. in the input queue, in process, and in the output queue) the event manager stores the message data in temp-files. These files are used during restart in case of a failure or interrupt.

An event is initiated by an input or internally generated message. A message can be either a “scheduled trigger” send by an interfacing system that causes the smart interface to “poll” a specific network directory file or database, or it can be a message that is generated by another event. The event processing is performed based upon the actions and subevents defined in the configuration file objects. If the incoming messages are not recognized (i.e., not defined in the configuration file objects) or fail parsing or validation, then the event will return the
rejected message to either the originating application or other designated destinations with appropriate explanation as to the reasons for terminating the process.

The processing component of the smart interface is comprised of several data transformation objects. The functions of the processing component are parsing, validation, translation, internal posting, unparsing (i.e., formatting) and generation of database transactions.

A normal process will result in one or more output messages forwarded to the routing function and its dispatching subfunction to be forwarded to the LAN Server queuing services.

As stated earlier in this petition, the behavior/operation of the smart interfaces is governed by the objects contained in the configuration file generated from metadata in the repository. The objects contained in the configuration file are listed in Appendix A. The functions of the meta object classes contained in the configuration files are as follows. Metadata objects represent entities in the metadata repository and contain comprehensive cross references of each other. They encapsulate methods enabling the automated processes of all the smart interface modules. Actual-data objects are used to create internal hierarchical data trees of the input and output messages. They contain all the needed information about data structures, values and meaning of the data in input and output messages. Managing objects are used to provide access to network, connectivity to databases and interaction with the operating system services.

The present invention together with the above and other advantages may best be understood from the following detailed description of the embodiments of the invention illustrated in the drawings, wherein:
5 BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram of a preferred embodiment of a dynamic information gateway system showing data flow according to principles of the invention;

Figure 2 is the block diagram of Figure 1 with expanded detail of the configuration utility;

Figure 3 is a block diagram of the data interface of the preferred embodiment of the invention of Figure 1, the diagram also showing process flow in the data interface;

Figure 4 is a flow chart of the process flow of the data interface of Figure 3 according to principles of the present invention;

Figure 5 is a diagram of the basic functionality of the graphical user interface (GUI) of the mapping facility (i.e., the visual language and its palette) of the dynamic information gateway system according to principles of the present invention;

Figure 6 is a flow chart summarizing the operation of the preferred embodiment according to principles of the invention;

Figure 7a is a first part of a database schema for the dynamic information gateway management system according to principles of the invention;

Figure 7b is a second part of the database schema of Figure 7a for the dynamic information gateway management system according to principles of the invention;

Figure 7c is a third part of a database schema Figures 7a and 7b for the dynamic information gateway management system according to principles of the invention; and,

Figure 7d is a fourth part of a database schema of Figures 7a, 7b and 7c for the dynamic information gateway management system according to principles of the invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS
While the presented invention is useful within a wide variety of configurations and may be embodied in several different forms, it is most advantageously employed in connection with WANs and LANs. Though this is the form of the preferred embodiment and will be described as such, this embodiment should be considered illustrative and not restrictive.

Referring now to Figure 1, there is shown a first computer application 10, a first data message 12, an information management system 14, a second data message 20, and a second computer application 22. The information management system 14 has a configuration utility 18 and a data interface 16. The first computer application and the second computer application may run in separate computer systems or in a second alternative embodiment they may run in the same computer system. The information management system may be in the same computer device as either the first or the second application or it may run on a computer system of its own.

The first computer application 10 may be, for example, a computer system such as a server; a network, such as a local area network (i.e., a LAN); a wide area network (i.e., a WAN); or even the Internet. There is no restriction as to the type of computer or the computer network the first computer application 10 represents. All that is required is the intercommunication performed by the first computer application 10 such that the first computer application 10 transmits a first data message 12. The second computer application 22, as with the first computer application 10, may also be any one of a number of computer types. The second computer application 22 must be able to receive data messages.

The first data message 12 may have a structure that is one of any of numerous structures, but the message structure
must be one that is known to the information management system 14. The structure and format of the first data message 12, will be generally governed by the type of the first computer application 10. The invention that will be herein described is not limited to just conventionally known message data types nor is the invention limited to use with commercially known computer systems and applications. The requirement is simply that the data message has a type which has a consistent structure and which can be profiled by placing the structure in a configuration file as will be described hereinafter. For example, the first data message 12, if transmitted over a TCP/IP network, has a structure that conforms to the transmission control protocol ("TCP") and the Internet protocol ("IP"). These two protocols have known and well-defined structures where header information includes various sub-elements such as message type and addressing information. The TCP/IP message can be profiled in a configuration file using the header and sub-elements of the message type.

