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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
27 March 2003 (27.03.2003)

PCT

(10) International Publication Number
WO 03/025779 A1

(51) International Patent Classification⁷: G06F 15/16

(21) International Application Number: PCT/US02/29351

(22) International Filing Date:
17 September 2002 (17.09.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/323,592	19 September 2001 (19.09.2001)	US
60/323,606	19 September 2001 (19.09.2001)	US
60/323,608	19 September 2001 (19.09.2001)	US
60/323,642	19 September 2001 (19.09.2001)	US

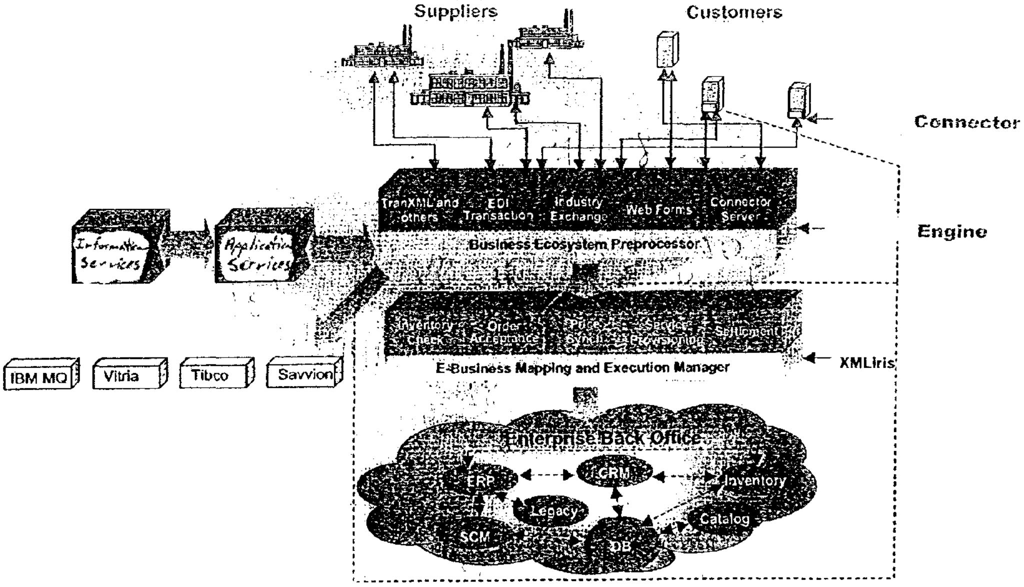
(71) Applicant *(for all designated States except US)*: **TRANSENTRIC LLC** [US/US]; 7930 Clayton Road, St. Louis, MO 63117-1368 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **DOMKE, Fred, M.** [US/US]; 602 Henry Oaks Court, Ballwin, MO 63011 (US). **BILBREY, Randall** [US/US]; 478 4th Street, Trenton, IL 62293 (US). **BOPPANA, Nageswara, R.** [IN/US]; 1233 Arbor Bluffs Circle, Ballwin, MO 63021 (US). **CAMPBELL, Thomas** [US/US]; 12100 Cedar Hills Drive, Highland, IL 62249 (US). **DESHPANDE, Rahul** [IN/US]; 12434j Lighthouse Way, Creve Couer, MO 63141 (US). **HEIMANN, John** [US/US]; 2273 Deer Springs Tr., Belleville, IL 62221 (US). **LEE, David** [US/US]; 15141 Isleview, Chesterfield, MO 63017 (US). **LIERMANN, Richard** [US/US]; 111 Keystone Drive, Fenton, MO 63026-4894 (US). **MILLER, Kyle** [US/US]; 507 North 13th. Street, Apt-701, St-Louis, MO 63103 (US). **MORIN, Randall** [US/US]; 9039 Harvest Run Court, O'Fallon, MO 63366 (US). **MUSENBROCK, Michael** [US/US]; 7509 Little Oaks Drive, O'Fallon, MO 63366-8218 (US).

(74) Agents: **BEULICK, John, S.** et al.; Armstrong Teasdale LLP, Suite 2600, One Metropolitan Square, St. Louis, MO 63102 (US).

[Continued on next page]

(54) Title: CONNECTIVITY SYSTEMS AND METHODS



(57) **Abstract:** Connectivity systems (figure 1) and methods for establishing connectivity between trading partners are described herein. In one example embodiment of the system, the system (figure 1) comprises a connector comprising a servlet runner for causing selected templates to be displayed to a first trading partner and a sender/receiver communicator for asynchronously communicating a message comprising information input by the first trading partner into at least one of the templates. The engine comprises a task list processor for processing the message communicated by the connector and a relational database for storing task instructions. The task list processor retrieves task instructions from the relational database based on a sender-receiver-transaction type triple associated with said message.

WO 03/025779 A1

(81) Designated States *(national)***:** AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)***:** ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

CONNECTIVITY SYSTEMS AND METHODS

CROSS REFERENCE TO RELATED APPLICATIONS

[0002] This application claims the benefit of U.S. Provisional Application Nos. 60/323,592, 60/323,606, 60/323,608, 60/323,642, all of which were filed on September 19, 2001, and are hereby incorporated by reference in their entirety.

BACKGROUND OF THE INVENTION

[0003] This invention relates generally to communicating messages between trading partners and, more particularly, to methods and systems which enable multiple trading partners to communicate using various combinations of formats and protocols.

[0004] Business-to-business electronic messaging is facilitated by connectivity between computers at each trading partner. Information exchanged by trading partners includes, for example, purchase orders, bills of lading, and financial settlements. Establishing true integrated and collaborative connectivity between trading partners, however, can be very complex. For example, mapping or transforming a message, or set of data, from the format used by a sending business to that used by a receiving business is highly complex and to be useful, must be performed with high reliability and speed.

[0005] In addition to the complexities associated with establishing messaging capability between just a few trading partners, in order to be cost effective, such infrastructure should be used across not just a few trading partners but by thousands of trading partners and be capable of handling millions of messages each day. Although the scalability of such messaging systems is important, as such systems are expanded to include hundreds of trading partners and to handle thousands of messages each day, complexity increases along with a possibility of reduced reliability.

BRIEF SUMMARY OF THE INVENTION

[0006] In one aspect, a connectivity system for communications between trading partners is provided. The system comprises a connector and an engine. The connector comprises a servlet runner for causing selected templates to be displayed to a first trading partner and a sender/receiver communicator for asynchronously communicating a message comprising information input by the first trading partner into at least one of the templates. The engine comprises a task list processor for processing the message communicated by the connector and a relational database for storing task instructions. The task list processor retrieves task instructions from the relational database based on a sender-receiver-transaction type triple associated with said message.

[0007] In another aspect, a method for operating a system to communicate between trading partners is provided. The system comprises a connector and an engine. The connector comprises a servelet runner and a sender/receiver communicator coupled to a database. The engine comprises a task list processor and a relational database having a plurality of tasks stored therein. The method comprises the steps of operating the servelet runner to cause a template to be displayed to a first trading partner, generate a file containing information input into the template by the first trading partner, and store the file in the connector database. The method further comprises the steps of operating the sender/receiver communicator to generate a message based on the information contained in the file, and asynchronously communicate, to the engine, the message. The method also comprises the steps of operating the engine task list processor to process the message, the processing comprising the step of retrieving, from the relational database, tasks to be executed based on a sender-receiver-transaction type triple associated with the message.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] Figures 1 – 5 illustrate example system architectures.

[0009] Figures 6 – 10 illustrate alternative system architectures.

[0010] Figures 11 – 13 illustrate example hardware architectures of a messaging engine.

[0011] Figures 14 – 16 illustrate messaging processes executed by the messaging engine.

[0012] Figures 17 – 27 illustrate workflow processes executed by the messaging engine.

[0013] Figures 28 – 37 illustrate message parse processes executed by the messaging engine.

[0014] Figures 38 – 40 illustrate example screen shots associated with the messaging engine.

[0015] Figure 41 illustrates an example hardware architecture for the connector.

[0016] Figure 42 illustrates an interactive application executed by the connector.

[0017] Figure 43 illustrates an acknowledgement process executed by the connector.

[0018] Figure 44 illustrates the envelope requirements of a message.

[0019] Figures 45 – 52 illustrate example screen shots associated with the connector.

[0020] Figures 53 – 54 illustrate an example hardware architecture of the integrator.

DETAILED DESCRIPTION OF THE INVENTION

[0021] The connectivity systems and methods, as described below in more detail, include a connector tool that allows trading partners of any size, volume or sophistication to participate in collaborative electronic commerce. Specifically, the connector tool facilitates creation of messages from common templates, and such messages are located in an outbound file in a directory. Of course, any message (not just messages created from a common template) that a trading partner desires to communicate can also be located in the directory. The outbound messages are then transmitted over a network (e.g., a wide area network such as the Internet) to a messaging engine.

[0022] The engine executes message capture, message management, and message forwarding tasks. Specifically, during message capture, an inbound message is received, identified, acknowledged, validated, and then either accepted or rejected. In message management, the message is translated, duplicates are eliminated, customer business rules are applied, and an audit trail is created. During message forwarding, messages are delivered to a recipient trading partner, routed using appropriate protocols and formats, and archived.

[0023] Set forth below are detailed descriptions of example embodiments of system architectures, messaging engines, connectors, and integrators. The present invention, however, is not limited to the specific embodiments described herein, and many variations and modifications are possible. In addition, components of each system and steps of each method described herein can be practiced independently and separately from other components and method steps described herein. The systems and methods described herein can be used in combination with other components and other steps.

[0024] The terms "engine", "connector" and "integrator" are sometimes used herein to refer to components of a connectivity system. Such terms refer to any processor, server or system capable of performing the functionality corresponding to such component, and such terms are not limited to any particular implementation. For example, the connector functionality can be implemented in a server, or in any processor capable of performing the functions described in connection with the connector. The specific functionality associated with each of the engine, connector, and integrator is set forth below.

[0025] Connector: comprises a servlet runner for causing selected templates to be displayed to a first trading partner and a sender/receiver communicator for asynchronously communicating a message comprising information input by the first trading partner into at least one of the templates.

[0026] Engine: comprises a task list processor for processing a message communicated thereto and a relational database for storing task instructions. The task list processor retrieves task instructions from the relational database based on a sender-receiver-transaction type triple associated with the message.

[0027] Integrator: comprises a processor for serving as a single source of information for connecting to the engine. The integrator is configured to couple to multiple connectors and other message sources. The integrator may be combined with enterprise applications adapters from third parties to provide tight coupling with internal applications.

I. System Architecture

[0028] Figure 1 illustrates an example architecture for a trading system. Generally, trading partners such as suppliers and customers communicate via a network that includes connectors and an engine. Generally, the connectors facilitate creation of messages from common templates. Messages can also have various other formats including web forms, EDI transactions, industry exchange specific formats, and other formats. The messages are supplied to the engine, which also is connected to other application services and information services.

[0029] The engine serves as a business ecosystem preprocessor that enables it to easily connect external trading partners and deliver immediate value. It connects with internal workflow management tools and leverages existing investments in electronic data interchange (EDI), extensible markup language (XML), and other connectivity methods. Generally, the engine executes message capture, message management, and message forwarding tasks as described above.

[0030] Transmission formats include Application-to-Application (JDE, SAP, etc.) EDI X12 (EDI X12 Standard), EDIFACT (EDI for Administration, Commerce, and Transport), EDIINT (EDI for Internet), Flat file, SMTP (Email), HTML (Web), HTTP (Web), and XML (Extensible Markup Language). Message protocols include dedicated line, SNA networking, frame relay, any VAN service (Interconnect), dial-up (Modem), FTP (File Transfer Protocol), SMTP (Email), and secure Internet connection

[0031] At the recipient trading partner, the received message can result in performance of an inventory check, order acceptance, pricing synchronization, service provisioning, settlement, and other functions. As a result of performing such functions, data is supplied to the enterprise back office functionality.

[0032] More specifically, and as illustrated in Figure 2, each trading partner can have various internal applications executed by servers and/or other processors coupled into a network. The network can be composed of loosely coupled processors, workflow servers, local area networks, wide area networks, and various other types of networks that enable communication of data. In the example shown in Figure 2, a connector is located in the DMZ and such server facilitates the creation of messages to be communicated to an application (engine) server via a wide area network (e.g., the Internet). As also shown in Figure 2, an integrator also can be utilized to communicate messages via the Internet to the engine.

[0033] Figure 3 illustrates, in more detail, a connector (e.g., connector host system) that communicates with various applications using various formats of a trading partner. Messages to and from the connector from the wide area network are communicated, in the example shown in Figure 3, using an HTTPS protocol.

[0034] Figure 4 illustrates an example connectivity between a trading partner and the engine. Specifically, a connector embodied as a web server is provided within a DMZ and is coupled to various applications. Communications need not, however, be transmitted through the connector and can be provided to a communication server via FTP. The connector facilitates creation of messages by executing connector servlets, and performs user validation via an LDAP server. More specifically, the connector includes a servletrunner for interactive processing and a messaging capability to ensure secure guaranteed message delivery. Together, these components provide distribution of applications functionality.

[0035] As shown in Figure 5, the engine can be coupled to various applications, a security database, a message store database, and personal computers (PCs) that perform functions such as a help desk, an administrator, and other internal users.

[0036] Of course, various alternative architectures also are contemplated and possible in addition to those described above in connection with Figures 1 – 5. For example, the connector can be located in a trading partner DMZ as illustrated in Figure 6. The connector can be located behind a trading partner firewall as illustrated in Figure 7. As shown in Figure 8, various trading partners can have the connector located in different security zones such as behind the firewall or in the DMZ. A connector behind a firewall can connect to the engine or to any other connector with a routable and accessible IP address. Either end may be accessible and messages may flow in both directions. The 'polling-pull' communications from the secure connector accommodates messages flowing into the secure connector. Also, and as shown in Figure 9, communications can be transmitted directly from one trading partner to another via the connectors, and the connectors can be located behind the firewall or in the DMZ. Two connectors, both behind firewalls, can communicate with each other using a connector relay. The relay provides an accessible node and the 'polling-pull' communications accommodates messages flowing into both secure connectors. As shown in Figure 10, the engine server also can be configured as a relay service and coupled to various trading partners with a variety of connectors. The connector relay is a feature of the engine that allows clients to send messages to each other through the relay, making it possible for both parties to operate from a secure node without a routable IP address. The connector relay will accept connections from both parties on a polling basis and when there is message traffic for the party, the party will 'pull' the message from the relay. This relay can be implemented in connection with a variety of protocols including HTTP(S) POST, FTP, and EDIINT (including connector clients).

II. Engine

[0037] Referring to Figures 11, 12 and 13A-B, the engine has a task list architecture in that the engine includes a task list processor to process messages. Tasks are the processing steps that enable the collaborative processes on the messaging community. These tasks include message storage, indexing, translation, transformation, transmission, validation, acknowledgement, rejection, and more complex processes based on 'Stateful Messaging'.

[0038] An inbound message is processed by the communications components and sent via JMS queue to an identify process that determines the sender, receiver and transaction type of the message, sometimes referred to herein as a sender-receiver-transaction triple. The engine then retrieves a task list of arbitrary length from a relational database and sends the message via JMS queue to the appropriate component to perform the first task. This process continues until all tasks have been accomplished.

[0039] Task lists are maintained at the sender-receiver-transaction ("SRT") level where any number of tasks may be defined for a unique SRT triple. This level of granularity provides flexibility. In order to minimize the effort of managing task lists for large, complex messaging communities, two additional features are provided based on object model inheritance. Specifically, "Pull Inheritance" enables the creation of a task list based on existing task lists in a hierarchy and "Push Inheritance" enables a task to be added to an entire hierarchy with one interaction.

[0040] The task list architecture employs chained transaction architecture to ensure that all steps are completed. The repository (RDBMS) provides persistence so that transactions always re-start or re-try from a known transaction state.

[0041] Components of the engine are multi-threaded. The use of enterprise JavaBeans and database connection pools enables many processes to share a pool of components. The use of JMS queues permits the engine to scale beyond multithreading in a near-linear fashion as shown in Figure 13B. Multiple instances of each task can process messages in parallel. These instances can be distributed across an arbitrary number of host processors to achieve very high throughput.

[0042] In the example embodiment, the engine conforms to the J2EE architecture and runs in mission-critical application servers with fault tolerant features provided by the container. JMS messages and database persistence also can be implemented using commercially available products to provide reliability and fault tolerance.

[0043] The engine stores message in a persistent repository (RDBMS) to provide an auditable record of message processing and transmission. These messages may be retrieved and viewed by messaging partners through an Internet Web interface. In addition, messages are retained for a configurable period to enable reporting and decision support systems to be constructed on community message content. Messages may be inspected in multiple formats (text, EDI-parsed format, XML) and may be modified and re-queued for transmission without re-sending from origin. This enables the correction of messages.

[0044] Message content is indexed using arbitrary message content so that it can be retrieved by user-specified criteria such as Purchase Order Number, Equipment Identifier, or other indexes. Reports can also be constructed that use these indexes.

[0045] The engine isolates all accessible nodes in a separate DMZ with clear layered protection zones in the system architecture. All processes that reside on nodes that can be accessed from outside the firewall immediately pass data through an internal firewall that only allows access from the DMZ. There is no persistent storage of message content in the DMZ. Proxy processes accept and process packets and forward content to the secure zone. This is an extensible architecture and proxies have been constructed for FTP, SMTP, EDIINT, and WebSphere MQ.

[0046] Example messaging flows are illustrated in Figures 14, 15, and 16. Generally, the task list processor processes an inbound message by executing message capture, message management, and message forwarding tasks. The message capture tasks comprise receipt of an inbound message, identification of an inbound message, validation of an inbound message, and acceptance/rejection of an inbound message. The message management tasks comprise translation of a message, eliminating duplicates of a message, applying customer business rules to a message, and creating an audit trail of a message. The message forwarding tasks comprise delivering a message to a recipient trading partner, and archiving a delivered message.

[0047] As shown in Figure 15, the engine also performs message management tasks. Such tasks include translate, eliminate duplicates, apply customer business rules, validate, and capture audit trail. Figure 16 illustrates the message forwarding task performed by the engine (e.g., deliver immediately, based on time, based on event). A message is delivered by the engine to the connector, for example, in a predefined format, and actual delivery can occur via various media (e.g., any VAN service, FTP, E-mail.

[0048] The workflow executed by the engine is illustrated in the flow diagrams shown in Figures 17 – 27. The message parse process executed by the engine is illustrated in the flow diagrams shown in Figures 28 – 37.

[0049] Figures 38, 39, and 40 illustrate example screen shots of messages as viewable at the engine. As shown in Figure 38, a user selects a viewer to use to view a message. Figure 39 illustrates an example bill of lading as viewed using a selected viewer. Figure 40 illustrates a message in a format different from the message (e.g., bill of lading) illustrated in Figure 39,

III. Connector

[0050] Figure 41 is a block diagram of an example connector and Figure 42 illustrates an example interactive program. As explained above, the connector facilitates creation of messages by executing connector servlets, and performs user validation via an LDAP server. Connector, in the example embodiment, is a standards-driven, cross-platform application that enables a trading partner to create and manage messages for secure, reliable transmission and receipt via the engine. The connector supports an expansive variety of data formats and data entry paradigms, including files, browser-based data entry, and connectivity with major packaged applications. The example embodiment described below is described in the context of TranXML, which is an example format. Of course, many other formats can be utilized. TranXML is an extensible markup language defined by the assignee of the present application, and details regarding TranXML are publicly available, including from Transentric LLC, 7930 Clayton Road, St. Louis, MO 63117.

[0051] The connector provides/performs the following functions.

1. Accepts files as input. The conversion of the data is done by the engine before delivering to the trading partner.
2. Has the ability to create templates. Provides for customization of data entry, reduces amount of data to be input, speeds up data entry, and lowers chance for errors.
3. Provides reliable message service with guaranteed once only delivery.
4. Converts the data to a TranXML message, which enables the trading partner to display it using a Web browser.

5. Contains a scoreboarding type match-up process and life-cycle management. This feature enables the user to verify that his messages have been received and processed. This process matches up the original message with the acknowledgement and keeps the unacknowledged or acknowledged with errors message available for correction and resubmission. Successfully acknowledged messages are deleted from the directory.
6. Utilizes Web forms to display what was sent, what has been acknowledged and processed, and what has been marked with errors.
7. Enables non-EDI and non-XML capable users to participate in electronic commerce without investing any money in software. Enables the small to medium company that has never done EDI to transact with trading partners electronically. Connector makes it easier for a company to expand it's fleet of trading partners.
8. Exchange documents automatically on a 24 x 7 basis without human intervention.
9. Enables the exchange of a variety of data formats, e.g., TRANXML, EDI
10. Data mapping from any-to-any format.
11. Connect back-office systems from one company to another.
12. Accelerates deployment and time-to-market.

[0052] The connector includes a servletrunner for interactive processing and a messaging capability to ensure secure guaranteed message delivery. File-based persistence between the servletrunner and the messaging engine enables asynchronous creation and delivery of messages. This makes it possible for users to interact with the connector on their own system and / or their own LAN and the connector will provide delivery when connectivity is established. Users can operate the connector with intermittent Internet connectivity including standalone systems and wireless systems. Connector relies on the underlying file system for persistence, logging, and storage of local configuration information. If workflow functionality is implemented in XMLiris (commercially available from FiveSight Technologies Inc., 213 N. Morgan, Suite 1A, Chicago, Illinois 60607), a process cascade provides connectivity between components (replacing JMS queues in a full implementation).

[0053] The connector functionality, in one example embodiment, is implemented in Java code, which enables platform independence. In the interest of supporting many users, connector should be operable to run on any operating system that supports a Java2 Virtual Machine. Commercially available and well known operating systems and versions currently supporting a Java2 Virtual Machine include

Microsoft Windows 95/98/NT/2000, Linux 2.x, Sun Solaris, IBM OS/2, IBM OS/400 (V4R5 or later), IBM AIX (4.3.3.10 or later), and HP-UX.

[0054] Connector provides import functionality for flat files as well as browser-based data entry and import functionality for TranXML. Also, connector enables the engine to identify and authenticate message senders and to send ("push") updates to business logic or transformation scripts. Connector also enables users to configure and maintain basic aspects of the local system through a browser-based interface.

[0055] Regarding security, and in one example embodiment, connector conforms to security standards set by the well known Electronic Data Interchange-Internet Integration work group (EDIINT) AS2 of the Internet Task Force. Connector uses HTTP and S/MIME standards to provide secure and reliable communications links. Connector uses the following tool sets to ensure secure exchanges of data: EDIINT AS2, S/MIME encryption, X.509 type certificates to authenticate the client (certificates are validated at both ends of the communication link), public and private keys for encryption, MD5 integrity checking to verify that the message has not been altered, and a user ID combined with the organization name that must be registered in the system. Other tools appropriate to security (e.g., secure socket layer (SSL)) and connectivity (e.g., JMS) can be utilized.

[0056] Both components of the connector are multi-threaded to provide scalability. An enterprise can install a single connector and share its servletrunner and its file system across a LAN, or multiple connectors may be installed, including configurations where each user installs their own connector. Response time is minimized and performance is maximized because it is independent of Internet access speed, communications throughput, and contention from users on other connectors.

[0057] Messages can be created or sent to the connector in a variety of ways including interactive creation using the servletrunner engine, HTTP file upload, JMS message queues, relational databases, and file-based upload. The local file system can accommodate any type of message creation and security model the host system provides. The connector uses a directory scanner to detect that new messages are present and ready to be processed.

[0058] The connector uses a staging directory to preserve file integrity. When the file is complete, it is moved to an outbox directory on the same file system using a rename to ensure that the entire file is present before processing begins. Use of a local servletrunner enables local access-control administration by each trading partner. The local administrator uses the access control features of the servletrunner (Tomcat) to allow or deny access to servlets (JSPs, Agilink Controllers, XML Roundtrip, and Java Servlets). These servlets create content that is forwarded by the connector to messaging partners.

[0059] Polling-push communications make it possible to deploy the connector from behind a firewall, without a routable IP address. A secure configuration of the connector behind the firewall enables Internet messaging without any inbound connections.

[0060] For communications, connector forms a secure transmission loop using the S/MIME protocol. When a message is sent, the following steps are performed.

> Connector signs and encrypts the data using S/MIME and requests that a signed receipt be returned.
>
> The engine decrypts the message using the public and private keys and verifies the signature and authenticate the sender.
>
> The engine then returns a signed receipt to connector in the form of an MDN. This signed receipt indicates that the message was received and decrypted correctly, that authentication of the sender occurred, and that the integrity of the interchange was validated.

[0061] The use cases fall into two categories: data submission, and system monitoring/management. The use-case descriptions below make some architectural assumptions in the interest of clarity, as shown in Figure 41. The major assumption is that the system provides for directory-based "buckets" from which files are either picked up, or to which files are deposited. It is assumed that at least three such buckets will exist:

[0062] The OUTGOING bucket will contain documents for upload to the engine. The user will be able to initiate an upload to the engine or set the application to automatically upload these documents.

[0063] The ERROR bucket will contain documents that failed to process for one reason or another, e.g., an incomplete TranXML document. The user will be able to view, modify, and resubmit a document in the ERROR bucket. Both the local connector processes and the engine may place documents in the ERROR bucket. In order to simplify resubmission, each document in the ERROR bucket will contain sufficient information to recreate the original message that caused the error.

[0064] The RECEIPTS bucket will contain any messages returned from the engine (excluding messages destined for the ERROR bucket), including acknowledgements of message successfully received and application acknowledgements generated by receiving applications.

[0065] Connector accommodates two primary methods of data submission: file-based and interactive. Both methods rely on directory-based buckets from which files are either picked up or deposited. The interactive submission method will provide a browser-based interface to these buckets, allowing a user to quickly create documents for submission.

[0066] The most fundamental use case is submission of documents by the client to the engine by placing TranXML files in a directory. The actor in this case can be either a person, a computer process, or another connector component. The activity in this case is summarized below:

1. TranXML documents are placed in the directory corresponding to the "OUTGOING" bucket.
2. The user or a timed process triggers the creations of a connection to the engine if one is not established.

3. Documents in the "OUTGOING" bucket are read from disk and checked locally against some basic sanity rules. Documents that fail the sanity test get routed to the "ERROR" bucket.
4. Documents that passed the sanity check are transmitted to the engine.
5. The engine applies logic to the incoming documents.
6. The engine generates replies (confirmations or rejections) for the documents received.
7. The engine transmits replies to the connector.
8. The connector routes the message confirmations and rejections from the engine to the "RECEIPTS" and "ERROR" buckets respectively.

[0067] In order to facilitate data capture from unsophisticated data collection points the system will provide for basic interactive data entry. This use case is a mere extension of file-based submission case as described in the previous section. Here the actor is a person interacting with an interactive browser-based application. The application forwards user input to the appropriate bucket. The interactive application component is illustrated in Figure 42. A typical interactive submission would proceed as follows:

1. User logs into browser-based application.
2. User selects data entry from the menu.
3. User selects a template to work on.
4. User fills-in template.
5. User submits form.
6. Application converts form submission into TranXML document.
7. Application writes TranXML document to the directory corresponding to the "OUTGOING" bucket.

[0068] Both templates and data entry can be implemented as HTML forms bound to an underlying XML document through XSLT. A template is then implemented as the HTML form image of a partially populated document.

[0069] For file-based submission (non-tranXML), the goal is to transmit a document of non-TranXML format to the engine for processing. The actor in this case may be either a person or another computer process. The activity is very similar to the file-based submission of TranXML documents, except that no local validity checking is performed on the submitted documents:

1. Non-TranXML documents are placed in the directory corresponding to the "OUTGOING" bucket.

2. The user or a timed process triggers the creations of a connection to the engine if one is not established.
3. Documents in the "OUTGOING" bucket are read from disk and transmitted to the engine.
4. The engine applies logic to the incoming documents.
5. The engine generates replies (confirmations or rejections) for the documents received.
6. The engine transmits replies to the connector.
7. The connector routes the message confirmations and rejections from the engine to the "RECEIPTS" and "ERROR" buckets respectively.

[0070] Registration is the first action to be done by a new user using the connector. Set forth below are the steps executed in connection with registration.

1. The user receives the connector software through a download or in a shrink-wrapped form.
2. The user runs the INSTALL process, which loads the software and creates the necessary directories (buckets).
3. The user then configures the system.
4. After the software is loaded and the system configured, the user logs into the system and performs the initial setup process. User provides necessary identification information such as E-mail address, what transactions he will be performing, what trading partners that he will be trading with, and what templates he will be using.
5. Above setup information is then automatically sent to the engine. Profiles are then established in the engine and a certificate is established. After proper verification of the identity of the new user is verified, the certificate is conveyed to the new user via E-mail.

[0071] The "configure system" process is executed only once by the new user. The process is described below.

File #1 - on connector client machine:
props.conf => located in the connector (client side) install directory, this file must be edited to have all file directories point to their correct location. These properties includes:
- 1.sub.read_dir (directory to poll for messages)
- 1.sub.error_dir (failed reads will be posted here)
- 1.task.as2_keystore
- 1.task.as2_receipt_dir (mdn responses are writtern here)
- 1.task.as2_error_dir (failed processing send mesages here)
- 1.task.as2_copy_dir (successful sent messages are copied here)

Other properties are:
- 1.task.as2_host (the server host to connect to)

- 1.task.as2_port (server port, by default 8443)

File #2 - on connector client machine

$TOMCAT_HOME/webapps/aglclient/WEB-INF/web.xml.
Again, the necessary parameters to edit are all file path parameters.

File #3 - xslt file for roundtrip - on connector client machine

In this case, warehouse_payload.xslt file is used. The field to edit is the <xsl:variable name="basepath" node, and the path should be set to the directory containing all the codes necessary for the xslt script (usually, the same directory containing the xslt file).

File #4 - on connector server machine

$TOMCAT_HOME/webapps/agl/WEB-INF/web.xml
Parameters to set here are the url to the machine housing weblogic and the name of the bean to receive incoming message, passing these to a queue.

[0072] For creating a TranXML document, the following use case is performed every time the user fills in information on a template and submits the data for processing.

1. User selects template and enters appropriate data.
2. User submits the document.
3. XiLA process reads template and builds the TranXML document.
4. TranXML envelope is placed around the document and it is placed into the "OUTGOING" bucket.

[0073] The create TranXML template process may be executed periodically by the user depending upon usage of new documents, business practices, and other factors.

1. User indicates that he wants to create a new template.
2. User selects skeleton template.
3. User keys in appropriate data using the skeleton template as a guide. Data to be captured should be repeatable information that would be present for each transaction. Variable data such as item information, weights, would be keyed in for each shipment.
4. When user completes the data entry, a new customized template is created and the user provides an appropriate name. This template will then be added to the selection list for future data entry of the variable data.
5. Maintenance for these customized templates will be the responsibility of the user.

[0074] To check a transaction status, acknowledgements for messages sent to the engine are received by the connector and placed in the "RECEIPTS" bucket. Figure 43 illustrates an example acknowledgement process initiated at the connector. Generally, a message is created and transmitted to the engine. The engine communicates the acceptance status of the message to the connector and the connector stores the message status locally.

[0075] Several types of acknowledgements, syntax and application, could be received by the connector. Syntax acknowledgements are generated by the engine when the message is received. These acknowledgements indicate if the format of the transaction conforms to standards. Only accepted acknowledgements will be received by the connector. Acknowledgements indicating errors will be handled by the engine. The syntax errors will be corrected and the transaction submitted for processing. The successful acknowledgement will then be sent to the connector indicating that the transaction has been accepted for processing. Application acknowledgements may also be sent to the connector. These acknowledgements indicate whether the content of the document is valid and are generated by an application system processing the data. Application acknowledgements may be for both accepted and rejected transactions.

1. Using a Web form, the user indicates that he wants to check the status of transactions that have been sent to the engine. Summary transaction data for transactions that have been sent to engine is displayed along with the matching acknowledgement information. The display will indicate if the message was accepted and if it has been processed OK or if it had errors.
2. If it was processed OK, the user can then delete the transaction from the "OUTGOING" bucket.
3. If it had application errors, the original outgoing transaction will be placed in the "ERROR" bucket. The user has the option of retrieving the original transaction from the "ERROR" bucket, correcting the errors, and resubmitting.
4. The user can work his way through all of the acknowledgements taking the appropriate action based on the type of acknowledgement or quit and resume at a later time. The acknowledgements and matching transactions will remain until cleaned up by the user.

[0076] If while checking the status of transactions, the user determines, based on the information in the acknowledgement, that one has an error, the message can be resubmitted. This would be a data error because all transaction formatting errors are handled by the engine.

1. User is checking the transaction status and has discovered an acknowledgement indicating an error.
2. The user then displays the errored transaction from the "ERROR" bucket and determines the reason why the transaction was errored.
3. The user then corrects the error in the original transaction and submits it for reprocessing.
4. The transaction is placed on the "OUTGOING" bucket ready for the next transmission to the host.

[0077] Figure 44 illustrates example envelope requirements for a TranXML format.

[0078] For AS2 MIME Templates, the structure of an AS2 MIME message - PGP/MIME, is set forth below.

```
No encryption, no signature
-RFC2068/2045
     -RFC1767/RFC2376 (application/EDIxxxx or
application/xml)

No encryption, signature
   -RFC2068/2045
     -RFC1847 (multipart/signed)
       -RFC1767/RFC2376 (application/EDIxxxx or
application/xml)
       -RFC2015 (application/pgp-signature)


Encryption, no signature
   -RFC2068/2045
     -RFC1847 (multipart/encrypted)
       -RFC2015 (application/pgp-encrypted)
         -"Version: 1"
       -RFC2015 (application/octet-stream)
         -RFC1767/RFC2376 (application/EDIxxxx or
application/xml) (encrypted)

Encryption, signature
   -RFC2068/2045
     -RFC1847 (multipart/encrypted)
       -RFC2015 (application/pgp-encrypted)
         -"Version: 1"
       -RFC2015 (application/octet-stream)
```

```
         -RFC1847 (multipart/signed)(encrypted)
           -RFC1767/RFC2376 (application/EDIxxxx or
application/xml)(encrypted)
           -RFC2015 (application/pgp-
signature)(encrypted)


No encryption, no signature
   -RFC2068/2045
     -RFC1767/RFC2376 (application/EDIxxxx or
application/xml)

No encryption, signature
   -RFC2068/2045
     -RFC1847 (multipart/signed)
       -RFC1767/RFC2376 (application/EDIxxxx or
application/xml)
       -RFC2633 (application/pkcs7-signature)

Encryption, no signature
   -RFC2068/2045
     -RFC2633 (application/pkcs7-mime)
       -RFC1767/RFC2376 (application/EDIxxxx or
application/xml) (encrypted)

Encryption, signature
   -RFC2068/2045
     -RFC2633 (application/pkcs7-mime)
       -RFC1847 (multipart/signed) (encrypted)
         -RFC1767/RFC2376 (application/EDIxxxx or
application/xml) (encrypted)
         -RFC2633 (application/pkcs7-signature)
(encrypted)
```

[0079] Skeletal Requirements for an EDIINT (sample format) are set forth below.

```
   To:  <<recipient organization >>
   Subject:
   From: <<sending organization >>
   Date:
   Mime-Version: 1.0
   Content-Type: Application/XML
   Content-Transfer-Encoding: QUOTED-PRINTABLE
```

<<standard TRANXML Interchange goes here>>

[0080] Figures 45 – 51 illustrate example screen shots displayed by one example embodiment of connector. Generally, and as shown in Figure 45, a user can select administration, viewing, and create/update functions within connector. Figures 46 – 47 illustrate viewing transaction status. Figures 48, 49, and 50 illustrate creating/updating a transaction. Figure 51 illustrates creating/updating templates that are displayed by connector and utilized in generating messages. Figure 52 illustrates a shipment status message.

IV. Integrator

[0081] Figures 53 and 54 illustrate an example embodiment of the integrator. The integrator provides functionality for trading partners to connect reliably and securely a single source of information to the engine. The integrator facilitates implementation of electronic connectivity of business partners, and extends the scope of responsibility from LAN to LAN at trading partners. The integrator may combine with enterprise applications adapters from third parties to provide tight coupling with internal applications.

[0082] In one example, integrator is implemented as a Java2 executable suitable for use on many different platforms with minimal modification. Typically, integrator supports a single-digit number of users per client installation. Integrator can be configured to directly support JMS-compatible message-oriented middleware such as IBM MQ Series.

[0083] Integrator can be implemented using open source third-party tools such as the Apache Jakarta tools for servlets/JSP, regular expressions, and other functions. Apache Xerces and Xalan can be used for for XSLT-based other to XML conversion. FiveSight RoundTrip can be used to populate and edit TranXML documents, and FiveSight XiLA can be used to validate TranXML documents. These tools are commercially available and well know.

[0084] Integrator transmits messages to the engine (e.g., by HTTPS POST), receives acknowledgements (e.g., MD5 in the response), verifies the acknowledgement, and receives an acknowledgement of receipt of the verification. Integrator can also send messages on a JMS queue connected to a hub outside of the firewall at the integrator server. A validation agent within the bridge will ensure that the message meets all requirements (completeness, destination, etc.)

[0085] While the invention has been described in terms of various specific embodiments, those skilled in the art will recognize that the invention can be practiced with modification within the spirit and scope of the claims.

WHAT IS CLAIMED IS:

1. A connectivity system for communications between trading partners, said system comprising:

a connector comprising a servlet runner for causing selected templates to be displayed to a first trading partner and a sender/receiver communicator for asynchronously communicating a message comprising information input by the first trading partner into at least one of said templates; and

an engine comprising a task list processor for processing said message communicated by said connector and a relational database for storing task instructions, said task list processor retrieving task instructions from said relational database based on a sender-receiver-transaction type triple associated with said message.

2. A system in accordance with Claim 1 further comprising a database, messages to be sent by said sender/receiver communicator being at least temporarily stored in said database.

3. A system in accordance with Claim 1 wherein said connector is electronically positioned within a DMZ of the first trading partner.

4. A system in accordance with Claim 1 wherein said connector is electronically positioned behind a firewall of the first trading partner.

5. A system in accordance with Claim 1 wherein said connector operates asynchronously with respect to creation and delivery of messages to said engine.

6. A system in accordance with Claim 1 wherein said connector is implemented in Java code.

7. A system in accordance with Claim 1 wherein the first trading partner has a plurality of connectors, each of said connectors configured to communicate with said engine.

8. A system in accordance with Claim 1 wherein messages are communicated to said engine via sources in addition to said connector.

9. A system in accordance with Claim 1 wherein access to said servlet runner is controlled by a local administrator.

10. A system in accordance with Claim 1 wherein said connector communicates with said engine in polling-push and in polling-pull operations.

11. A system in accordance with Claim 1 wherein said engine task list processor executes at least one of message storage, indexing, translation, transformation, transmission, validation, acknowledgement, and rejection tasks.

12. A system in accordance with Claim 1 wherein for an inbound message received by said engine, said processor determines the sender, receiver and transaction type of the message, retrieves a task list of arbitrary length from a relational database and sends the message via a queue to an appropriate component to perform the first task.

13. A system in accordance with Claim 12 wherein said task list is created based on at least one of a "Pull Inheritance" and a "Push Inheritance".

14. A system in accordance with Claim 1 further comprising a persistent repository, said engine coupled to said repository for storing messages therein to provide an auditable record of message processing and transmission.

15. A method for operating a system to communicate between trading partners, said system comprising a connector and an engine, said connector comprising a servelet runner and a sender/receiver communicator coupled to a database, said engine comprising a task list processor and a relational database having a plurality of tasks stored therein, said method comprising the steps of:

operating the servelet runner to:

cause a template to be displayed to a first trading partner,

generate a file containing information input into the template by the first trading partner, and

store the file in the connector database;

operating the sender/receiver communicator to:

generate a message based on the information contained in the file, and

asynchronously communicate, to the engine, the message; and

operating the engine task list processor to process the message, the processing comprising the step of retrieving, from the relational database, tasks to be executed based on a sender-receiver-transaction type triple associated with the message.

16. A method in accordance with Claim 15 wherein the connector operates asynchronously with respect to creation and delivery of messages to the engine.

17. A method in accordance with Claim 15 wherein access to the servlet runner is controlled by a local administrator.

18. A method in accordance with Claim 15 wherein the connector communicates with the engine in polling-push and in polling-pull operations.

19. A method in accordance with Claim 15 wherein the engine task list processor executes at least one of message storage, indexing, translation, transformation, transmission, validation, acknowledgement, and rejection tasks.

20. A method in accordance with Claim 15 wherein for an inbound message received by the engine, the processor determines the sender, receiver and transaction type of the message, retrieves a task list of arbitrary length from a relational database and sends the message via a queue to an appropriate component to perform the first task.

21. A method in accordance with Claim 20 wherein said task list is created based on at least one of a "Pull Inheritance" and a "Push Inheritance".

22. A connector for communicating messages, said connector comprising:

a servelet runner for causing selected templates to be displayed to a first trading partner and for generating an outbound file containing a message; and

a sender/receiver communicator for asynchronously communicating the message to a recipient.

23. A connector in accordance with Claim 22 further comprising a database, messages to be sent by said sender/receiver communicator being at least temporarily stored in said database.

24. A connector in accordance with Claim 22 wherein said connector operates asynchronously with respect to creation and delivery of messages.

25. A connector in accordance with Claim 22 wherein said connector is implemented in Java code.

26. A connector in accordance with Claim 22 wherein access to said servlet runner is controlled by a local administrator.

27. A connector in accordance with Claim 22 wherein said connector communicates messages in polling-push and in polling-pull operations.

28. A messaging engine comprising:

a task list processor for processing an inbound message, said processor operable to execute at least one of message capture, message management, and message forwarding tasks; and

a relational database for storing task instructions, said task list processor retrieving task instructions from said relational database based on a sender-receiver-transaction type triple associated with the inbound message.

29. A messaging engine according to Claim 28 wherein said message capture tasks comprise receipt of an inbound message, identification of an inbound message, validation of an inbound message, and acceptance/rejection of an inbound message.

30. A messaging engine according to Claim 28 wherein said message management tasks comprise translation of a message, eliminating duplicates of a message, applying customer business rules to a message, and creating an audit trail of a message.

31. A messaging engine according to Claim 28 wherein said message forwarding tasks comprise delivering a message to a recipient trading partner, and archiving a delivered message.

32. A messaging engine in accordance with Claim 28 wherein said engine task list processor executes at least one of message storage, indexing, translation, transformation, transmission, validation, acknowledgement, and rejection tasks.

33. A messaging engine in accordance with Claim 28 wherein for an inbound message received by said engine, said processor determines the sender, receiver and transaction type of the message, retrieves a task list of arbitrary length from a relational database and sends the message via a queue to an appropriate component to perform the first task.

34. A messaging engine in accordance with Claim 33 wherein said task list is created based on at least one of a "Pull Inheritance" and a "Push Inheritance".

35. A messaging engine in accordance with Claim 28 comprising JMS queues wherein multiple instances of each task can process messages in parallel, said instances distributed across a plurality of host processors.

36. A connectivity system for communications between trading partners, said system comprising:

a first connector comprising a servlet runner for causing selected templates to be displayed to a first trading partner and a sender/receiver communicator for asynchronously communicating a message comprising information input by the first trading partner into at least one of said templates; and

a second connector comprising a servlet runner for causing selected templates to be displayed to a second trading partner and a sender/receiver communicator for asynchronously communicating a message comprising information input by the second trading partner into at least one of said templates.

37. A connectivity system in accordance with Claim 36 further comprsing an engine for receiving communications from said first and second connectors, said engine comprising a task list processor for processing a message communicated by one of said connectors and a relational database for storing task instructions, said task list processor retrieving task instructions from said relational database based on a sender-receiver-transaction type triple associated with said message.

38. A system in accordance with Claim 37 wherein at least one of said connectors is electronically positioned within a DMZ of the respective trading partner.

39. A system in accordance with Claim 37 wherein at least one of said connectors is electronically positioned behind a firewall of the respective trading partner.

40. An integrator comprising a processor and configured to couple to at least one connector, said integrator further configured to function as a single source of information for communicating messages to an engine.

41. An integrator in accordance with Claim 40 wherein said integrator is implemented in Java code.

42. An integrator in accordance with Claim 40 wherein said integrator is configured to transmit messages to an engine, receive acknowledgements from the engine, verify the acknowledgement, and receive an acknowledgement of receipt of the verification.