The first data message 12 is transmitted across a computer network in a manner well known in the art. The first data message 12 travels across the computer network and is intercepted by the data interface 16. Though not shown, the computer network is a system of computers interconnected by wires, infrared communicators, or any of various other means in order to share information. The arrow marked as the first data message 12 is indicative of transmission over such a computer network which may include numerous computer systems or may be internal communication in a single computer. For example, if the first data message 12 is transmitted by the same computer having the information management system 14, as described later herein, then the transmission is substantially internal to that computer. Otherwise, the information management system 14
may be disposed on a computer system remote from the first computer application 10 and may be one of many such computer systems attached to the computer network.

The configuration utility 18 associated with the data interface 16 contains a database of the structures of most known message types. For example, the message structure of a TCP/IP message which was used as an example for the first data message 12 would be contained in the configuration utility 18.

The data interface 16, uses information obtained from the configuration utility 18 to parse, or break apart, the data message 12 received from the sending system 10. By doing so, the substantive information contained within the first data message 12 can be removed and stored as an entity independent of the message structure.

As previously described, the first data message 12 is associated by the information contained in the configuration file to a data message 20 and to a receiving system 22 and the receiving system’s 22 network type and address. The network address information is descriptive of the device to which the first data message 12 is being transmitted. Continuing the previous example, the addressing information in a TCP/IP message would consist of an IP address. The IP address is simply a numeric value assigned to a computer much like a telephone number is assigned to a home. When a packet of data travels from one computer to another, the packet contains the addresses of both the transmitting and the receiving computer. The information management system 14, through its configuration utility 18, contains information related to each of the known devices on the computer network. Thus, the IP address contained in the first data message 12 can be referenced within the configuration utility 18 to determine a type of device to which the first data message 12 is being addressed.
The configuration utility 18 transmits the message structure information to the data interface 16 at the request of its administrator (i.e., the software engineer responsible for its operation). The data interface 16 then uses the information transmitted to it by the configuration utility 18 to build the second data message 20 containing the substance of the first data message 12 in a form and structure compatible with the protocols required by the second computer application 22.

In the manner described above, heterogeneous systems such as the first computer application 10 and the second computer application 22 intercommunicate across a network through the information management system 14 which acts as a universal message translator.

Figure 2 is the block diagram of Figure 1 with expanded detail of the configuration utility 18. The configuration utility 18 has configuration files 30, a metadata repository 32, and a graphical user interface (GUI) 34. The GUI 34 and the data interface 16 communicate via a control interface 36.

As previously described, when the first data message 12 is passed into the data interface 16, it uses information transmitted to the data interface 16 from the configuration utility 18 to process the first data message 12 into the second data message 20 and sends the second data message 20 to the destination computer system 22. The transfer is made through a monitor control interface 36 which bi-directionally interfaces with the data interface 16. The monitor control interface is part of the GUI 34. In this instance, the direction of data flow is from the GUI 34 where information pertaining to the methods for the transformation and transmission of the first data message 12 into the second data message 20 by the data interface 16, is
passed to data interface 16 in the form of configuration file 30.

The software engineer uses GUI 34, to access the metadata repository 32. The metadata repository 32 is a database of metadata related to each of the various message types and interfacing systems including network protocols and addresses. Metadata are essentially high-level data that describe the structure, content and meaning of data. The metadata objects can be found in Appendix A. Metadata contains information such as message and record formats, the schema of databases for interfacing applications, data communication information (i.e. network addresses and protocols), translation algorithms, and business rules. The metadata repository 32 is a database of message types that include each of the substructures of each of the known message types in the information management system 14. With respect to the previous example, the structure of a TCP/IP message would be contained within the metadata repository 32.

The configuration files 30 contain all mapping algorithms (i.e., methods), system definitions, directories, lookup tables and other things needed by the information management system 14 to perform its functions.

In the information management system 14, the second computer application 22 is specifically defined as interfacing across the network and information related to the second computer application 22 is stored within the configuration files 30. If the second computer application 22 is an SQL server, then that is stored within the configuration files 30. Upon receipt of the request through the monitor control interface 36, the structure of an SQL message is recognized by the data interface 16 through the information contained in the configuration file 30 that was transmitted to it. The configuration file containing the
message structure for data message 20 is then used by the data interface 16 and the second data message 20 is built as previously described.

The GUI 34 serves an authoring, monitoring function as well as a control function. The GUI 34 provides numerous facilities by which the configuration files 30 and metadata stored within the metadata repository 32 can be edited. The GUI 34 also provides a facility by which the data interface 16 can be monitored and independently controlled. The GUI 34 will be described more fully in Figure 5 below.