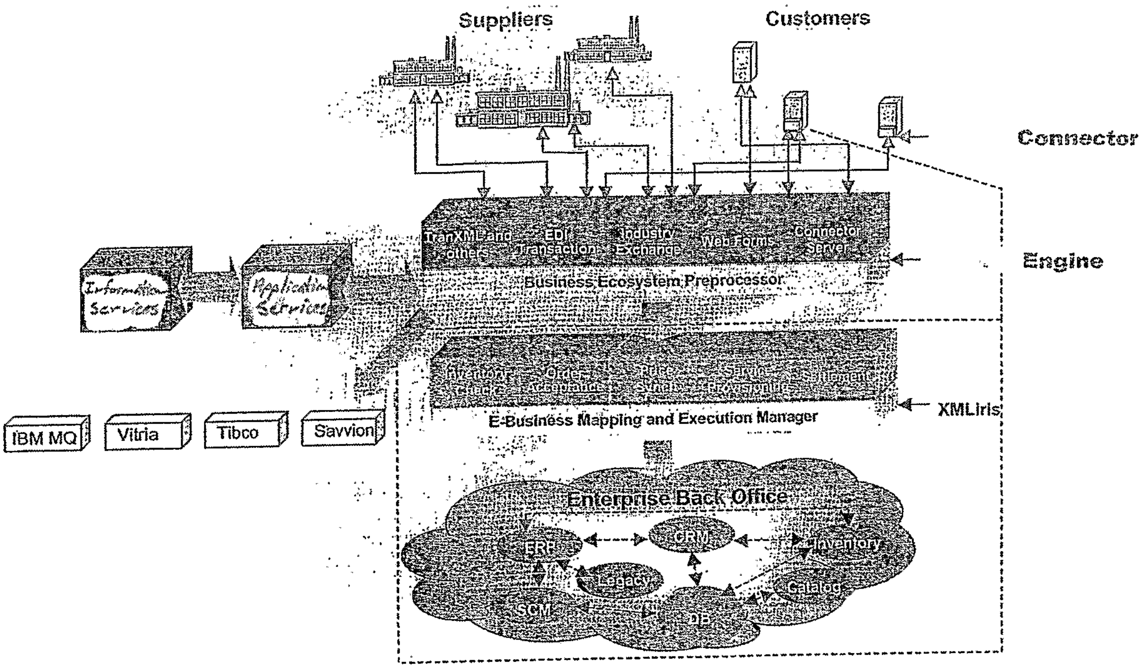

Suppliers
Customers
Connector
TranXML and others
EDI Transaction
Industry Exchange
Web Forms
Connector Server
Engine
Business Ecosystem Preprocessor
Information Services
Application Services
Inventory Check
Order Acceptance
Price Synch
Service Provisioning
Settlement
XMLiris
E-Business Mapping and Execution Manager
IBM MQ
Vitria
Tibco
Savvion
Enterprise Back Office
ERP
CRM
Inventory
Legacy
Catalog
SCM
DB

Fig. 1

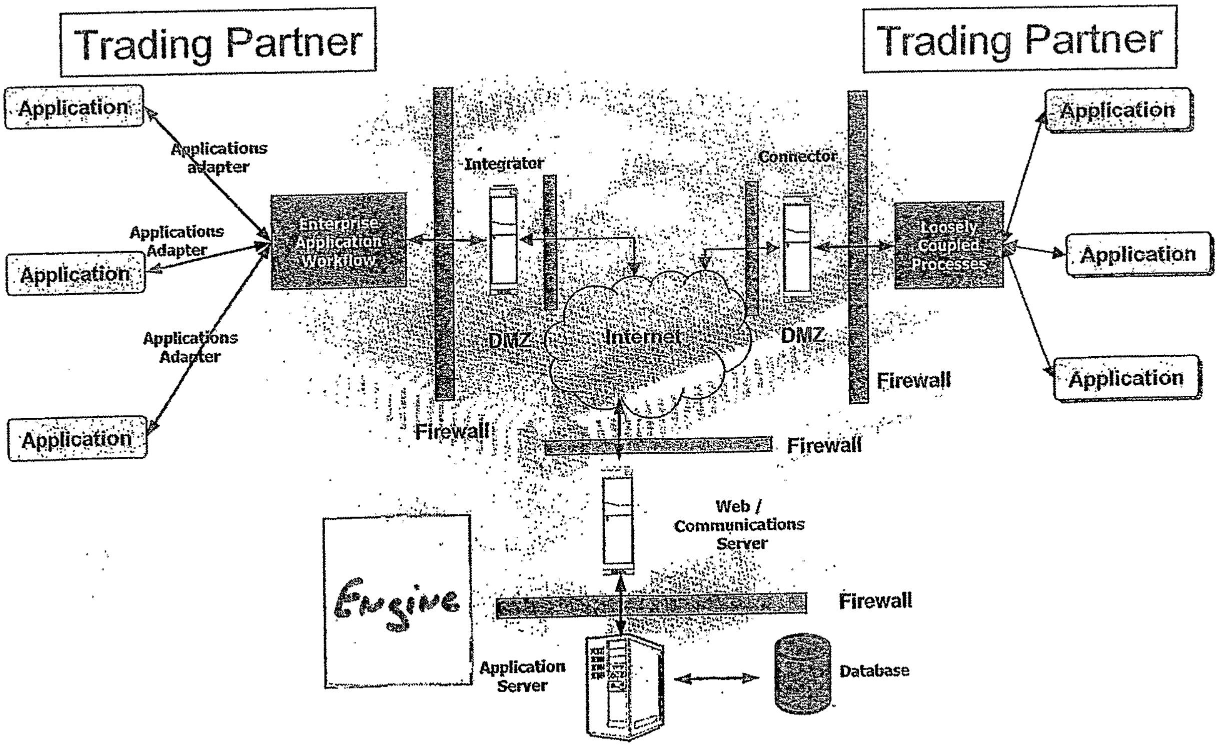

Trading Partner
Trading Partner
Application
Applications adapter
Applications Adapter
Application
Applications Adapter
Application
Enterprise Application Workflow
Integrator
Connector
DMZ
Internet
DMZ
Firewall
Firewall
Loosely Coupled Processes
Application
Application
Application
Firewall
Web / Communications Server
Engine
Firewall
Application Server
Database

Fig. 2

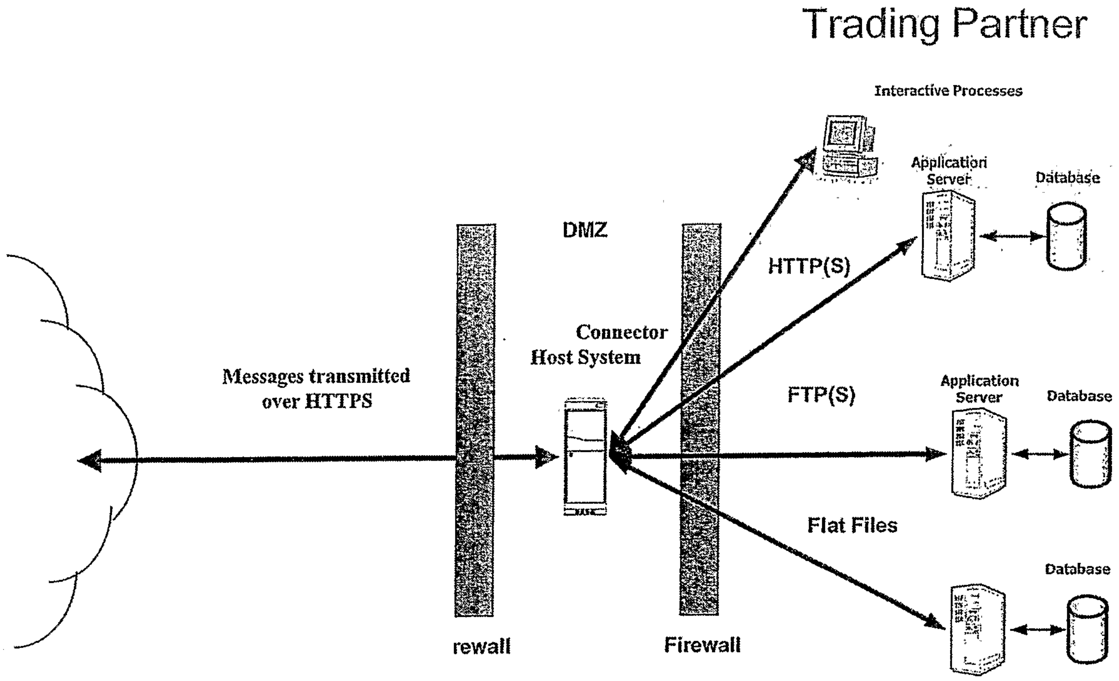

Trading Partner
Interactive Processes
Application Server
Database
DMZ
HTTP(S)
Connector
Host System
Messages transmitted
over HTTPS
FTP(S)
Application Server
Database
Flat Files
Database
rewall
Firewall

Fig. 3

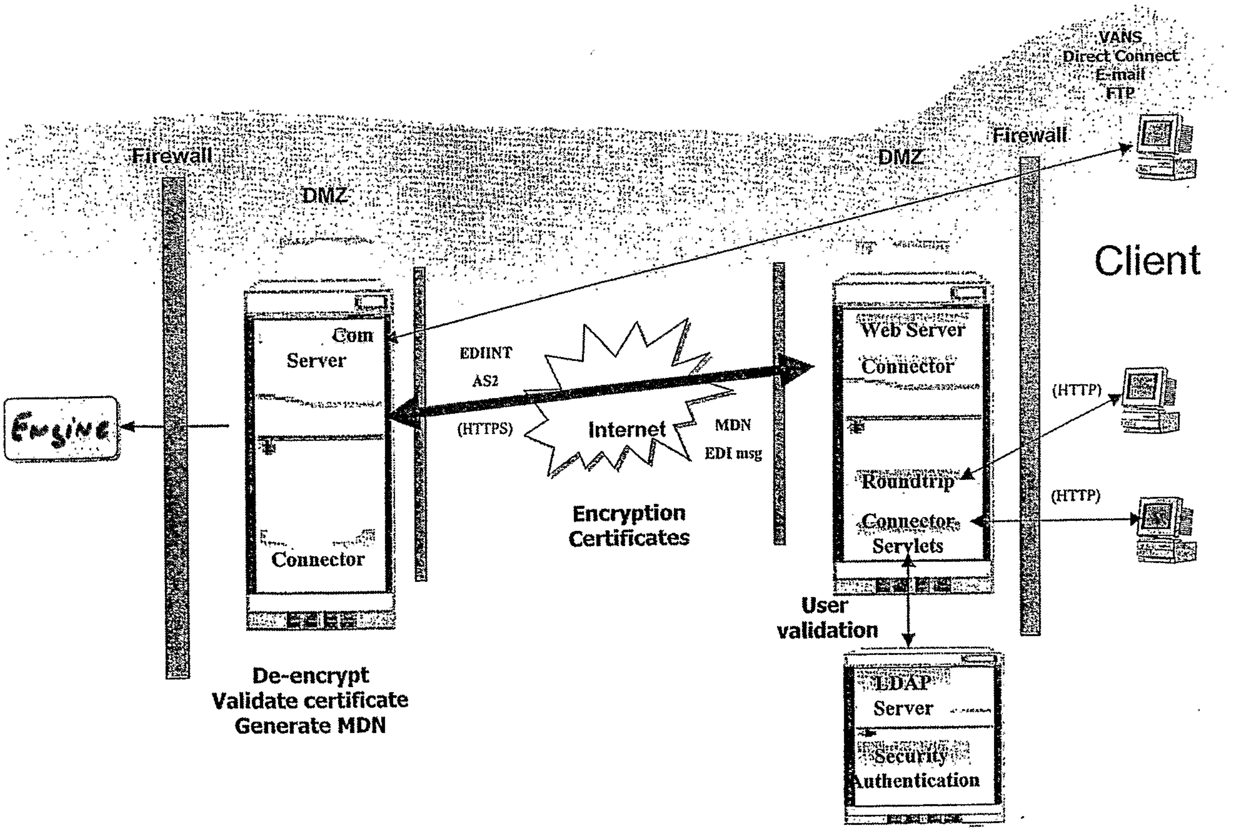

VANS
Direct Connect
E-mail
FTP
Firewall
DMZ
Firewall
DMZ
Client
Com
Server
Connector
Engine
EDIINT
AS2
(HTTPS)
Internet
MDN
EDI msg
Encryption
Certificates
Web Server
Connector
Roundtrip
Connector
Servlets
(HTTP)
(HTTP)
User
validation
LDAP
Server
Security
Authentication
De-encrypt
Validate certificate
Generate MDN

Fig. 4

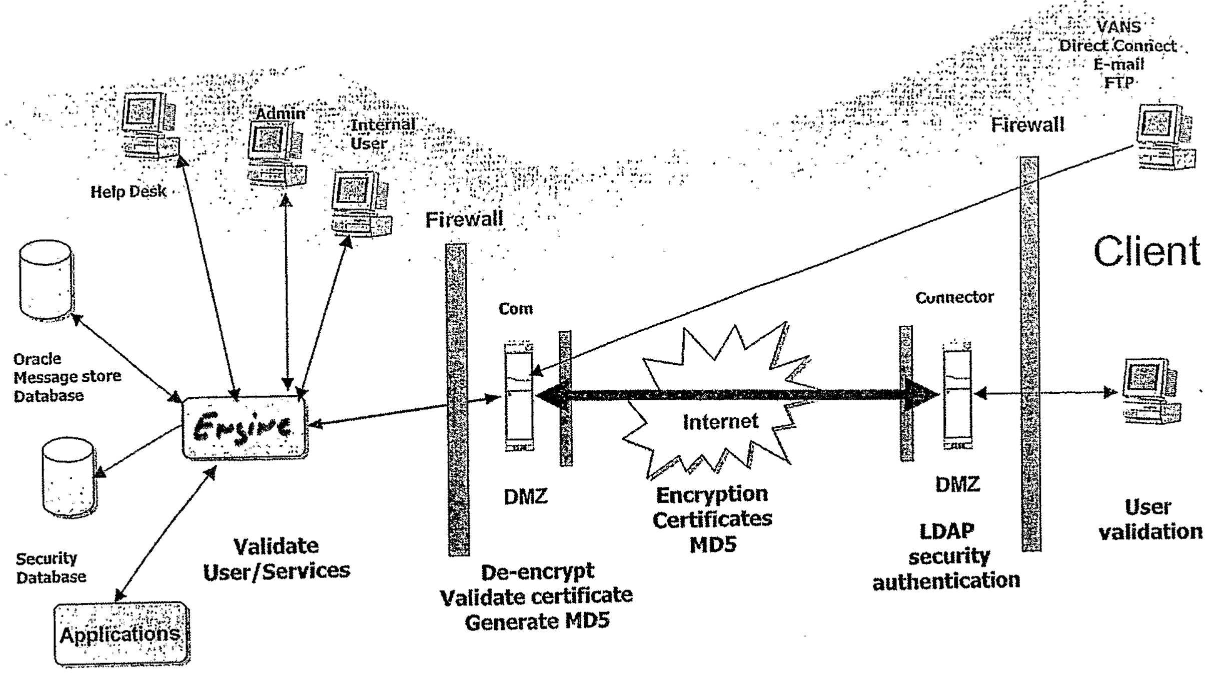

VANS
Direct Connect
E-mail
FTP
Help Desk
Admin
Internal User
Firewall
Firewall
Client
Oracle Message store Database
Com
Connector
Engine
Internet
Security Database
DMZ
DMZ
Encryption Certificates MD5
LDAP security authentication
User validation
Validate User/Services
De-encrypt Validate certificate Generate MD5
Applications

Fig. 5

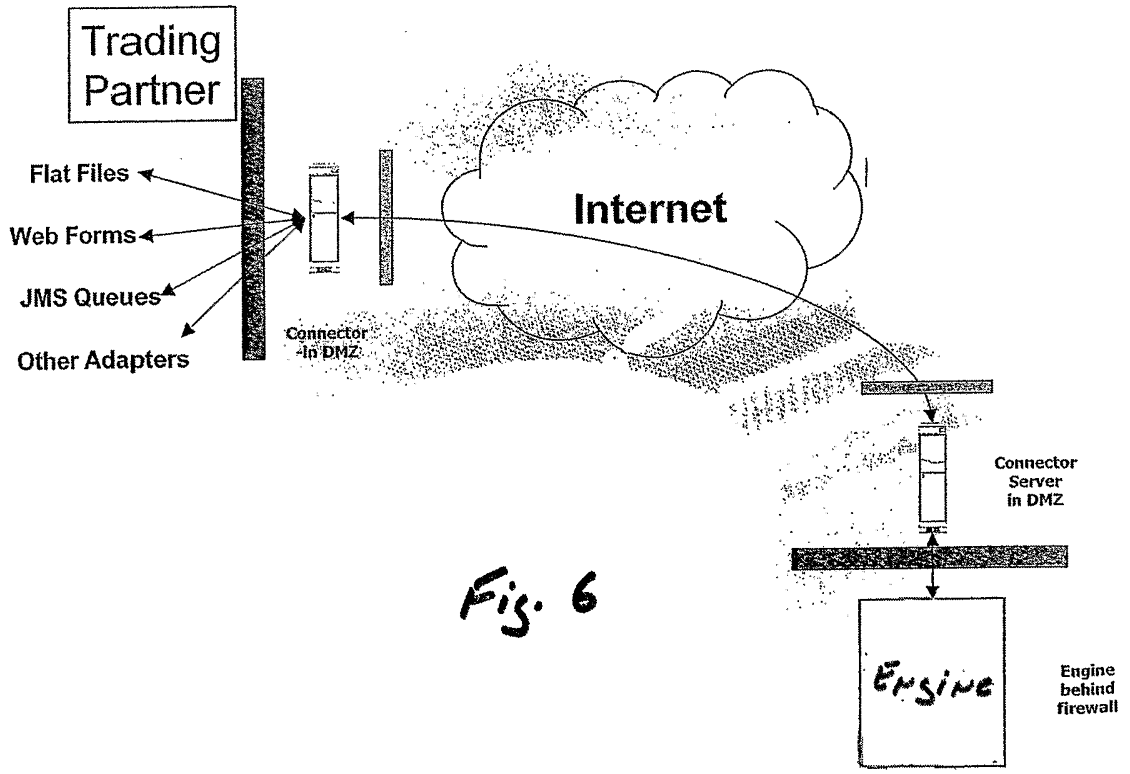

Trading Partner
Flat Files
Web Forms
JMS Queues
Other Adapters
Internet
Connector in DMZ
Connector Server in DMZ
Engine
Engine behind firewall

Fig. 6

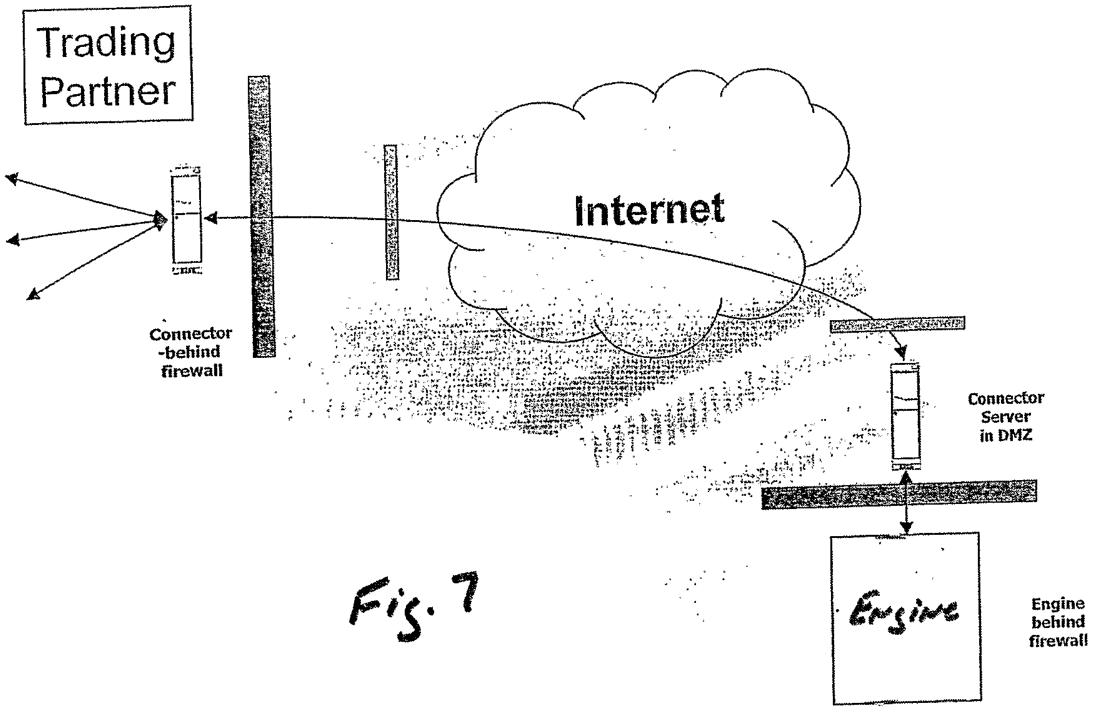

Trading Partner
Internet
Connector -behind firewall
Connector Server in DMZ
Engine
Engine behind firewall

Fig. 7

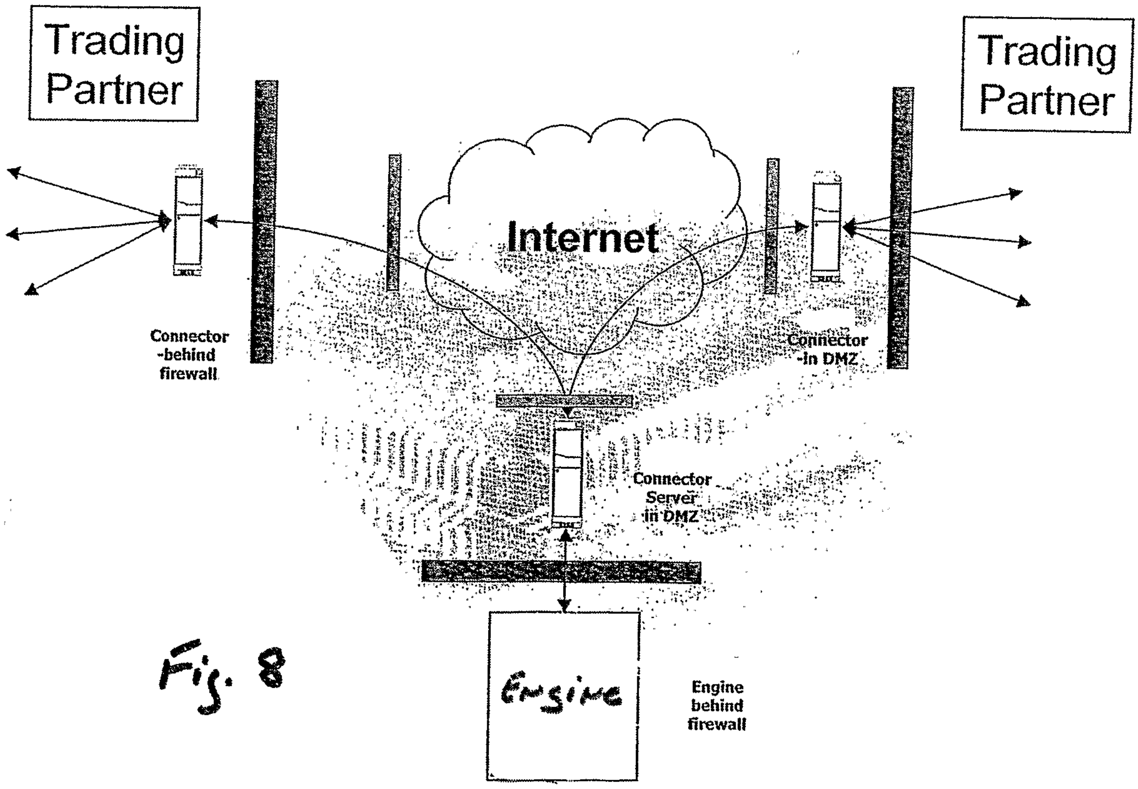

Trading Partner
Trading Partner
Internet
Connector -behind firewall
Connector -in DMZ
Connector Server in DMZ
Engine
Engine behind firewall

Fig. 8

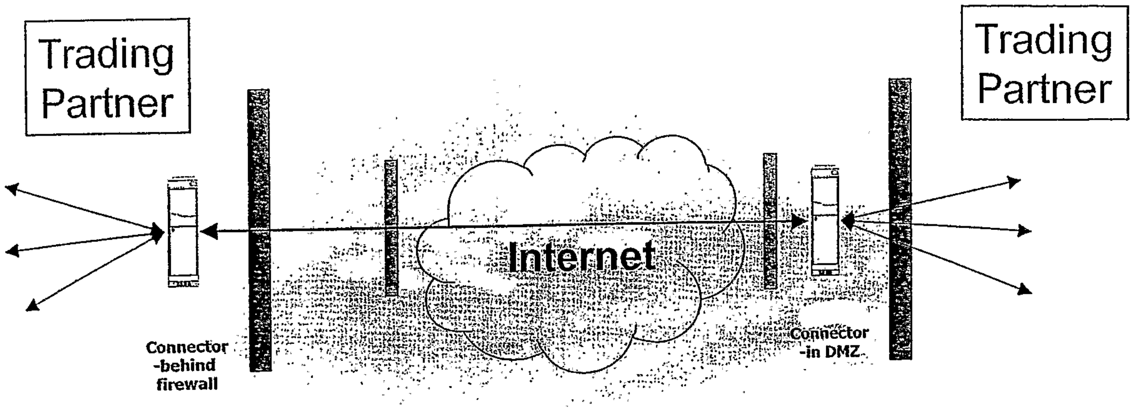

Trading Partner
Trading Partner
Internet
Connector -behind firewall
Connector -in DMZ

Fig. 9

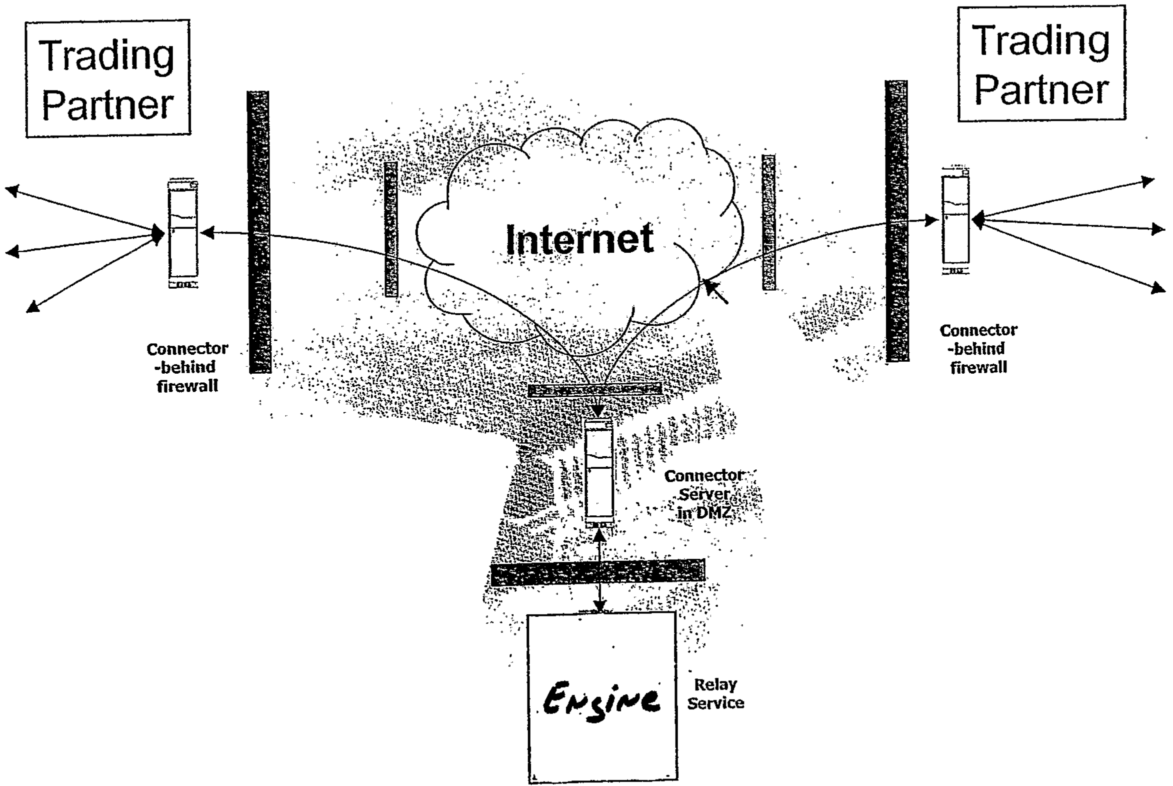

Trading Partner
Trading Partner
Internet
Connector -behind firewall
Connector -behind firewall
Connector Server in DMZ
Engine
Relay Service

Fig. 10

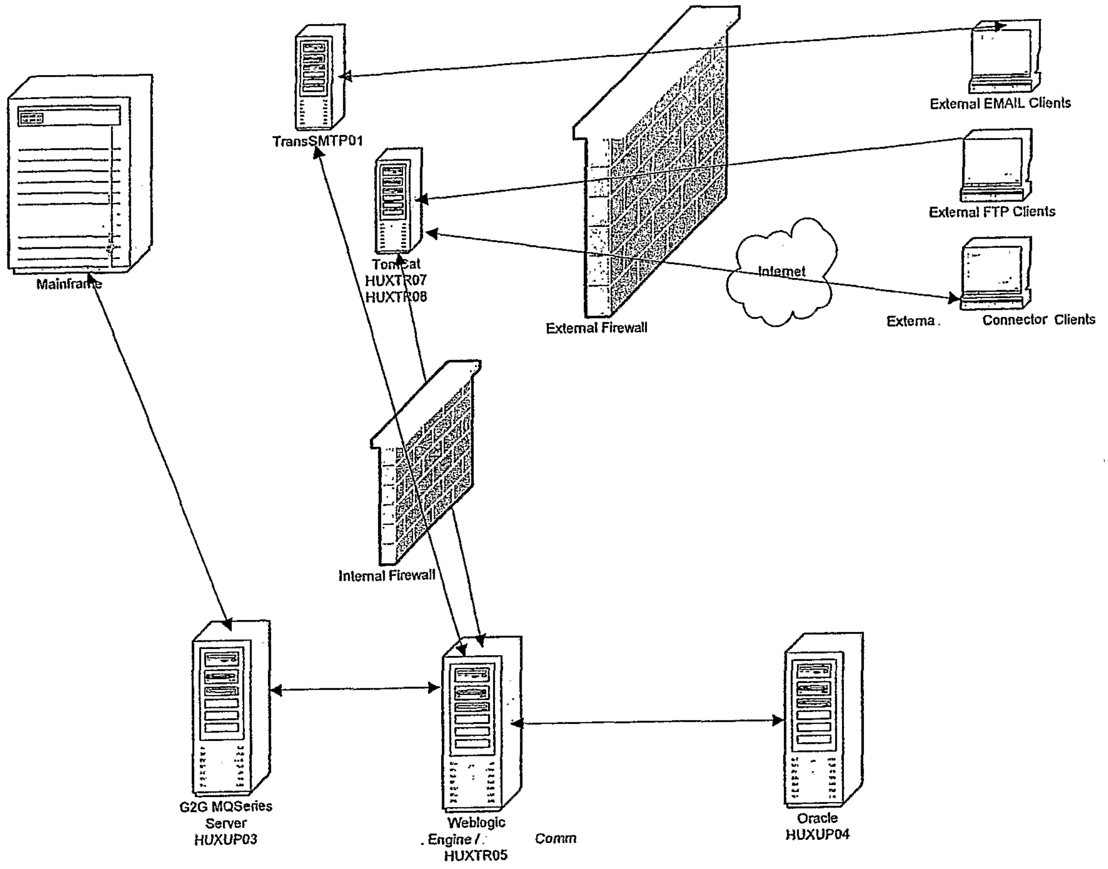

TransSMTP01
External EMAIL Clients
External FTP Clients
Mainframe
TomCat
HUXTR07
HUXTR08
Internet
External Firewall
Externa. Connector Clients
Internal Firewall
G2G MQSeries
Server
HUXUP03
Weblogic
. Engine / Comm
HUXTR05
Oracle
HUXUP04

Fig. 11

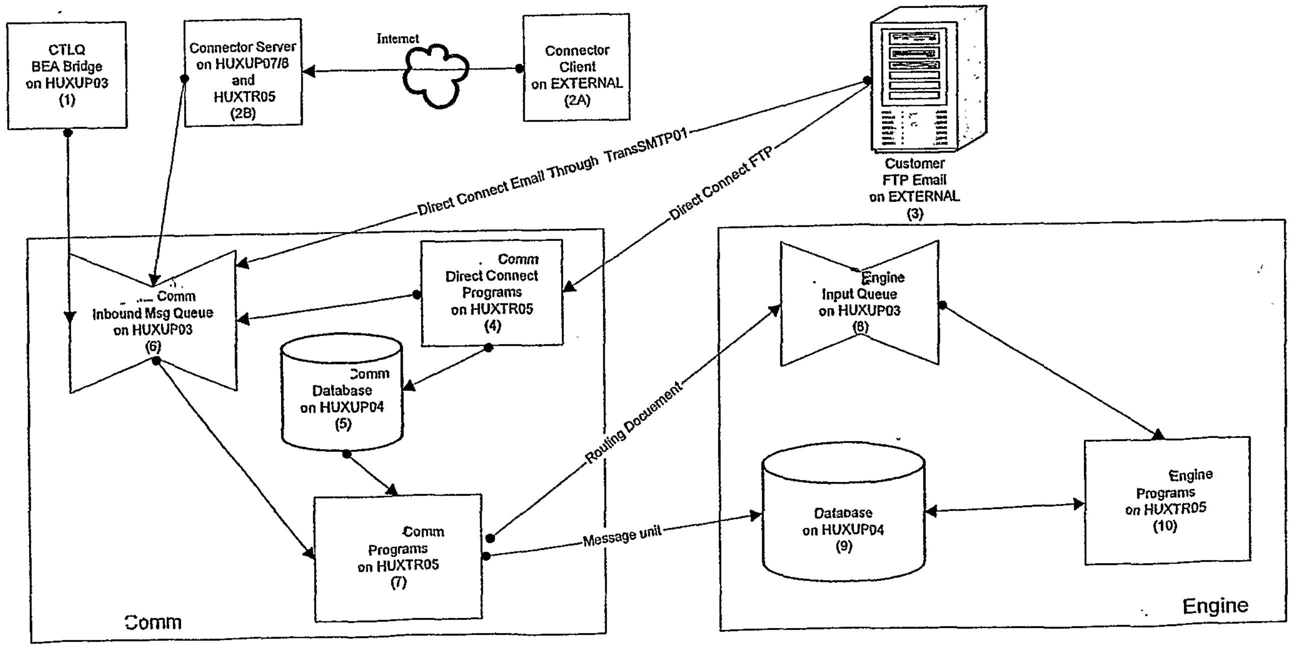

CTLQ
BEA Bridge
on HUXUP03
(1)
Connector Server
on HUXUP07/8
and
HUXTR05
(2B)
Internet
Connector
Client
on EXTERNAL
(2A)
Customer
FTP Email
on EXTERNAL
(3)
Direct Connect Email Through TransSMTP01
Direct Connect FTP
Comm
Inbound Msg Queue
on HUXUP03
(6)
Comm
Direct Connect
Programs
on HUXTR05
(4)
Comm
Database
on HUXUP04
(5)
Comm
Programs
on HUXTR05
(7)
Comm
Routing Document
Message unit
Engine
Input Queue
on HUXUP03
(8)
Database
on HUXUP04
(9)
Engine
Programs
on HUXTR05
(10)
Engine

Fig. 12

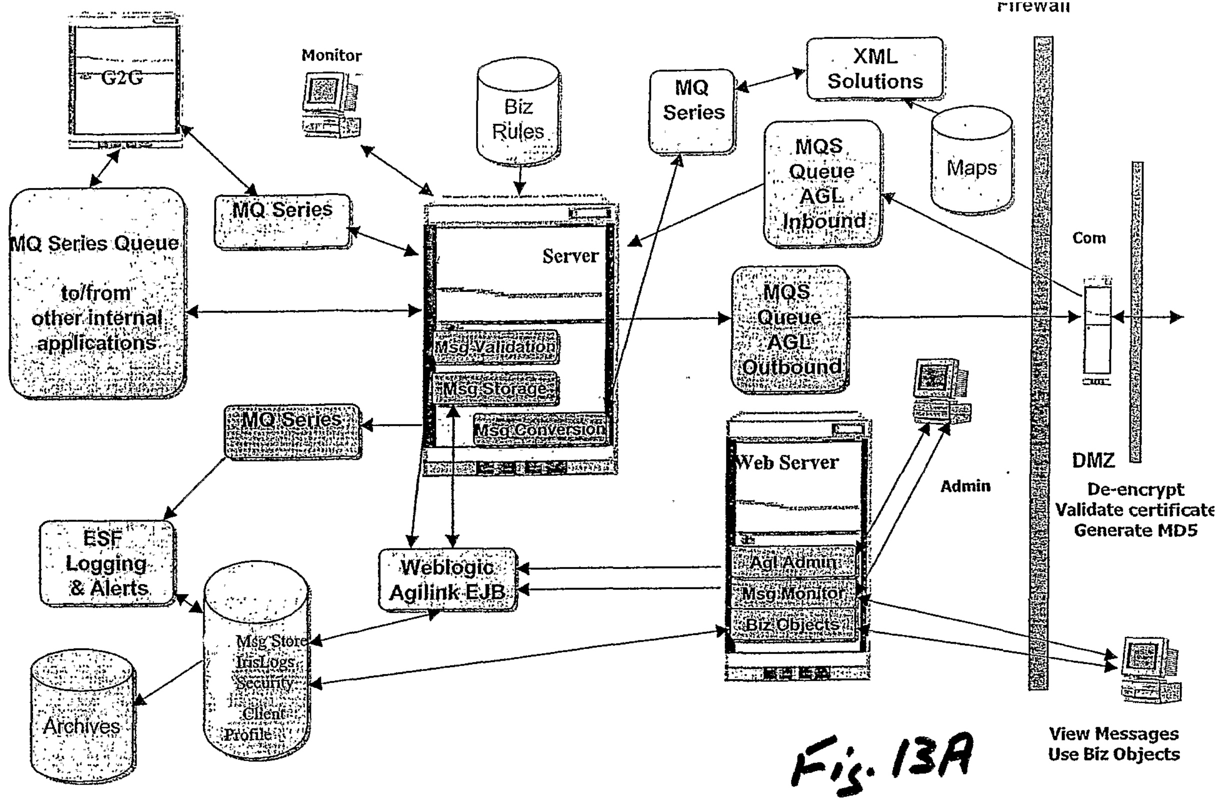


Fig. 13A

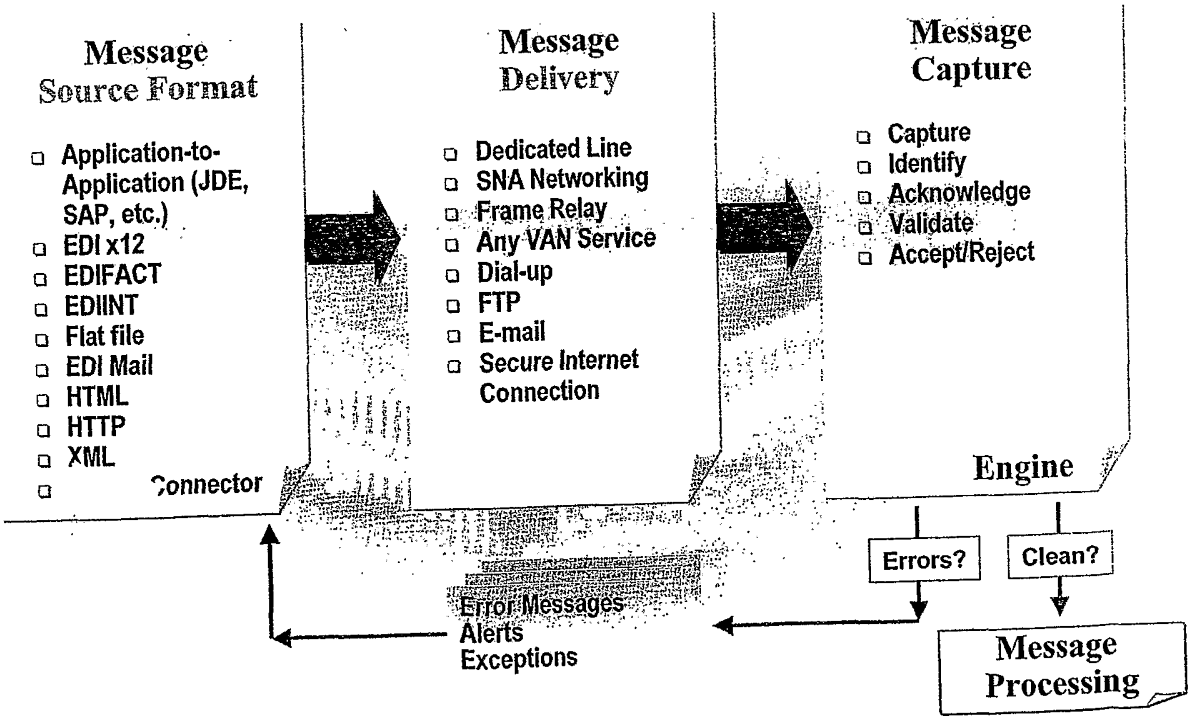


Fig. 14

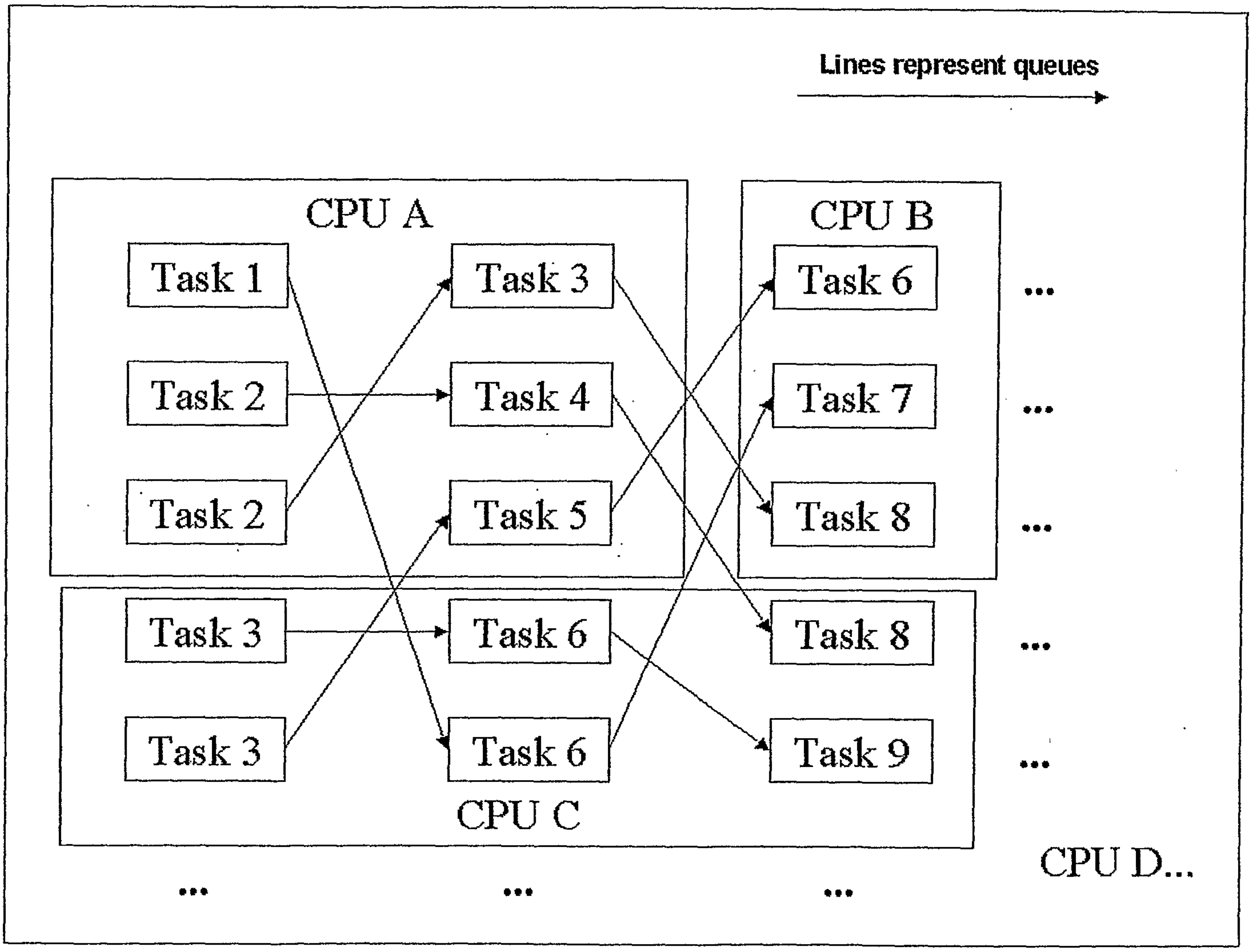

Lines represent queues
CPU A
Task 1
Task 2
Task 2
Task 3
Task 4
Task 5
CPU B
Task 6
Task 7
Task 8
Task 3
Task 3
Task 6
Task 6
Task 8
Task 9
CPU C
CPU D...
...

Fig. 13B

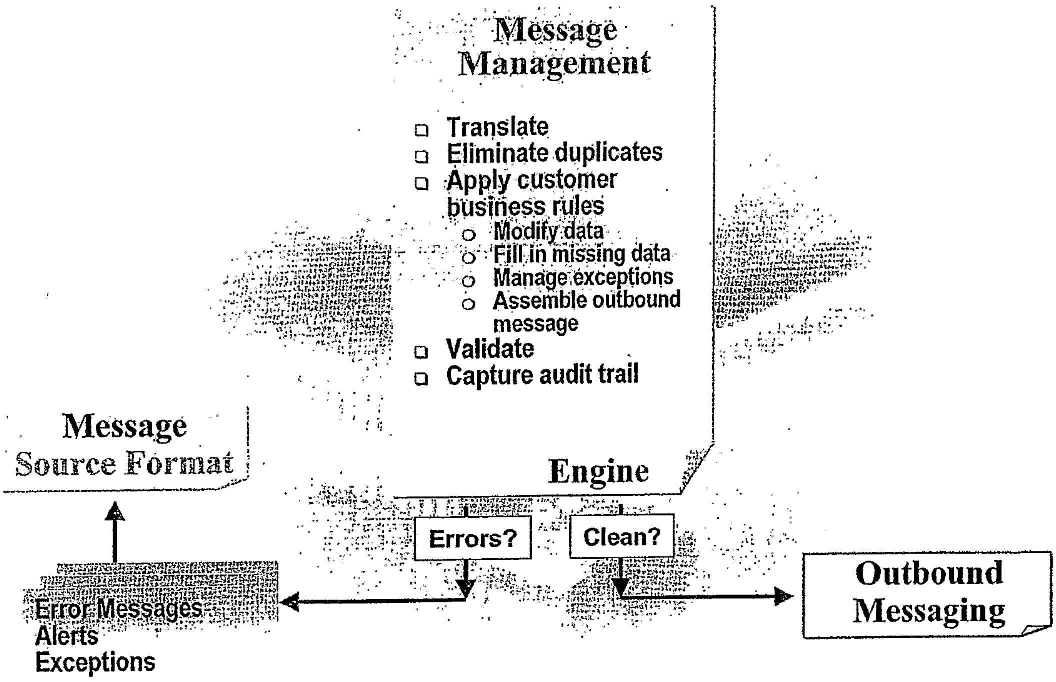

Message Management
□ Translate
□ Eliminate duplicates
□ Apply customer business rules
○ Modify data
○ Fill in missing data
○ Manage exceptions
○ Assemble outbound message
□ Validate
□ Capture audit trail
Engine
Message Source Format
Errors?
Clean?
Error Messages
Alerts
Exceptions
Outbound Messaging

Fig. 15

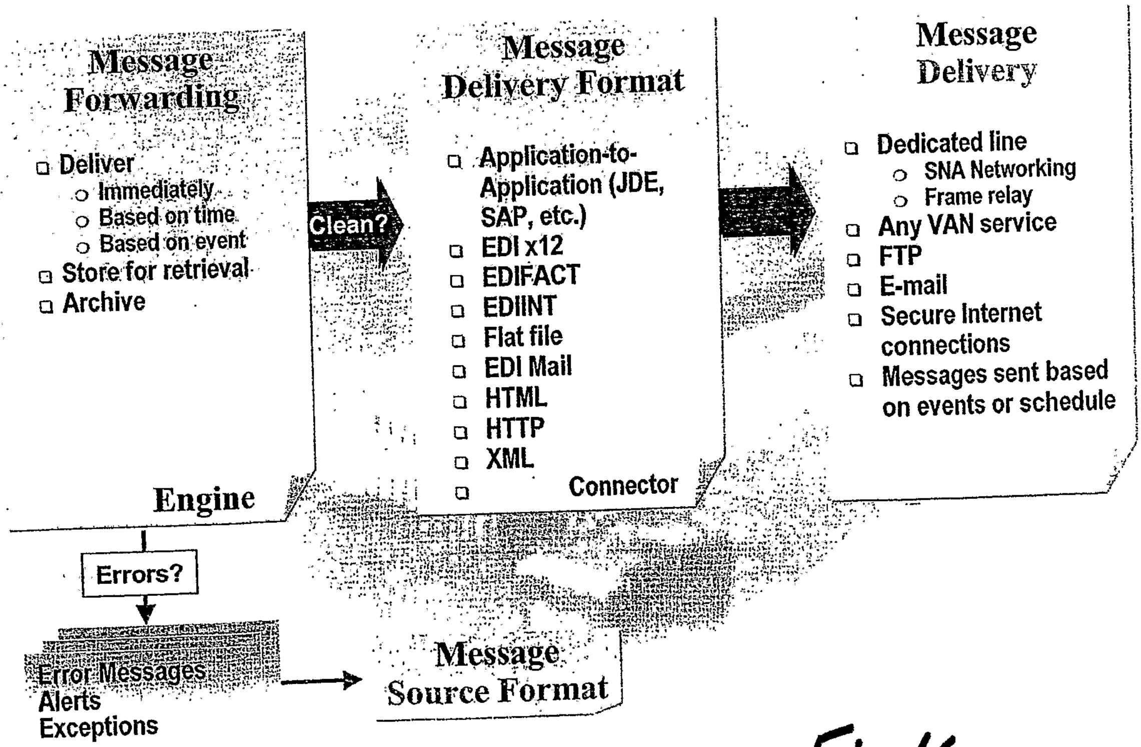

Message Forwarding
□ Deliver
○ Immediately
○ Based on time
○ Based on event
□ Store for retrieval
□ Archive
Engine
Clean?
Message Delivery Format
□ Application-to-Application (JDE, SAP, etc.)
□ EDI x12
□ EDIFACT
□ EDIINT
□ Flat file
□ EDI Mail
□ HTML
□ HTTP
□ XML
□ Connector
Message Delivery
□ Dedicated line
○ SNA Networking
○ Frame relay
□ Any VAN service
□ FTP
□ E-mail
□ Secure Internet connections
□ Messages sent based on events or schedule
Errors?
Error Messages
Alerts
Exceptions
Message Source Format