Figure 3 is a block diagram of the data interface 16 showing process flow in the data interface 16. The data interface 16 has a listener 40, an input queue 48, a message processor 50, an output queue 52, and a dispatcher 54. The listener 40 has a plurality of sub-listeners, for example a TCP/IP listener 42, a database listener 44, and a mail listener 46. One skilled in the art will realize that this set of sub-listeners is but a small subset of the numerous message types for which the sub-listeners can be configured and are thus exemplary. The dispatcher 54 has a plurality of sub-dispatchers, for example a TCP/IP dispatcher 56, a database dispatcher 58, and a mail dispatcher 60. One skilled in art will realize that this set of sub-dispatchers is but a small subset of the numerous message types for which the sub-dispatchers can be configured and are thus exemplary.

The sub-listeners operate to intercept predetermined types of messages. For example, the TCP/IP listener 42 monitors the network for TCP/IP messages being transmitted to the host server of the data interface. Likewise, the database listener 44 monitors the network for predetermined types of database messages (i.e., database calls and SQL statement), such as the aforementioned SQL message type. The mail listener 46 listens for electronic mail messages.
Upon the interception of a message for which the sub-listeners are configured, that message is passed into the input queue 48. The input queue 48 is a sequential buffer queue for incoming messages until the message processor 50 can process the messages.

The message processor 50 is a multithreaded engine that used the information contained in the configuration file to break apart and subsequently rebuild the output data message(s).

Once the input message(s) is transformed into the output message(s), the output message(s) is passed to the output queue 52. The output queue 52 is like the input queue 48 in that the output queue 52 is simply a sequential buffer by which messages are held and transmitted on a first in-first out (FIFO) basis. Once the output messages stored in the output queue 52, the dispatcher 54 uses the network and address information of the destination (i.e. receiving) system for the output message (the network and address information is contained in the configuration file) and passes the out message to its sub-dispatchers for transmission.

In general, there is a one-to-one correspondence between the types of sub-listeners and sub-dispatchers. This construction provides bi-directional communication between devices. That is, if there are one or more databases defined on the network, there must be a sub-listener that processes incoming database messages and a sub-dispatcher that can process output messages so that a full input / output database connectivity is achieved.

Figure 4 is a flow chart of the process flow of the data interface shown in Figure 3. An event is initiated when a message is received at the data interface, block 62. The input message is parsed by the message processor, block 64, and if there is an error, the error message is addressed
to the receiving system, block 80, to be dispatched as an error. If there is no error after parsing, the input message is validated based on rules in the dynamic information management system, block 66. If there is an error in validation, the error message is addressed to the receiving system, block 80, to be dispatched as an error. If there is no error after the validation step, then the message processor determines whether additional data is needed to process the message, block 68. If no additional data is needed, the message is translated by the message processor, block 72. If additional data is needed, the message processor accesses the appropriate database for the data as specified in the message, block 70, and then translates the message into an output message, block 72. The output from translation, block 72, can be one or more messages. After translation, each output message is validated, block 74. If there is an error, an error message is addressed to the receiving system, block 80. If there is no error after validation, each output message is addressed to the receiving system(s), block 80. Addressed messages are then transferred to the output queue, block 82, and to the appropriate dispatcher, block 84. An output message is used as input to initiate another event called a sub-event, block 78, if so addressed. The message processor determines whether the event is complete, block 76, when all messages are delivered and all sub-events are initiated.

Figure 5 is a diagram of the basic functionality of the graphical user interface (GUI) 34. The GUI 34 facilitates the modeling of the message translation transactions and so enables a dynamic, configurable interface among heterogeneous systems and databases that communicate using messages. Each transaction is modeled using the GUI 34. The GUI 34 allows users to specify message formats and database schema, define data and message transformations
(mappings). The GUI 34 is used not only to create the transaction model, but also the configuration files 30 which contain the transaction data. The configuration files 30 contain the metadata that define the network components and how to communicate with them and define translation algorithms between messages that have been modeled. Using the GUI 34, messages are converted into patterns that can be translated into other patterns.

The GUI 34 has a series of frames, each providing a subset of the data necessary to create relationships that determine the input and output structure that will be employed.

An output standard/family-type window 92 allows a user to choose the available output standards on the network from a hierarchical list. Below the output standard/family-type window 92 is an output mapping context window 94. Below the output mapping context window is an output data elements window 96 which allows the user to select the structure of the output message from a hierarchical list.

In a column next to the three output windows, 92, 94 and 96, previously described, there are input windows 98, 100. An input standard/family-type window 98 contains the available input standards on the network in the form of a hierarchical list. Below the input standard/family-type window 98 is an input data elements window 100.