Fig. 16

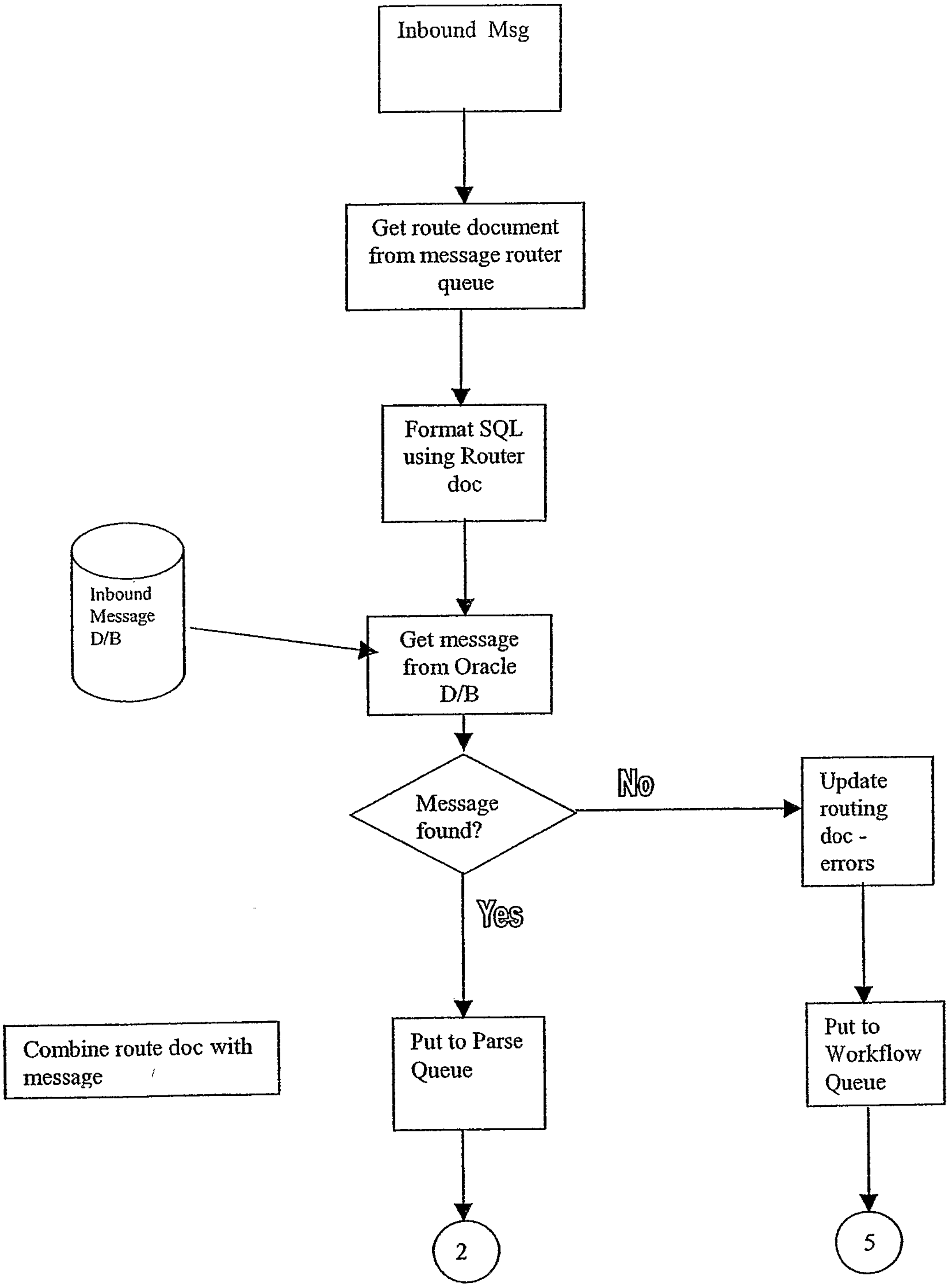

Inbound Msg
Get route document from message router queue
Format SQL using Router doc
Inbound Message D/B
Get message from Oracle D/B
Message found?
No
Update routing doc - errors
Yes
Combine route doc with message
Put to Parse Queue
Put to Workflow Queue
2
5

Fig. 17

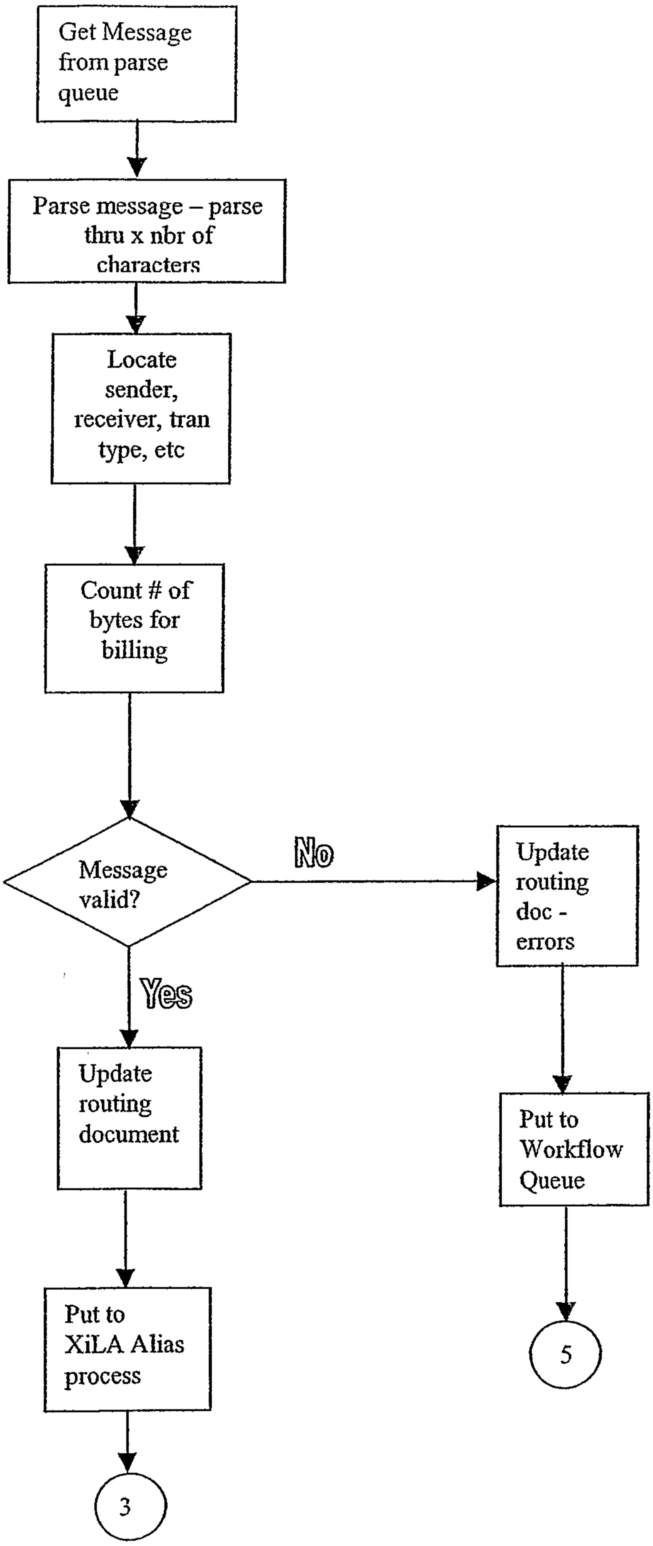

Get Message from parse queue
Parse message – parse thru x nbr of characters
Locate sender, receiver, tran type, etc
Count # of bytes for billing
Message valid?
No
Update routing doc - errors
Put to Workflow Queue
5
Yes
Update routing document
Put to XiLA Alias process
3

Fig. 18

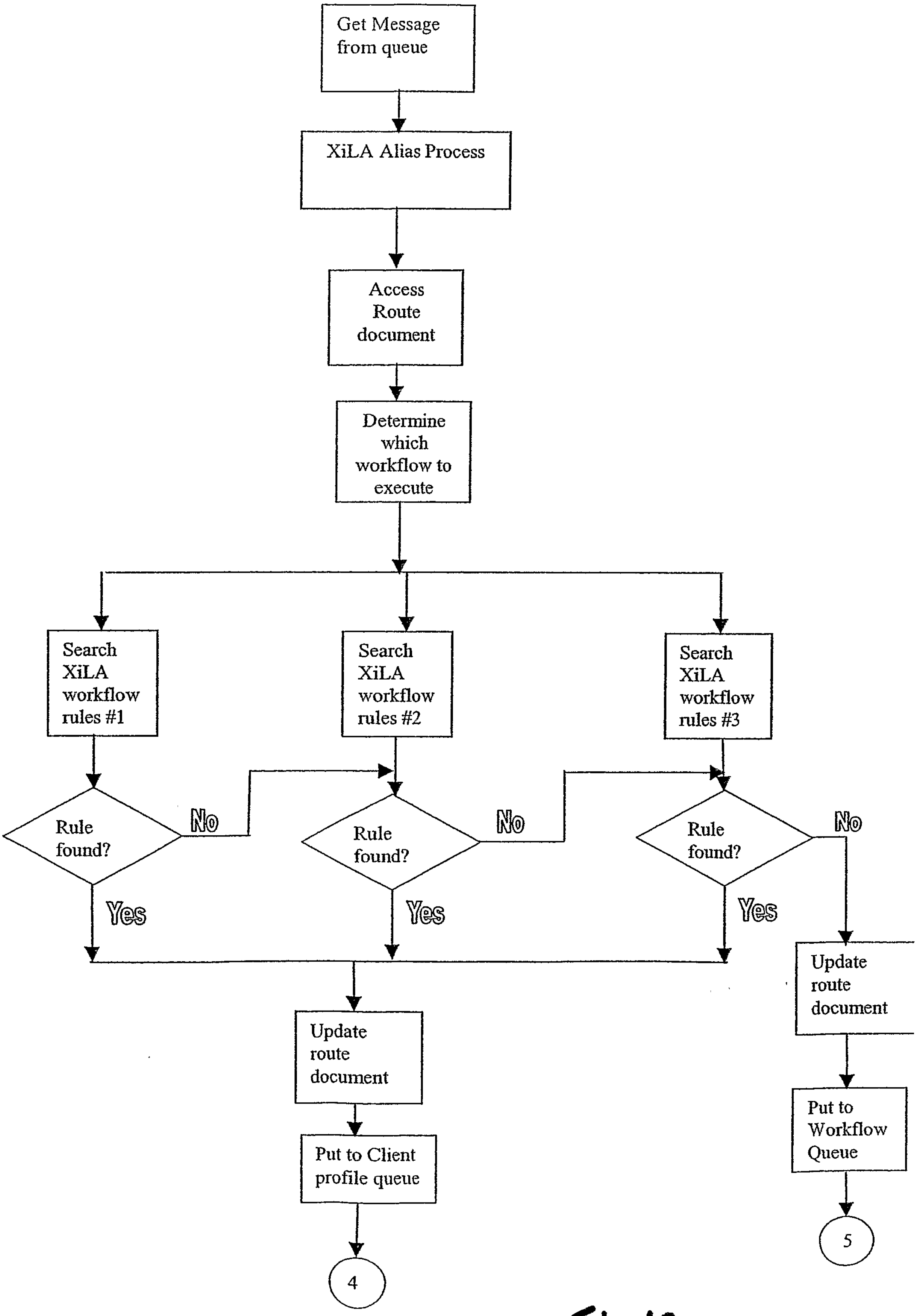

Get Message from queue
XiLA Alias Process
Access Route document
Determine which workflow to execute
Search XiLA workflow rules #1
Search XiLA workflow rules #2
Search XiLA workflow rules #3
Rule found?
No
Yes
Rule found?
No
Yes
Rule found?
No
Yes
Update route document
Put to Client profile queue
4
Update route document
Put to Workflow Queue
5

Fig. 19

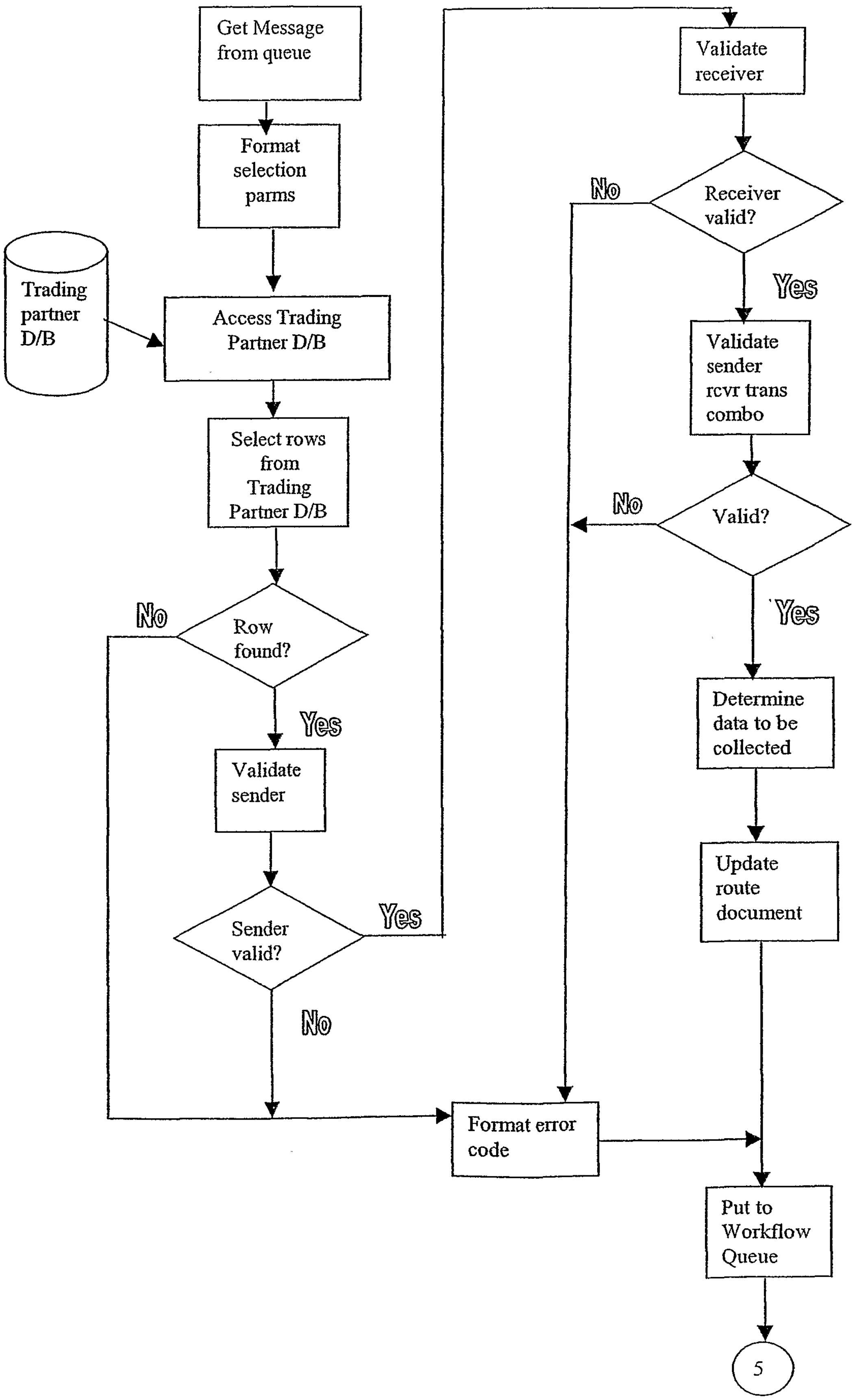

Get Message from queue
Format selection parms
Trading partner D/B
Access Trading Partner D/B
Select rows from Trading Partner D/B
Row found?
No
Yes
Validate sender
Sender valid?
Yes
No
Validate receiver
Receiver valid?
No
Yes
Validate sender rcvr trans combo
Valid?
No
Yes
Determine data to be collected
Update route document
Format error code
Put to Workflow Queue
5

Fig. 20

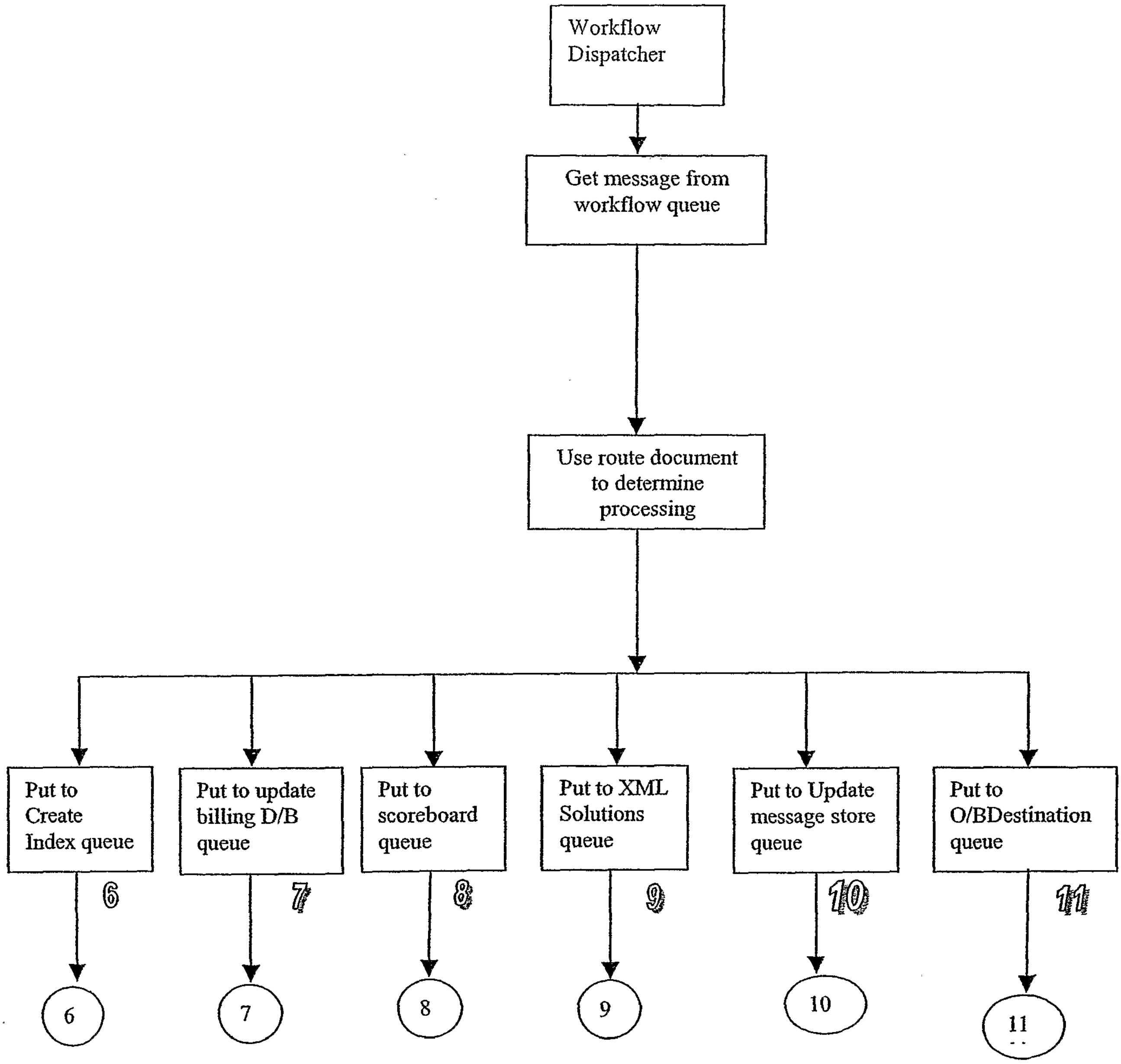

Workflow Dispatcher
Get message from workflow queue
Use route document to determine processing
Put to Create Index queue
Put to update billing D/B queue
Put to scoreboard queue
Put to XML Solutions queue
Put to Update message store queue
Put to O/BDestination queue
6
7
8
9
10
11
6
7
8
9
10
11

Fig. 21

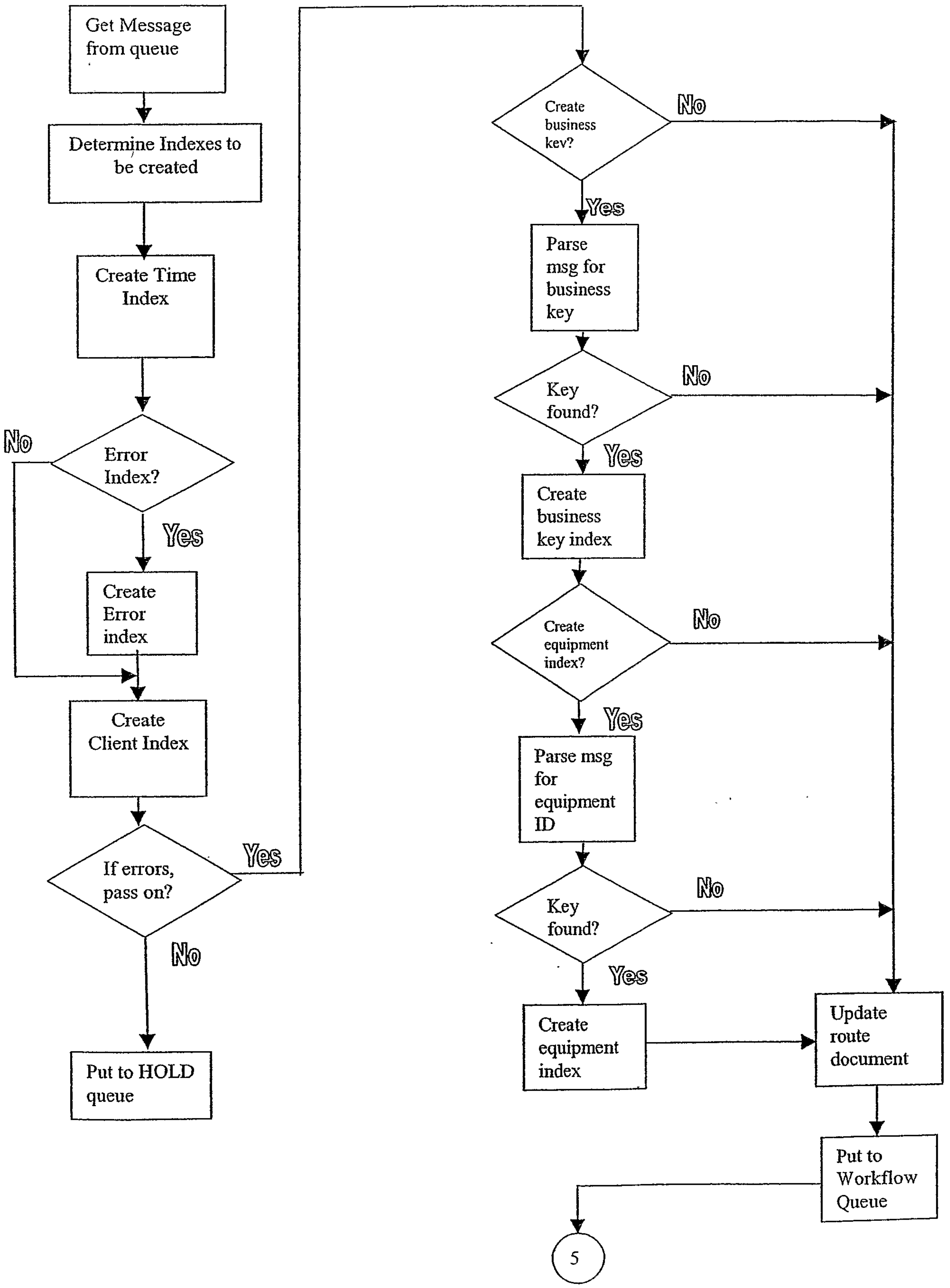

Get Message from queue
Determine Indexes to be created
Create Time Index
Error Index?
No
Yes
Create Error index
Create Client Index
If errors, pass on?
Yes
No
Put to HOLD queue
Create business kev?
No
Yes
Parse msg for business key
Key found?
No
Yes
Create business key index
Create equipment index?
No
Yes
Parse msg for equipment ID
Key found?
No
Yes
Create equipment index
Update route document
Put to Workflow Queue
5

Fig. 22

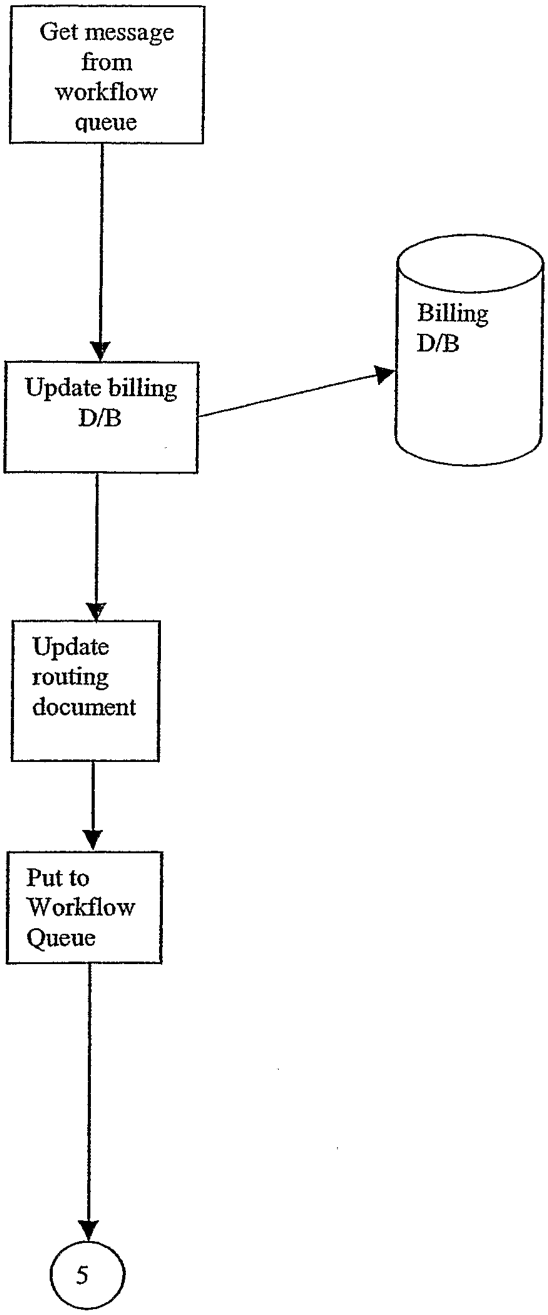

Get message from workflow queue
Update billing D/B
Billing D/B
Update routing document
Put to Workflow Queue
5

Fig. 23

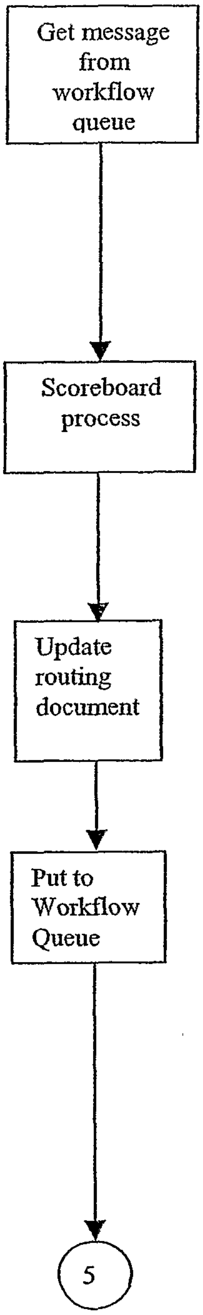

Get message from workflow queue
Scoreboard process
Update routing document
Put to Workflow Queue
5

Fig. 24

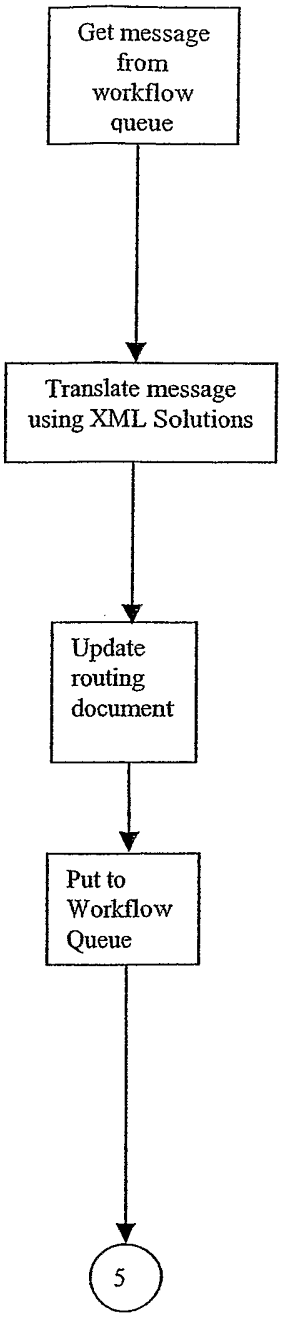

Get message from workflow queue
Translate message using XML Solutions
Update routing document
Put to Workflow Queue
5

Fig. 25

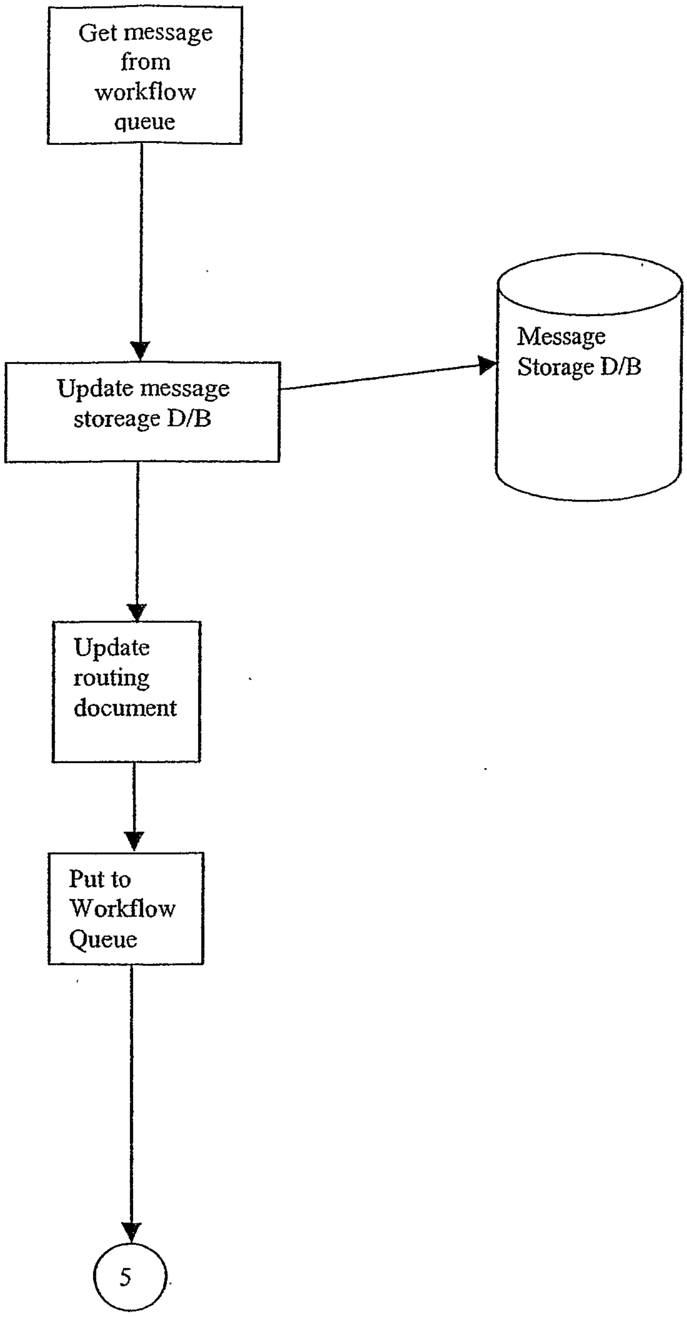

Get message from workflow queue
Update message storeage D/B
Message Storage D/B
Update routing document
Put to Workflow Queue
5

Fig. 26

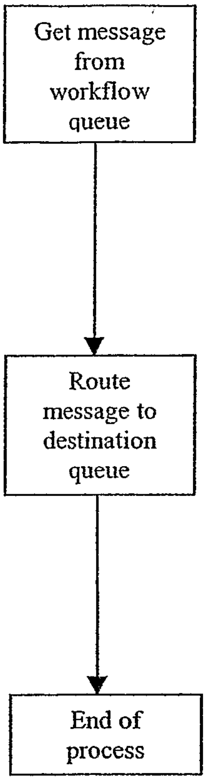

Get message from workflow queue
Route message to destination queue
End of process

Fig. 27

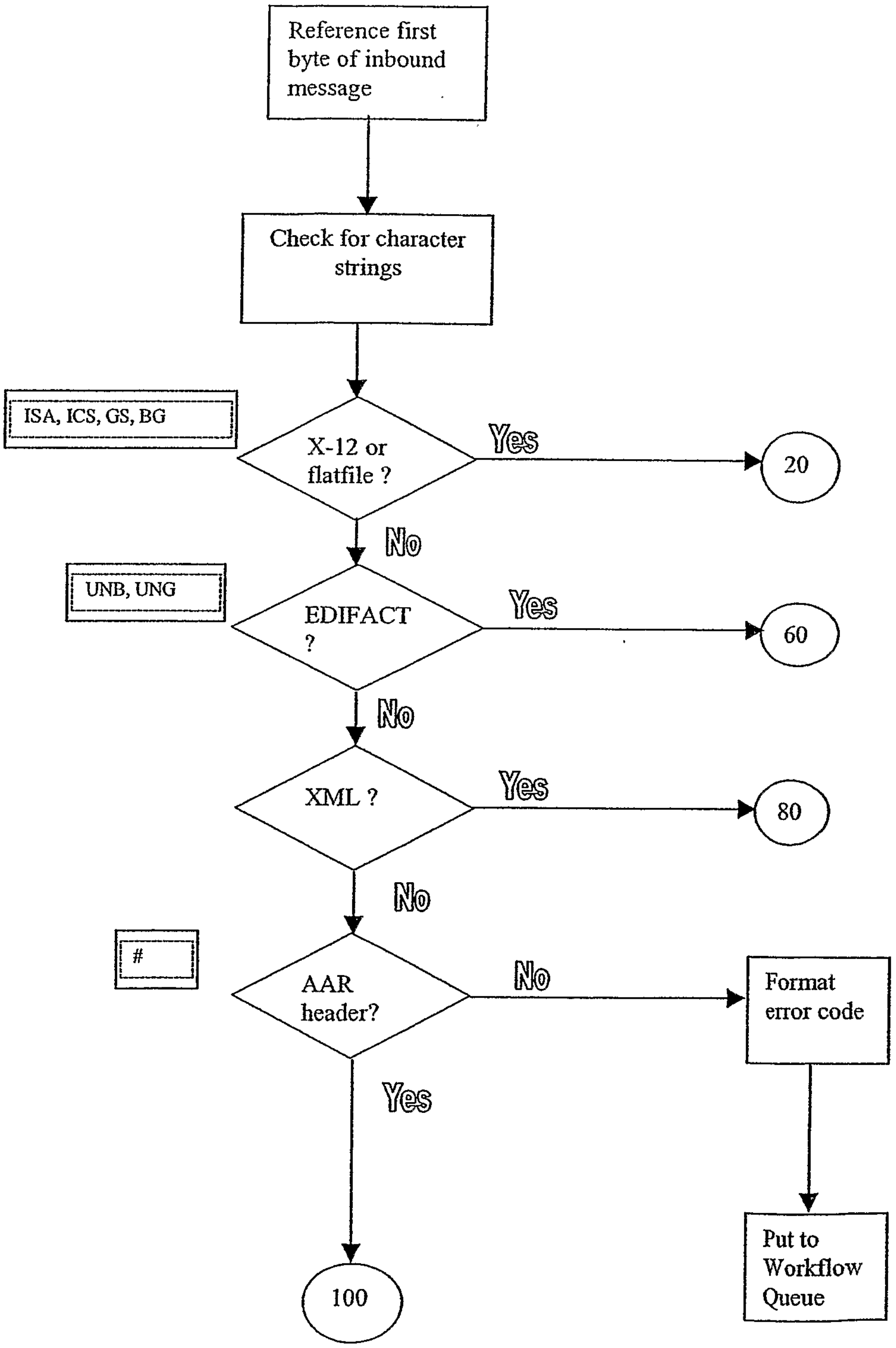
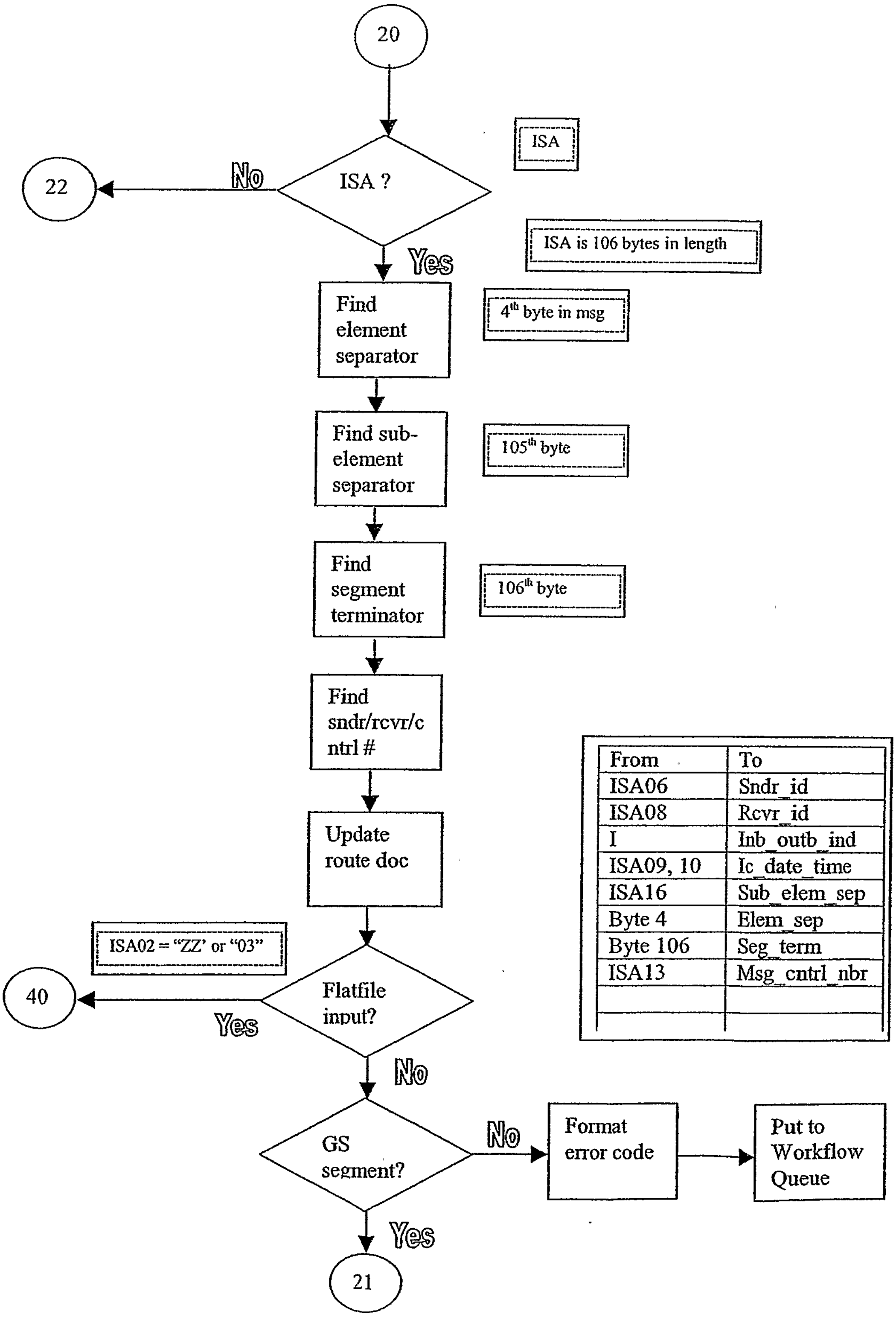
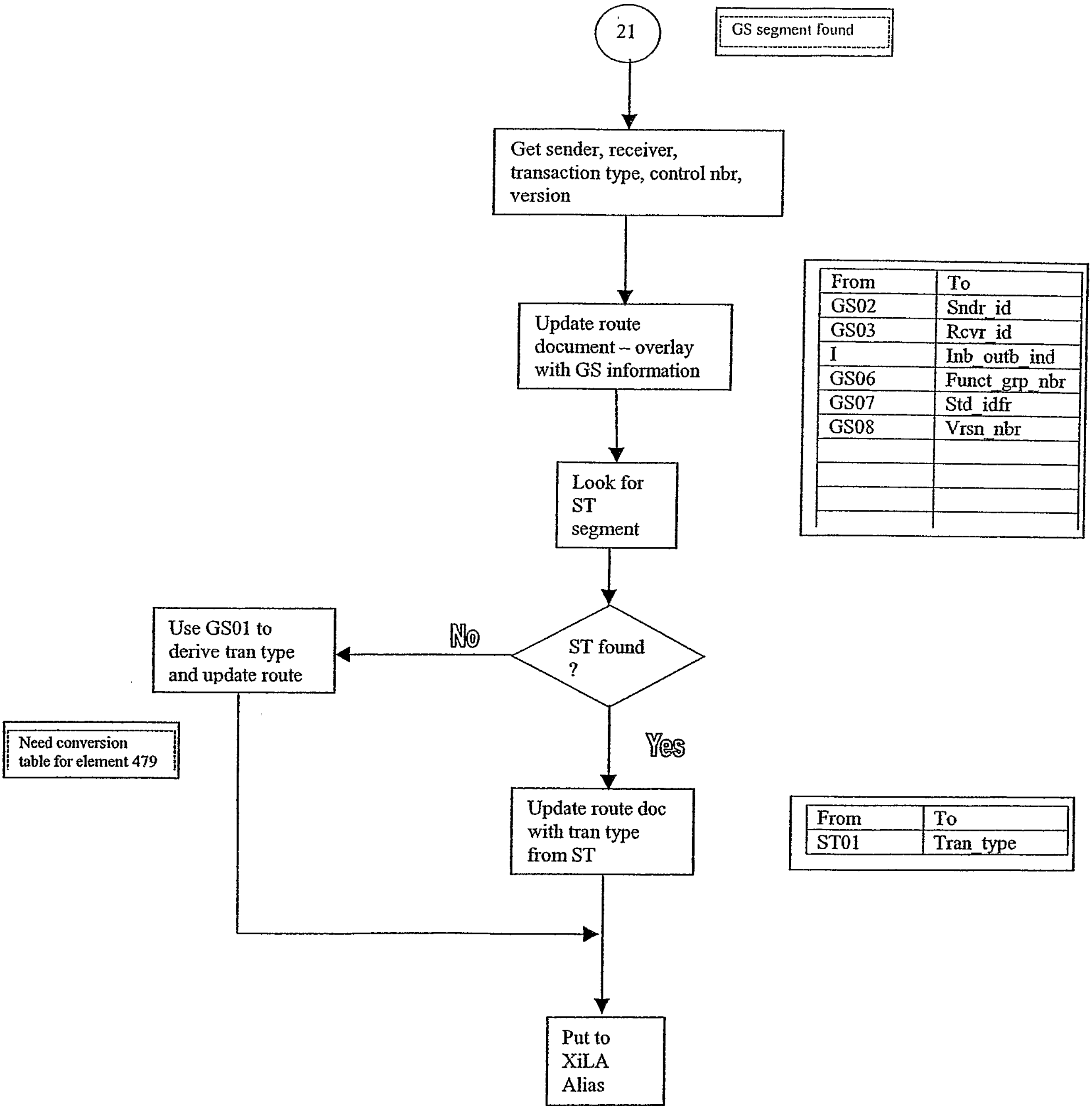

Reference first byte of inbound message
Check for character strings
ISA, ICS, GS, BG
X-12 or flatfile ?
Yes
20
No
UNB, UNG
EDIFACT ?
Yes
60
No
XML ?
Yes
80
No
#
AAR header?
No
Format error code
Put to Workflow Queue
Yes
100

Fig. 28



From	To
ISA06	Sndr_id
ISA08	Rcvr_id
I	Inb_outb_ind
ISA09, 10	Ic_date_time
ISA16	Sub_elem_sep
Byte 4	Elem_sep
Byte 106	Seg_term
ISA13	Msg_cntrl_nbr