To create a data flow algorithm which models the translation transaction, a user moves the appropriate blocks into the mapping window 102, and then connects the inputs and outputs with operators. That is, once a structure of an input message is determined using the input windows 98, 100, functions which must be performed on the message in order to get the proper output message can be mapped graphically through the mapping window 102.
Below the mapping window 102 is a control window 134, having control buttons disposed therein. The control buttons are simply functional controls relating to the mapping. For example, the buttons shown include the functions of saving, viewing and clearing. Other buttons can, of course, be included without detriment to the invention.

Below the control window 134 is a blocks window 126. The blocks window 126 is selective between different types of blocks windows. That is, depending upon the type of mapping that will be performed, the contents of the blocks windows 126 will be altered. Field-type tabs 124 select the type of message. The blocks window 126, as the name implies, contains the blocks which build the translation transaction. Each type of message has its own set of blocks and the blocks window allows the user to choose from among the sets of blocks available in the information gateway system 14. One skilled in the art will realize that the available tabs or types of data are limited only by the types of data contained within computer systems and, therefore, each of the above examples should be considered illustrative and not restrictive.

To create a translation transaction model, blocks from the blocks window 126 are selected, then dragged and dropped into the mapping window 102. Connections are then drawn between the blocks to determine data flow between an incoming message substructure and an outgoing message substructure. Forming part of the connections between the blocks are manipulating operators. Each of the operators perform a function on the contents of the blocks and take a predetermined set of inputs to establish one or more outputs. Examples of these operators in text manipulation translations are a LEN function, for determining a length of a string; a PADL function, for padding a string to the left;
a PADR function, for padding the string to the right; an
STRPL function, for stripping a string to the left; and an
STRPR function, for stripping a string to the right. These
are simply examples, of course, and numerous functions can
be employed as is well-known in the art.

Each of the blocks and the operators have inputs and/or
output ports (small round circles). Though not illustrated
as such, the ports are color coded according to the data
type associated with the port (green for text, indigo for
integer, red for real, and blue for Boolean, for example).
The ports located at the top of the token (block or
operator) are input ports and the ports located at the
bottom of the token (block or operator) are output ports.

If another field-type tab 124 is chosen in the blocks
window 126, then the types of blocks will vary. Examples of
types of blocks are: text, integer, real, and Boolean. In
the case of text literals, as was previously exemplified,
the user types a string that represents the text literal
data object into the block, which is a rectangle that
represents the object. For integer literals, the user types
an integer that represents the integer literal into the
rectangle that represents the object. For real number
literals, the user types a real number that represents the
real literal into the rectangle that represents the object.
For Boolean literals, the user types the truth value
represented by the Boolean literal data object into the
rectangle that represents the object. In the preferred
embodiment, "T" represents "True" and "F" represents
"False."

For input and output fields, the field is specified by
the name of the element's message type followed by the name
of the element. The user selects these names via the input
standard/family-type window 98 and the output
standard/family-type window 92, respectively.
Operators are field-type specific. As with the example above, text fields have a predetermined set of operators with which they operate. Other field types have their own set of operators. For example, integer operators (numeric operators that have integer inputs and outputs) include addition, subtraction, multiplication, division, and modulo. Real operators (numeric operators that have real inputs and outputs) are limited to addition, subtraction, multiplication, and division. Boolean operators (logical operators that have truth-value inputs and outputs) are represented by NOT, RND, OR, XOR having the meanings well known in the art. Bitwise operators (logical operators that have integer inputs and outputs) have the same operators as the Boolean operators.

Comparison operators present a different class of operators that will vary again by data type. Real and Text comparison operators (comparison operators that have truth-value outputs and real or text inputs, respectively) are identical except for the color of their input ports, which are red and green, respectively.

Selection operators allow a choice between input or output paths based upon a decision criteria. Real, Boolean, and Text switch operators (switch operators that are identical to the integer switch operator except that they have real, truth-value, or text left and right inputs and outputs, respectively) are identical except for the color of their left and right input ports and output ports, which are red, blue, or green, respectively.

The table lookup operator allows a user to select the name of a pre-defined table from a drop down list when the button near the block is clicked. A table consists of multiple entries. Each entry contains a match item and a result item. No two entries may have identical match items.
Type Conversion Operators (also known as a Pipe operator) govern type conversion where the first half of the operator is the color associated with the data type before type conversion and the second half of the operator is the color associated with the data type after conversion, where the colors Red, Blue, Green, and Indigo are associated with the data types Real, Boolean, Text, and Integer, respectively.