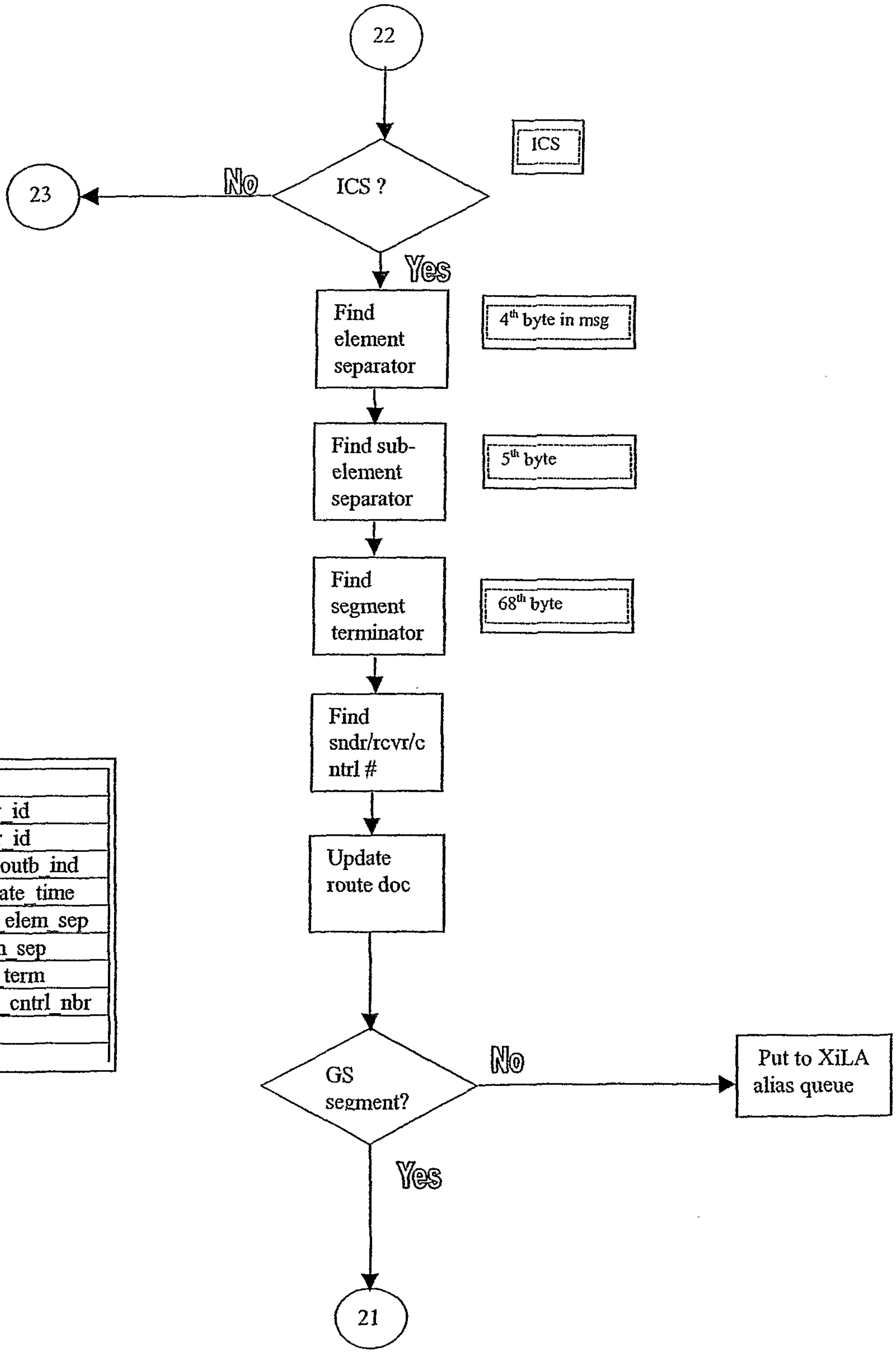
Fig. 29


21
GS segment found
Get sender, receiver, transaction type, control nbr, version
Update route document – overlay with GS information
Look for ST segment
ST found ?
No
Use GS01 to derive tran type and update route
Need conversion table for element 479
Yes
Update route doc with tran type from ST
Put to XiLA Alias

From	To
GS02	Sndr_id
GS03	Rcvr_id
I	Inb_outb_ind
GS06	Funct_grp_nbr
GS07	Std_idfr
GS08	Vrsn_nbr

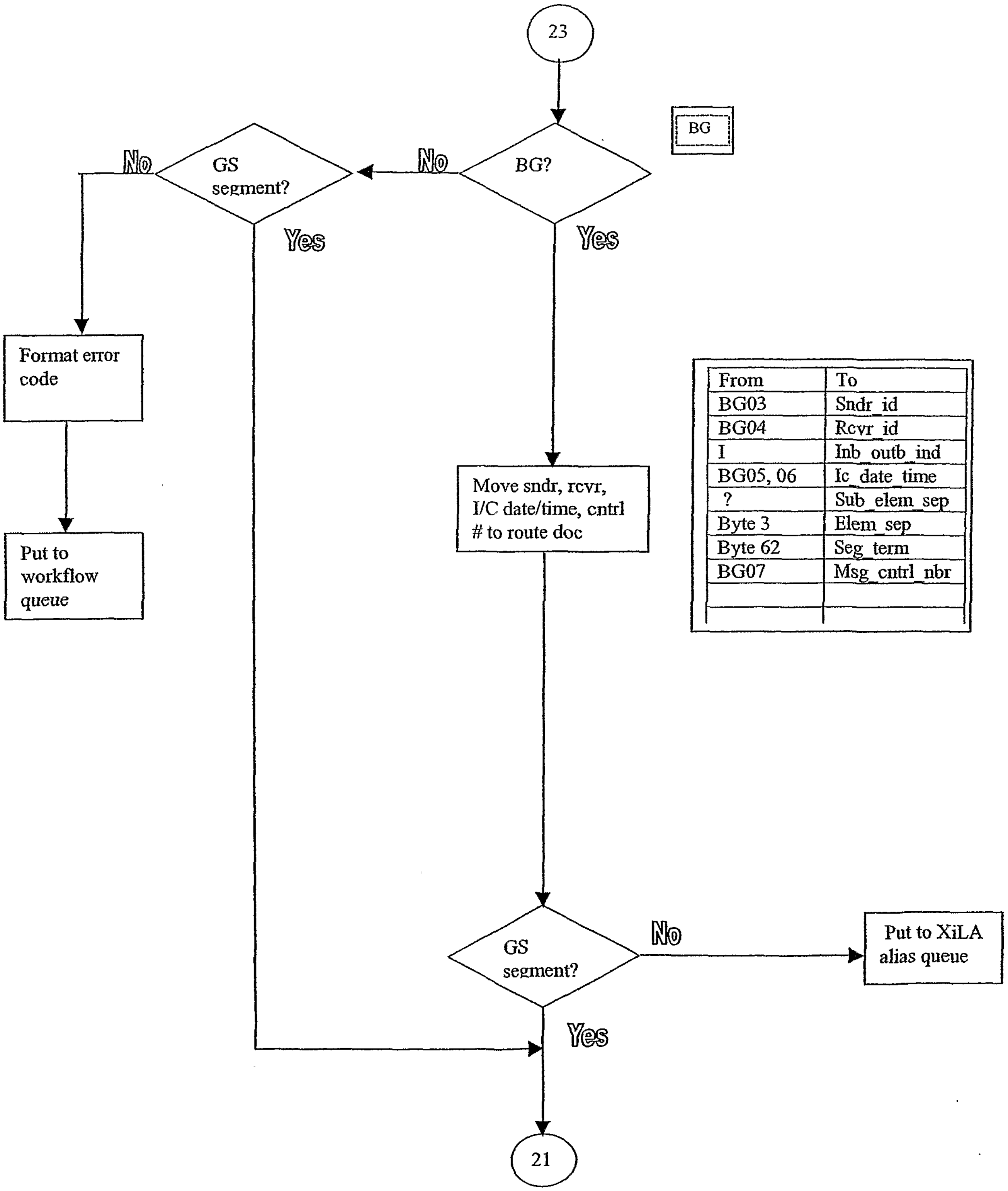
From	To
ST01	Tran_type

Fig. 30



From	To
ICS06	Sndr_id
ICS08	Rcvr_id
I	Inb_outb_ind
ICS09, 10	Ic_date_time
Byte 5	Sub_elem_sep
Byte 4	Elem_sep
Byte 68	Seg_term
ICS11	Msg_cntrl_nbr

Fig. 31



From	To
BG03	Sndr_id
BG04	Rcvr_id
I	Inb_outb_ind
BG05, 06	Ic_date_time
?	Sub_elem_sep
Byte 3	Elem_sep
Byte 62	Seg_term
BG07	Msg_cntrl_nbr

Fig. 32

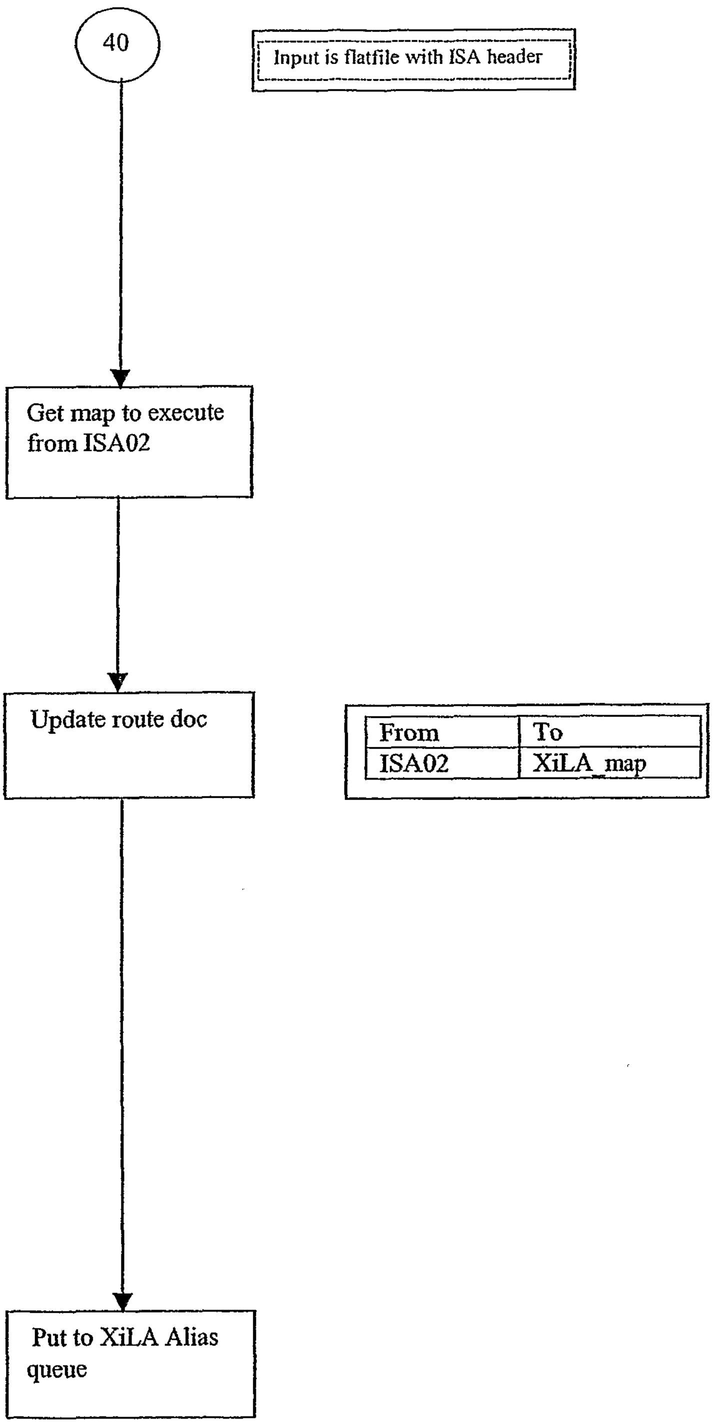
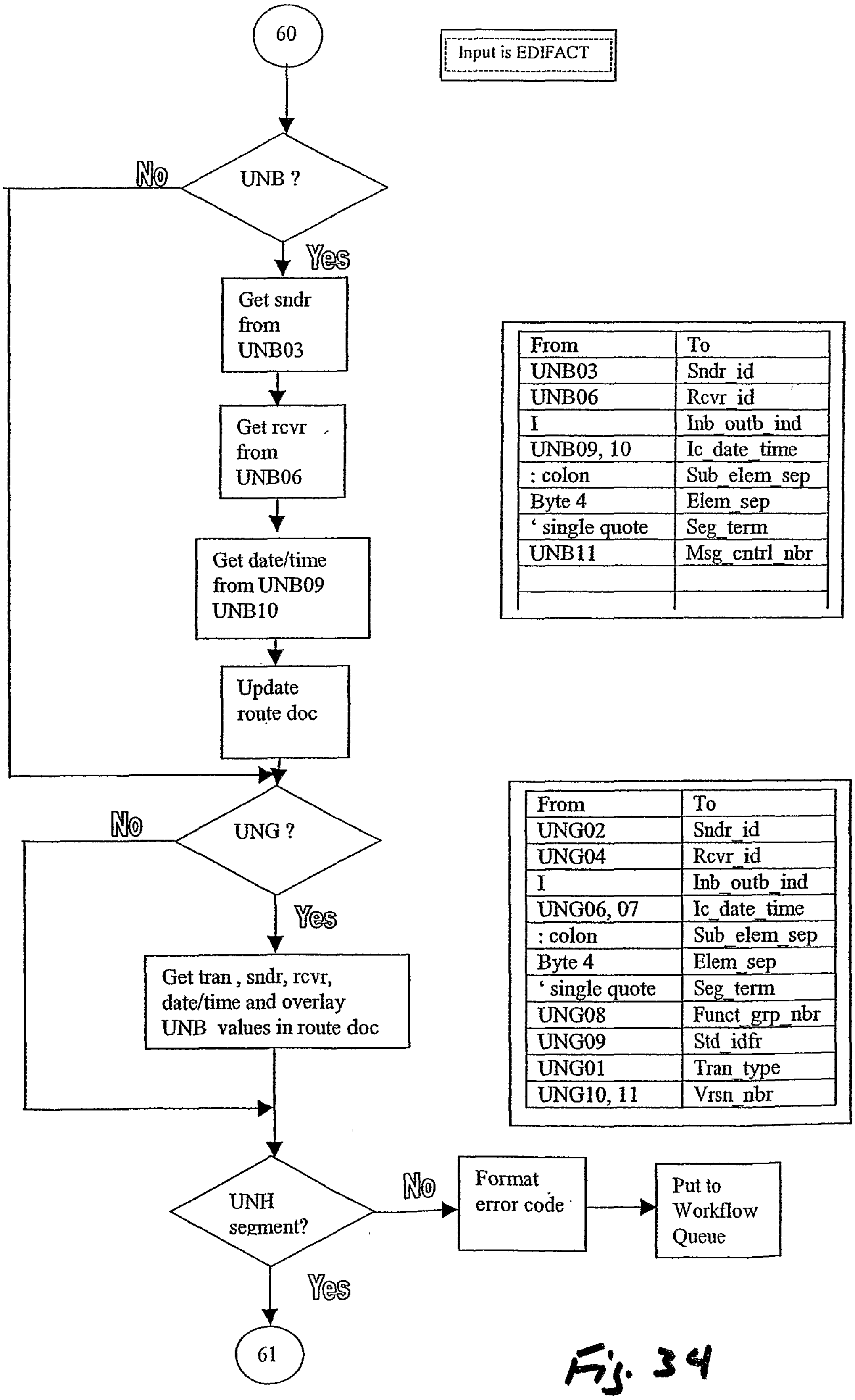

40
Input is flatfile with ISA header
Get map to execute from ISA02
Update route doc
From
To
ISA02
XiLA_map
Put to XiLA Alias queue

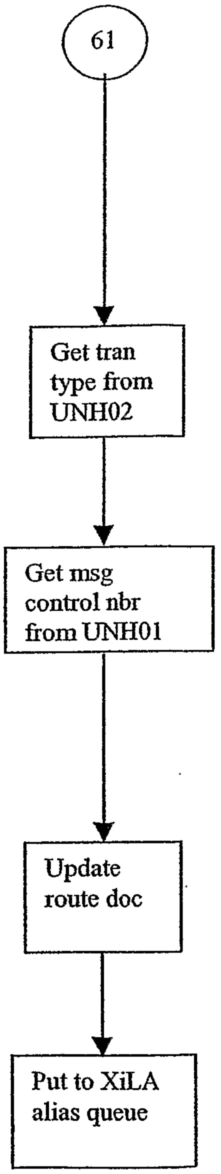
Fig. 33



From	To
UNB03	Sndr_id
UNB06	Rcvr_id
I	Inb_outb_ind
UNB09, 10	Ic_date_time
: colon	Sub_elem_sep
Byte 4	Elem_sep
‘ single quote	Seg_term
UNB11	Msg_cntrl_nbr

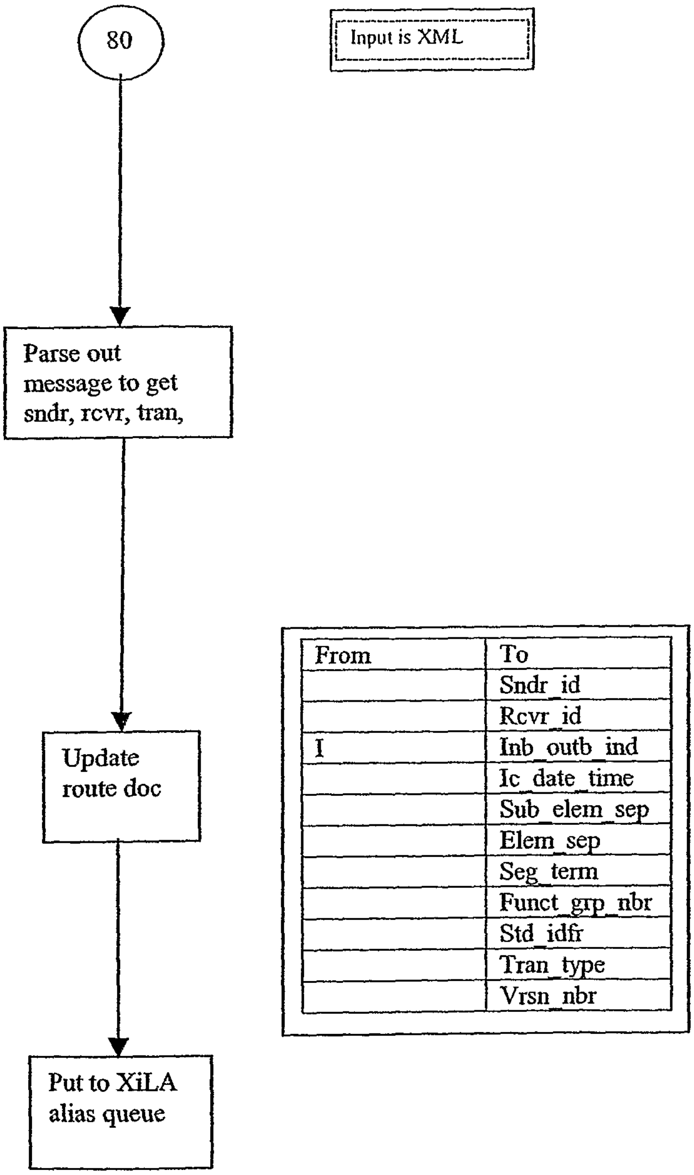
From	To
UNG02	Sndr_id
UNG04	Rcvr_id
I	Inb_outb_ind
UNG06, 07	Ic_date_time
: colon	Sub_elem_sep
Byte 4	Elem_sep
‘ single quote	Seg_term
UNG08	Funct_grp_nbr
UNG09	Std_idfr
UNG01	Tran_type
UNG10, 11	Vrsn_nbr

Fig. 34



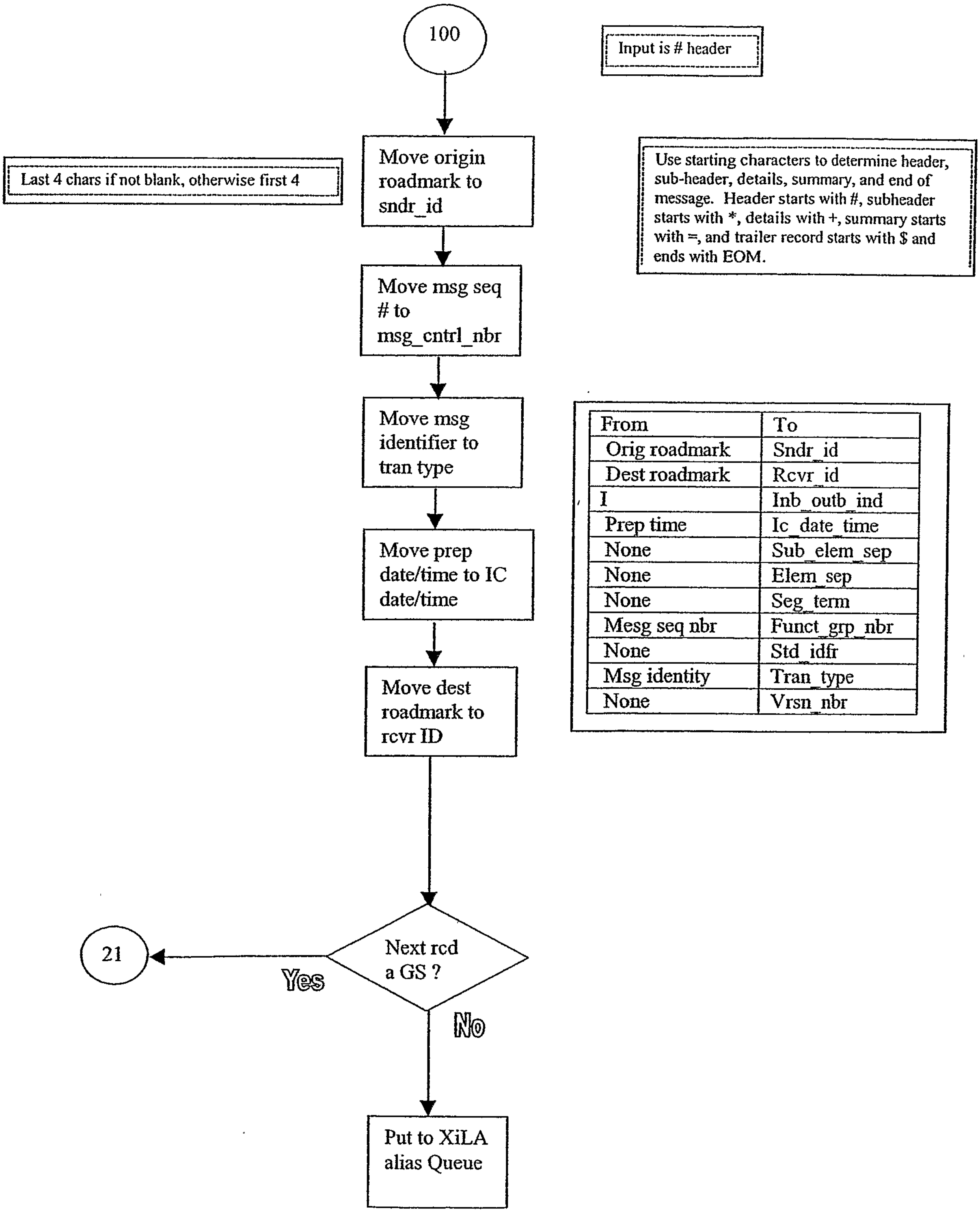
From	To
UNH02	Tran_type
UNH01	Set_cntrl_nbr