Figure 6 is a flow chart of the message translation process. A first message is sent out to the network from a first computer system, block 250. The data interface listeners intercept the message, block 255, and pass it on to the input queue in the data interface, block 260. From the input queue, the message is passed to the message processor, block 265. The message process gets the information from the configuration files 30 used in breaking down the message and rebuilding it. The message is then passed to the output queue, block 270. From the output queue, the message is passed to the unit dispatcher 54 which breaks down the first message and rebuilds it into a second message in a new format, block 275. The unit dispatcher 54 separates the substance of the message from its format. Then the message is passed onto the dispatcher which sends the message out to the second computer system, block 280.

Figures 7a, 7b, 7c and 7d show an entity relationship diagram for the dynamic information gateway management system according to principles of the invention. Lines connecting the entities in the drawings are numbered 300 - 337. Lines having the same numbers in the Figures 7a - 7d are connected.

Referring to Figure 7a, an action entity 400 is an entity in the database that defines the translations to be performed after an event 410 has been initiated. Action 400 links into logical state input messages, output messages and
set of maps 468, containing business and data translation rules.

An action map entity 402 defines a relationship between action 400 and maps 468 that defines transformation and correspondence between input data fields, data domains and output data fields.

An action message entity 404 defines input and output messages for an instance of action 400. Action message 404 also defines the application instance 424 where the input and output messages persist.

A communication protocol entity 406 defines the network protocols used by an application instance 424 to exchange data with the smart interface. The communication protocol structure specifies the set of parameters needed: a) to define the networking connectivity of an application instance 424; and b) the parameters needed to define a network event.

A CP parameter (Communication Protocol Parameter) entity 408 specifies a name of a parameter that should be set for particular Communication Protocol (such as "IP address").

An event structure 410 defines a step in the processing of a message. There are three types of events: a) a network event that is initiated by the arrival of a message via a network; b) a scheduled event that is initiated when a scheduled "trigger" occurs; and c) an internal event that is initiated by another event as part of a logical work-flow representing a business process.

A subevent entity 412 defines a parent/child relationship between an event 410 that is performed first and subsequently initiates other events.

An event triggering message entity 414 defines a three-way relationship between the event structure 410, the message entity 418, and the action structure 400. For each
event 410, the event triggering message identifies the messages that can initiate the event 410, and the actions 400 to be executed.

A logical restriction rule entity 416 identifies a clause that must be evaluated to determine if the associated action 400 needs to be executed and the evaluation is based on the data of the incoming messages.

A message/transaction entity 418 defines one unit of communication between the smart interface and application instances 424 involved in data exchange. Internally to the database, a message is represented as a hierarchical tree of objects and attributes. Externally, it is accepted/sent as textual or binary data or database transactions.

A parsing standard entity 420 defines a universal means of expression of rules for parsing and unparsing of the messages. It defines initial and determination rules for the structural element of the message. One parsing standard can be attached on various levels of abstraction: object type, object, data element, attribute, and relationship. It can be used by many of the above items or it can be created specifically for only one of these items. Each structural item can receive its parsing information from one or many parsing standards (for example prefix from one parsing standard and separator from the other).

A transaction parameter entity 422 defines additional data that can be passed to a database transaction to be used for the generation of SQL statements.

Referring now to Figure 7b, an application entity 424 defines particular a computer program or a system that exchanges messages of a particular standard.

An application instance entity 426 defines installation of an application on a particular computer or network node. Each application instance is defined by communication parameters.
A buffer definition entity 428 defines the buffer used by the SI to exchange data with external sources. The main characteristics of the buffer are: the size of byte (i.e., 7 bit, 8 bit or one bit), max length of line, and line break indicator.

An CP parameter value entity (Communication Protocol Parameter Value) 430 specifies the value of a communication protocol parameter for a specific application instance 426 or event 410 (e.g., “122.10.33” or “ons@abxxxxzzz.com”).

An event directory entity 432 identifies source for incoming messages or destination for outgoing messages for an event. Source and destination can be external (application instance 426) or internal (subevent 412 i.e. another event).

A family entity 434 defines a family of standards that have common data types, object types and format definitions (for example “SWIFT Messages”, “EDI X.12 Messages”, or “ORACLE Databases”).

A format definition entity 436 defines how “native” format symbols of a family correspond to a regular expression.

A standard entity 438 represents a grouping of messages inside a family 434. Some families can have only one standard while others have a standard that represents variations of the data structures between versions of the family (such as versions of the same software package or message standard).

Referring now to Figure 7c, the attribute entity 440 defines the type of data fields that could be present in an object. An object can be part of a messages or a single record in a database transaction. Attribute 440 relates an object to its corresponding data element(s). For database tables, it corresponds to a database column. For transactions, it corresponds to a transaction field.
A data domain entity 442 specifies the "set" that data may belong to. It could be a broad set (such as integer number) or restricted set specifying a list of values. Domain is defined by format, verification function or list of permissible values.

A data element entity 444 defines the dictionary element used by a standard 438 to construct messages. Each data element 444 can be used across multiple messages of the standard 438 to identify atomic pieces of data that have common meaning, format and verification rules.

A data type entity 446 defines "native" data types for each family 434 (such as database or language data types).

A domain mapping entity 448 defines the relationship between domain 442 and data type 446.

A domain value 450 defines a specific value from a list of values within a domain 442.

A foreign key rule entity 452 identifies how attributes 440 of one object 470 correspond to attributes 440 of other objects 470 as a foreign key.

A lookup table 454 is used for translation when data is converted between two data domains expressed by list of values.

A lookup table record 456 identifies correspondence between one value of incoming data domain and one value of outgoing data domain for a particular look-up table 454.

A object type entity 458 defines a logical grouping of object inside a standard. Objects of an object type have common meaning and usage and usually have some common parsing standards 420.

Referring now to Figure 7d, an argument entity 460 defines an argument of a visual language function. An argument can be an attribute 440, a data element 444 or a data domain 442. A determination of "what type" of argument is dependent on the nature of the function.
A clause entity 462 defines all logical restrictions used by the smart interface. The clause is expressed by function. In the present embodiment of the invention, there are four uses for a clause: a) initiation rule group clause; b) initiation rule Having clause; c) action logical restriction, and c) relationship clause.

A functional connection entity 464 defines visual language function, contains function text and identifies input and output arguments. The functional connection structure is used for two purposes: a) to generate values for attributes of outgoing message during translation; and b) to express rules. In the latter case, the result is always interpreted as "TRUE"/"FALSE".

An initiation rule entity 466 defines a rule that expresses when to initiate an object of a message, when the message is generated by translation. For this purpose, corresponding incoming object(s) are identified, packaged using group and Having clauses.

A map entity 468 defines a grouping of functions and initiations rules that can be used in one or more actions. A map can be created at various levels of abstraction: action 400 (most specific level), standard 438, relationship 472, or object 470.

An object entity 470 defines object as a component of a message or database transaction. As a part of a message hierarchy, the object is a group of attributes and relationships to children objects. As part of a database transaction, the object is a single "select, insert or update" statement. As a part of database description, the object represents a single table or a view.

A relationship entity 472 identifies parent/child relationships between the objects for message or database. A select rule entity 474 is used to specify database transactions. Select rule identifies the database
relationship used to attach child records to parent record and the action (SELECT/INSERT/UPDATE) to be taken on a child record.

It is to be understood that the above-described embodiments are simply illustrative of the principles of the invention. Various and other modifications and changes may be made by those skilled in the art which will embody the principles of the invention and fall within the spirit and scope thereof.
## Appendix A

**Meta-data Objects**

<table>
<thead>
<tr>
<th><strong>UniConfiguration</strong></th>
<th>Manages all metadata information contained within the configuration file of Interface instance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MetaAction</strong></td>
<td>Specifies possible translation action to be taken on one or more messages</td>
</tr>
<tr>
<td><strong>MetaApplication</strong></td>
<td>Specifies computer application that could run on multiple locations over network</td>
</tr>
<tr>
<td><strong>MetaApplicationInstance</strong></td>
<td>Particular network location where application is running. Defined by application and its communication parameters.</td>
</tr>
<tr>
<td><strong>MetaArgument</strong></td>
<td>Argument definition for a MetaFunction</td>
</tr>
<tr>
<td><strong>MetaAssociation</strong></td>
<td>Defines a Super Class of the MetaAttribute or MetaRelationship</td>
</tr>
<tr>
<td><strong>MetaAttribute</strong></td>
<td>Association between Object and DataElement in the message</td>
</tr>
<tr>
<td><strong>MetaBuffer</strong></td>
<td>Specifies mode of exchange of between interface and outside applications. Defines parameters such as lengths of byte, line and page for data exchange.</td>
</tr>
<tr>
<td><strong>MetaClause</strong></td>
<td>Association between rules (such as Initiation Rule) and functions that implement it.</td>
</tr>
<tr>
<td><strong>MetaContainer</strong></td>
<td>Defines a Super Class of MetaStandard or MetaDatabase</td>
</tr>
<tr>
<td><strong>MetaDatabase</strong></td>
<td>Represents particular relational database as a collection of MetaObject for DB tables and MetaRelationship for relations</td>
</tr>
</tbody>
</table>
| **MetaDataDomain**    | Specifies general rules for validation of the data element based on abstract meaning of data (such as “text”, “date”, “time”, “money”)
<p>| <strong>MetaDataElement</strong>   | Elementary data dictionary entry. One data element can be used in multiple attribute of messages across message standard. |
| <strong>MetaDomainValue</strong>   | Used for domains restricted by value (such as “month of the year”)                           |
| <strong>MetaElement</strong>       | Defines a Super Class of MetaDataElement or MetaObject                                       |
| <strong>MetaEvent</strong>         | Specifies DIGMAS event. Defines what initiates event (message arrival from particular network location, polling time, etc.). What possible actions to |</p>
<table>
<thead>
<tr>
<th>UniConfiguration</th>
<th>Manages all metadata information contained within the configuration file of Interface instance.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MetaFamily</td>
<td>Family of messages, such as USMTF or SWIFT.</td>
</tr>
<tr>
<td>MetaFunction</td>
<td>Description of a function for Data Conversion or Rule classes. The function body is expressed using Visual Language.</td>
</tr>
<tr>
<td>MetaInitiationRule</td>
<td>Rules used to derive the structure of the out-going message or SQL statement based on the structure of the incoming message or SQL statement.</td>
</tr>
<tr>
<td>MetaItem</td>
<td>Keeps metadata for particular UniElement or UniAttribute object.</td>
</tr>
<tr>
<td>MetaLookupTable</td>
<td>Used when values of two domains have to be correlated (such as 3-letter currency code to 4-letter currency code).</td>
</tr>
<tr>
<td>MetaMap</td>
<td>Collection of MetaFunctions and MetaInitiationRules for particular context of a translation.</td>
</tr>
<tr>
<td>MetaObject</td>
<td>Main building block of messages and SQL statements. A MetaObject represents a node within a structure tree modeling a message or a SQL statement. Associated with the MetaObject are the MetaRelationships and MetaAttributes.</td>
</tr>
<tr>
<td>MetaObjectType</td>
<td>Groups of MetaObject of the same MetaStandard.</td>
</tr>
<tr>
<td>MetaParsingStandard</td>
<td>Specifies parsing rules for various levels of message standard.</td>
</tr>
<tr>
<td>MetaRelationship</td>
<td>Association between parent object and child object in a message or SQL statement.</td>
</tr>
<tr>
<td>MetaSelectRule</td>
<td>Specifies database transaction in hierarchical form. Multi-level database transaction is represented as a tree of MetaSelectRules.</td>
</tr>
<tr>
<td>MetaStandard</td>
<td>Represents standard for messages. Messages that belong to a standard, share data elements, objects and their definitions, parsing standards and validation rules.</td>
</tr>
<tr>
<td>MetaSubEvent</td>
<td>Links event to sub-event (sub-event is an event initiated by the message generated by parent event).</td>
</tr>
<tr>
<td>UniConfiguration</td>
<td>Manages all metadata information contained within the configuration file of Interface instance</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MetaTransaction</td>
<td>Represents database transaction incorporating multiple tables in the database as described by its corresponding MetaDatabase object that in effect describes the database Schema</td>
</tr>
</tbody>
</table>

**Actual Data Objects**

<table>
<thead>
<tr>
<th>Message</th>
<th>Manages all data related to a single message. Message is represented as a tree structure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniItem</td>
<td>Super Class of UniElemet or UniAttribute</td>
</tr>
<tr>
<td>UniElement</td>
<td>It is a node in the message or SQL transaction structure tree. Can have UniAttributes and children UniElements</td>
</tr>
<tr>
<td>UniAttribute</td>
<td>Plays role as a “leaf” in the message or SQL transaction tree. Stores the values that can represent logical or physical edits for actual data</td>
</tr>
<tr>
<td>UniBuffer</td>
<td>Buffer, that keeps text of the message</td>
</tr>
</tbody>
</table>

**Managing Objects**

<table>
<thead>
<tr>
<th>UniEventProcess or</th>
<th>Organizes processing of single event. Receiving all incoming data, selection and execution of actions and distribution of generated messages.</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniParser</td>
<td>Manages parsing of messages / SQL statements from buffer to internal Message representation</td>
</tr>
<tr>
<td>UniTranslator</td>
<td>Manages translation from incoming messages SQL statements to outgoing messages or SQL statements</td>
</tr>
<tr>
<td>UniValidator</td>
<td>Validates Messages in accordance with rules, expressed by the metadata</td>
</tr>
<tr>
<td>MessagePort</td>
<td>Receives and sends messages via various communications modes.</td>
</tr>
<tr>
<td>UniDatabaseMgr</td>
<td>Generates SQL and data sets for messages and database transactions</td>
</tr>
</tbody>
</table>
What is claimed is:

1. A computer-implemented method for exchanging data between a first computer application and a second computer application where the first computer application and the second computer application are heterogeneous, the method comprising the steps of:

   receiving a first data message from the first computer application;

   translating the first data message in a first message format that is compatible with the first computer application to a second data message having a second message format that is compatible with the second computer application while maintaining the substance conveyed by the first data message; and

   transmitting the second data message to the second application.

2. The method of claim 1 further comprising:

   modeling a translation transaction between the first message format and the second message format for use as a model in translating the first data message to the second data message.

3. The method of claim 2 wherein said modeling step comprises creating a map.

4. The method of claim 1 further comprising the steps of:

   listening for a specific message type; and

   accepting for translation those messages having a predetermined type.

5. The method of claim 1 wherein the translating step further comprises:
5 using stored metadata about the first and second data messages to perform the translation.

6. The method of claim 1 wherein the translating step further comprises:
10 using stored configuration information about data transactions to perform the translation.

7. The method of claim 1 where the translating step further comprises the steps of:
15 parsing the first data message to separate the substance of the first data message from the first message format; and,
building the substance of the first data message into the second message format to create the second data message.

8. An apparatus for exchanging data between a first computer application and a second computer application connected over a network, where the first computer application and the second computer application are heterogeneous, comprising:
25 a configuration utility storing data information and data transaction information; and
 a data interface for translating a first data message in a first message format from the first computer application into a second data message in a second message format for the second computer application, the data interface performing the translation according to the information stored in the configuration utility, the second message having the substance of the first message with the second message format compatible with the second computer application.
9. The apparatus of claim 8 wherein the data stored in the configuration utility further comprises:
   message type data, data transaction information, and at least one message translation model.

10. The apparatus of claim 8 wherein said data interface further comprises a listener listening for messages of at least one predetermined type.

11. The apparatus of claim 8 wherein said data interface has a message processor for fetching information from the configuration utility.

12. The apparatus of claim 8 wherein said data interface includes a unit dispatcher for parsing the first data message to separate the substance of the first data message from the first message format, and for building a second data message from the substance of the first data message into the second message format compatible with the second computer application.

13. The apparatus of claim 12 wherein the unit dispatcher comprises:
   a parser for separating the first data message into pieces;
   a converter for converting the parsed pieces to a format usable by the second computer application; and,
   an unparsers for putting the converted pieces together to form the second data message.

14. The apparatus of claim 13 further comprising a validator for confirming that each of the parsed pieces has the correct data based on the message type of the first data message.
15. The apparatus of claim 8 wherein the configuration utility further comprises:
configuration files containing information describing translation transactions, and,
a metadata repository containing information describing message types,
the configuration files and the metadata repository sending information to enable message translation in response to a request from the data interface.

16. The apparatus of claim 15 wherein said configuration utility further comprises a graphical user interface for controlling graphically the translation between the first data message and the second data message.

17. The application of claim 16 wherein the graphical user interface further comprises:
a mapping facility for mapping the translation transaction between the first data message and the second data message.

18. A method for exchanging data between a first computer system and a second computer system where the first computer system and the second computer system are heterogeneous, the method comprising the steps of:
listening on a network for messages having one of a plurality of predetermined types;
receiving at a data interface a first data message having a type of the plurality of predetermined types;
queuing the first data message for translation;
fetching stored configuration information for use in translating the first message, where the configuration information includes a translation model;
5 parsing the first data message to produce pieces which separate the substance of the first data message from the format of the first data message;

building the substance of the first data message into a second format compatible with the second computer system to create a second data message; and

transmitting the second data message.

19. The method of claim 18 further including the step of validating the parsed pieces of the first data message to confirm that each of the parsed pieces has the correct data based on the message type of the first data message.

20. The method of claim 18 further including a mapping facility whereby new translation transactions may be modeled.
FIG. 3

TCP/IP listener

Database listener

Mail listener

Input Queue

Message Processor

Output Queue

Dispatchers:
- TCP/IP dispatcher
- Database dispatcher
- Mail dispatcher

Data Interface

SUBSTITUTE SHEET (RULE 26)
Event Initiated

62

Parse message/ Error?

64

Validate/ Error?

66

YES

Generate call to database/ return data

70

YES

Action/ more data needed?

68

NO

Translation

72

Validate/ Error?

74

YES

Event/Action Is Event Process complete?

76

NO

Store data/ Initiate sub-event

78

Forward message(s) to receiving system(s)

80

Output queue

82

Dispatcher

84

Dispatch the output message(s)

86

FIG. 4

SUBSTITUTE SHEET (RULE 26)
FIG. 5

Input message sent from transmitting system

Data interface listener intercepts input message

Input message passed to input queue

Message processor processes input message and creates output message

Output message passed to output queue

Dispatcher reads output queue and transmits output message to receiving system

FIG. 6