Fig. 35



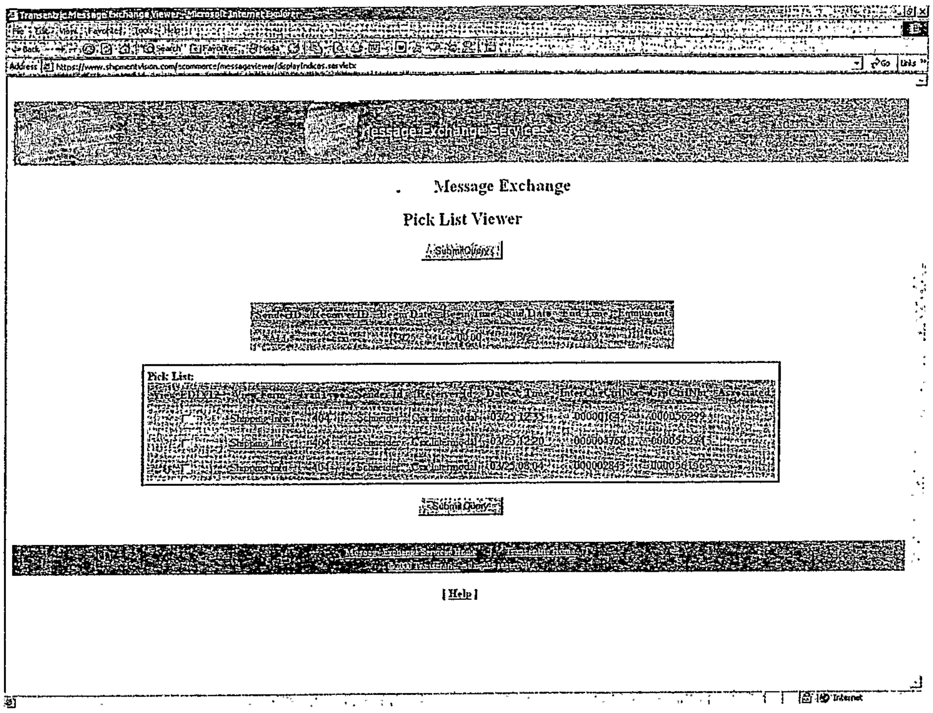
From	To
	Sndr_id
	Rcvr_id
I	Inb_outb_ind
	Ic_date_time
	Sub_elem_sep
	Elem_sep
	Seg_term
	Funct_grp_nbr
	Std_idfr
	Tran_type
	Vrsn_nbr

Fig. 36


100
Input is # header
Last 4 chars if not blank, otherwise first 4
Move origin roadmark to sndr_id
Use starting characters to determine header, sub-header, details, summary, and end of message. Header starts with #, subheader starts with *, details with +, summary starts with =, and trailer record starts with $ and ends with EOM.
Move msg seq # to msg_cntrl_nbr
Move msg identifier to tran type
Move prep date/time to IC date/time
Move dest roadmark to rcvr ID
Next rcd a GS ?
Yes
21
No
Put to XiLA alias Queue

From	To
Orig roadmark	Sndr_id
Dest roadmark	Rcvr_id
I	Inb_outb_ind
Prep time	Ic_date_time
None	Sub_elem_sep
None	Elem_sep
None	Seg_term
Mesg seq nbr	Funct_grp_nbr
None	Std_idfr
Msg identity	Tran_type
None	Vrsn_nbr

Fig. 37


Transentric Message Exchange Viewer - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address https://www.shipmentvision.com/ecommerce/messageviewer/displayIndices.servletx
Go Links
Message Exchange Services
Submit Query
Submit Query
Message Exchange Services Home
Transentric Home
[Help]
Internet

Message Exchange

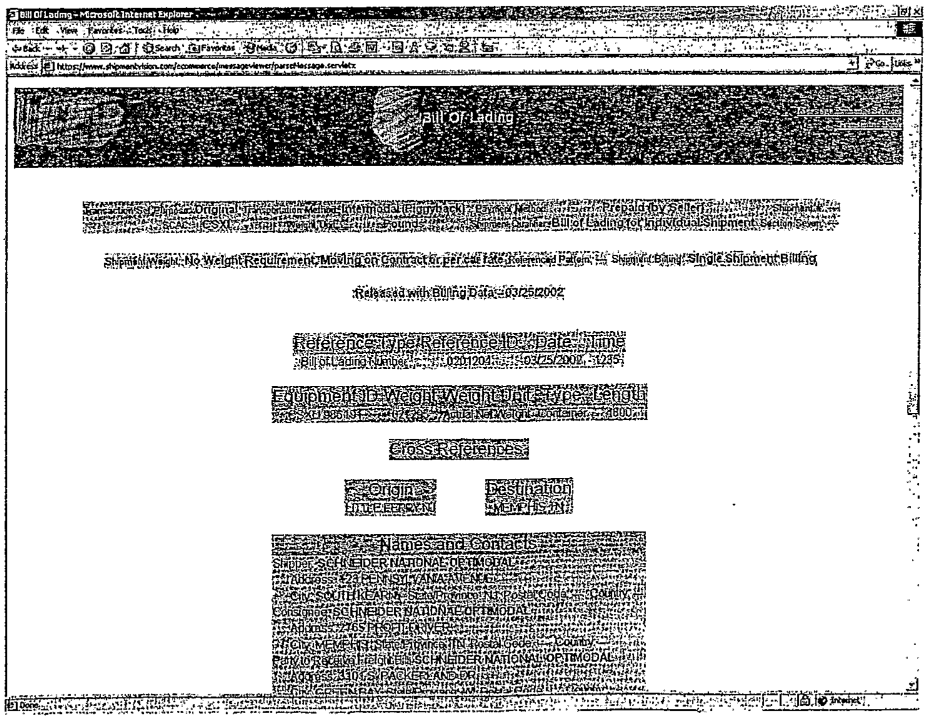
Pick List Viewer

SenderID	ReceiverID	Begin Date	Begin Time	End Date	End Time	Equipment
ALL		03/25	00:00	03/25	2359	[illegible]

Pick List:

View EDI X12	View Form	TranType	SenderId	ReceiverId	Date & Time	InterChgCtlNbr	GrpCtlNbr	Associated
□	Shipping Info	404	Schneider	Csx Intermodal	03/25 12:35	000001645	000056299	
□	Shipping Info	404	Schneider	Csx Intermodal	03/25 12:20	000004768	000056294	
□	Shipping Info	404	Schneider	Csx Intermodal	03/25 08:04	000002843	000056166	

Fig. 38


Bill Of Lading - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address https://www.shipmentvision.com/ecommerce/messageviewer/parseMessage.servletx
Go Links
Bill Of Lading
Done
Internet

Transaction Set Purpose: Original Transportation Method: Intermodal (Piggyback) Payment Method: Prepaid (by Seller) Shipment #: SCAC: CSXI Weight Unit: Pounds Shipment Qualifier: Bill of Lading for Individual Shipment Section Seven:

Shipment Weight: No Weight Requirement, Moving on Contract or per car rate Referenced Pattern: Shipment Billing: Single Shipment Billing

Released with Billing Data: 03/25/2002

Reference Type	Reference ID	Date	Time
Bill of Lading Number	0201204	03/25/2002	1235

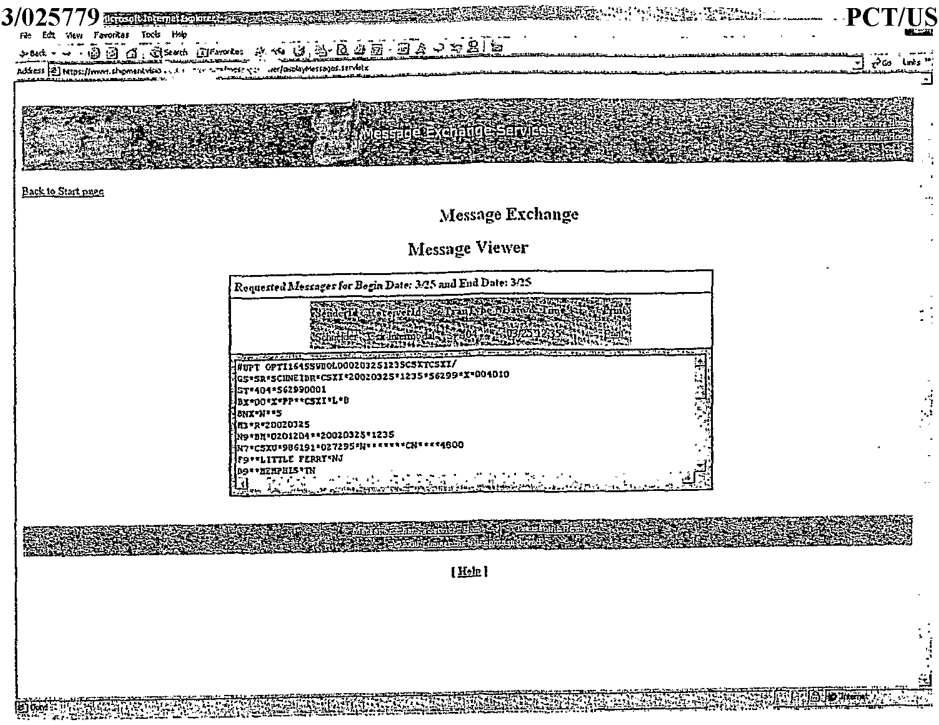
Equipment ID	Weight	Weight Unit	Type	Length
CSXU 985191	021295	Actual Net Weight	Container	4800

Cross References

Origin	Destination
LITTLE FERRY NJ	MEMPHIS TN

Names and Contacts
Shipper: SCHNEIDER NATIONAL OPTIMODAL
Address: 423 PENNSYLVANIA AVENUE
City: SOUTH KEARNY State/Province: NJ Postal Code: — Country: —
Consignee: SCHNEIDER NATIONAL OPTIMODAL
Address: 2765 PROFIT DRIVE
City: MEMPHIS State/Province: TN Postal Code: — Country: —
Party to Receive Freight Bill: SCHNEIDER NATIONAL OPTIMODAL
Address: 3101 S PACKERLAND DR
City: GREEN BAY State/Province: WI Postal Code:

Fig. 39


Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address https://www.shipmentviso ... /displayMessages.servlet
Go Links
Message Exchange Services
Done
Internet

Back to Start page

Message Exchange

Message Viewer

Requested Messages for Begin Date: 3/25 and End Date: 3/25

SenderId	ReceiverId	TranType	Date & Time	[illegible]	Print
Schneider	Csx Intermodal	404	03/25 1235	[illegible]	Print

```
#UPT OPTI1645SVBOL0002032S1235CSXTCSXI/
GS*SR*SCHNEIDR*CSXI*20020325*1235*56299*X*004010
ST*404*562990001
BX*00*X*PP**CSXI*L*B
BNX*N**5
N3*R*20020325
N9*BM*0201204**20020325*1235
N7*CSXU*986191*027295*N*******CN****4800
F9**LITTLE FERRY*NJ
D9**MEMPHIS*TN
```

[Help]

Fig. 40

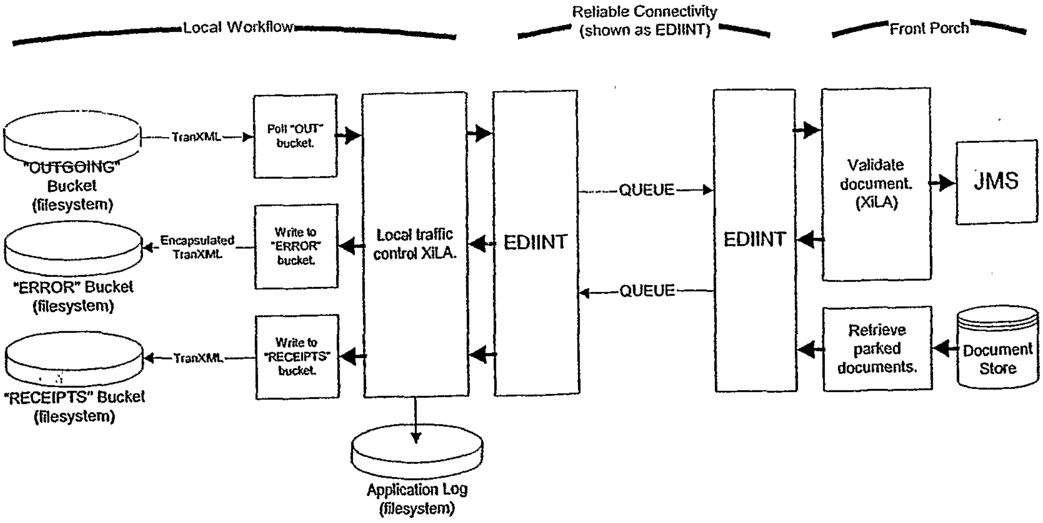

Local Workflow
Reliable Connectivity (shown as EDIINT)
Front Porch
"OUTGOING" Bucket (filesystem)
TranXML
Poll "OUT" bucket.
"ERROR" Bucket (filesystem)
Encapsulated TranXML
Write to "ERROR" bucket.
"RECEIPTS" Bucket (filesystem)
TranXML
Write to "RECEIPTS" bucket.
Local traffic control XiLA.
Application Log (filesystem)
EDIINT
QUEUE
QUEUE
EDIINT
Validate document. (XiLA)
JMS
Retrieve parked documents.
Document Store

Fig. 41

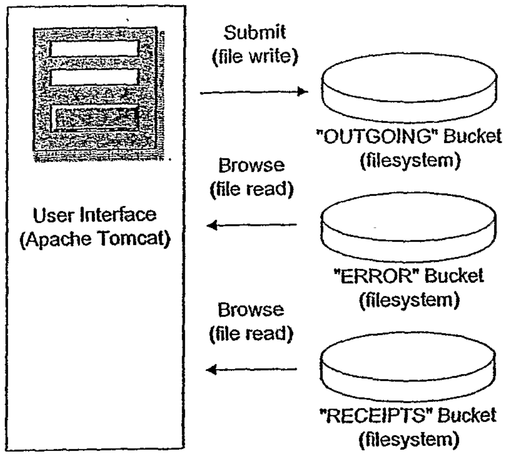

User Interface
(Apache Tomcat)
Submit
(file write)
"OUTGOING" Bucket
(filesystem)
Browse
(file read)
"ERROR" Bucket
(filesystem)
Browse
(file read)
"RECEIPTS" Bucket
(filesystem)

Fig. 42

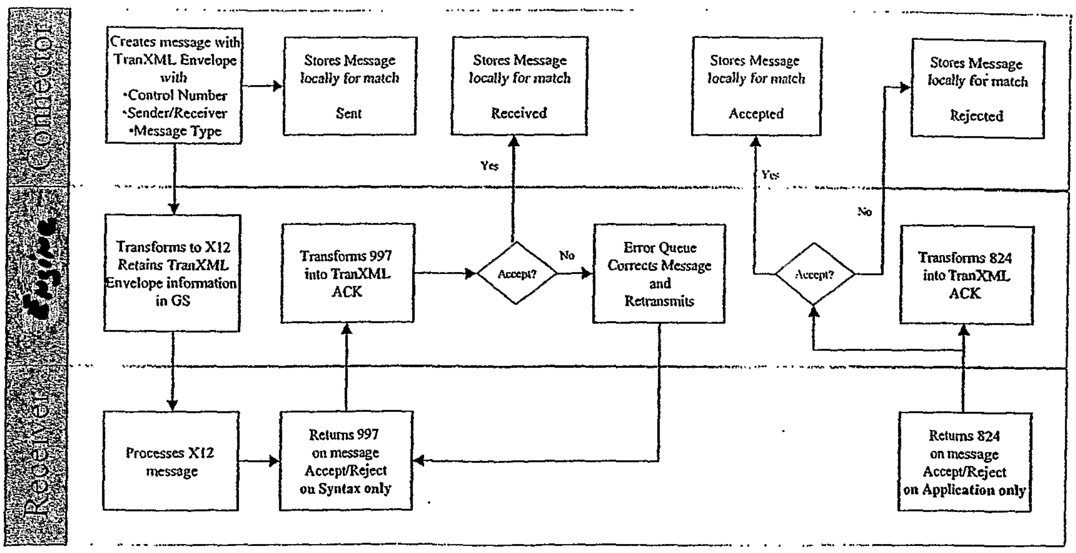

Connector
Engine
Receiver
Creates message with TranXML Envelope with
•Control Number
•Sender/Receiver
•Message Type
Stores Message locally for match
Sent
Stores Message locally for match
Received
Stores Message locally for match
Accepted
Stores Message locally for match
Rejected
Yes
Yes
No
Transforms to X12 Retains TranXML Envelope information in GS
Transforms 997 into TranXML ACK
Accept?
No
Error Queue Corrects Message and Retransmits
Accept?
Transforms 824 into TranXML ACK
Processes X12 message
Returns 997 on message Accept/Reject on Syntax only
Returns 824 on message Accept/Reject on Application only

Fig. 43

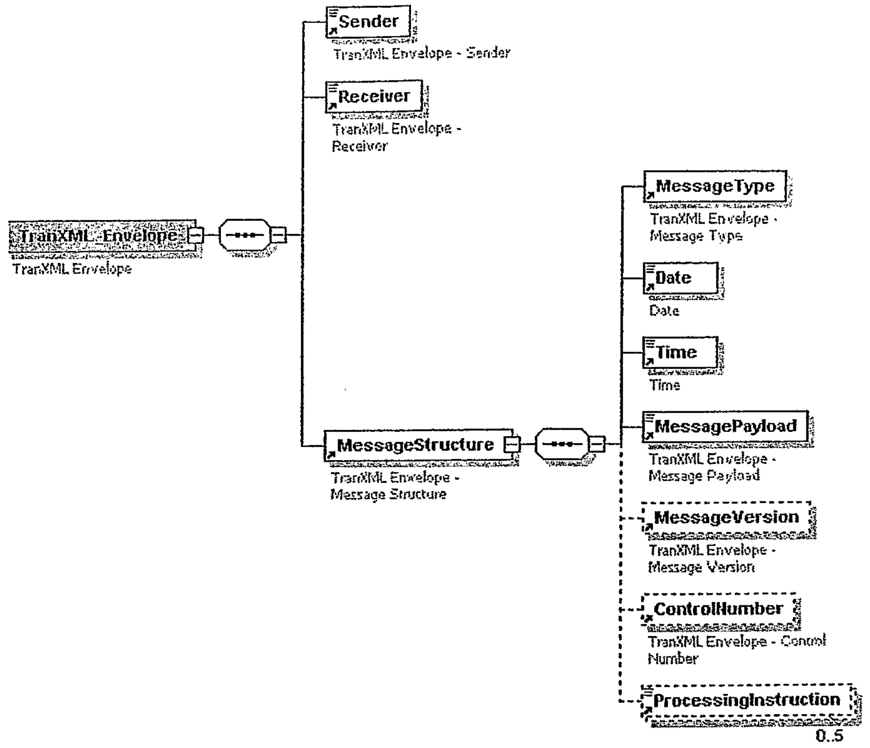
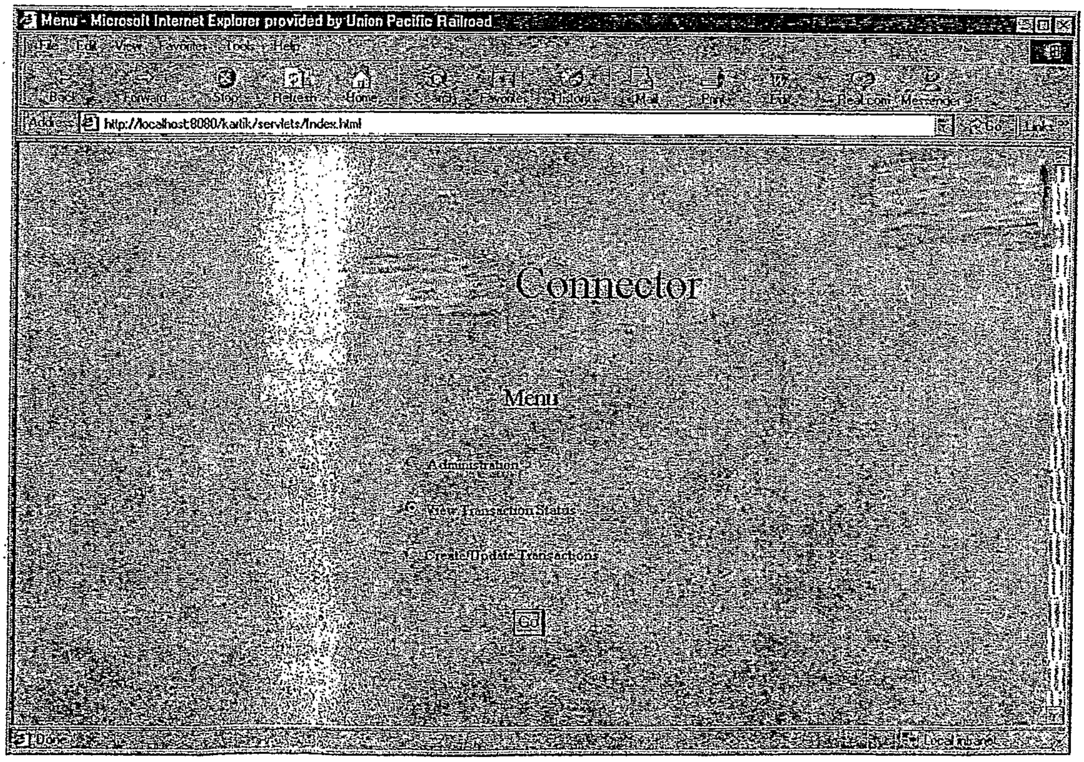

Sender
TranXML Envelope - Sender
Receiver
TranXML Envelope - Receiver
TranXML Envelope
TranXML Envelope
MessageType
TranXML Envelope - Message Type
Date
Date
Time
Time
MessageStructure
TranXML Envelope - Message Structure
MessagePayload
TranXML Envelope - Message Payload
MessageVersion
TranXML Envelope - Message Version
ControlNumber
TranXML Envelope - Control Number
ProcessingInstruction
0..5

Fig. 44



http://localhost:8080/kartik/servlets/Index.html

Connector

Menu

○ Administration

● View Transaction Status

○ Create/Update Transactions

Go

Fig. 45

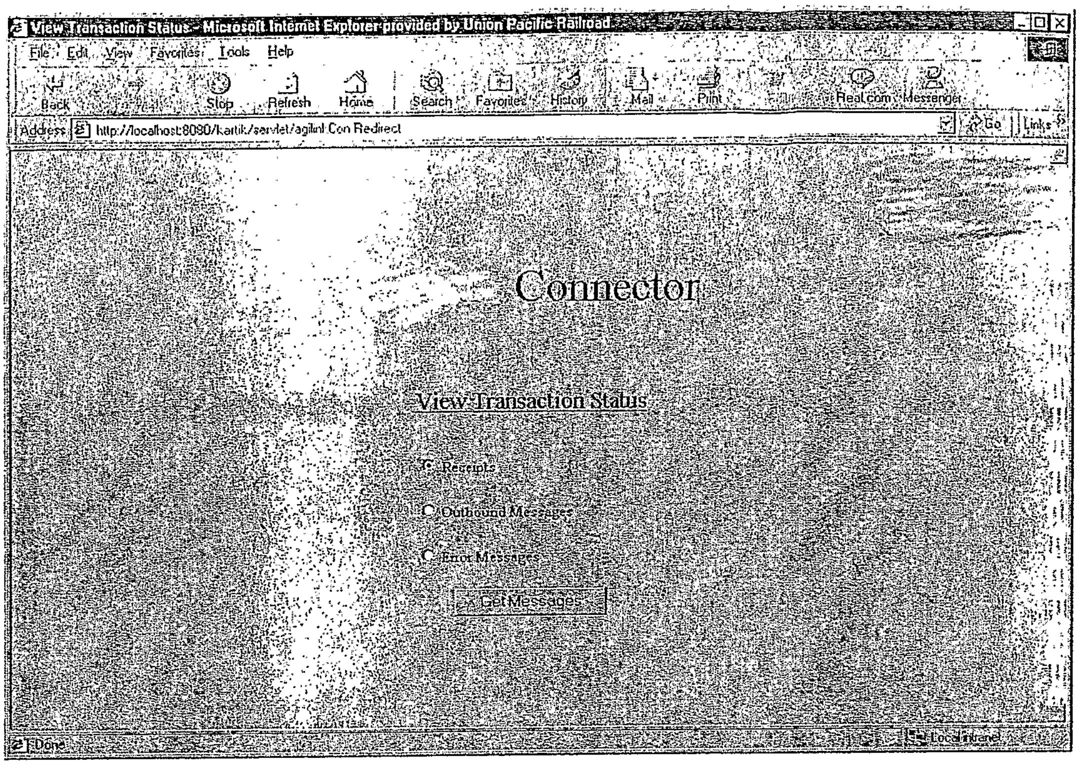
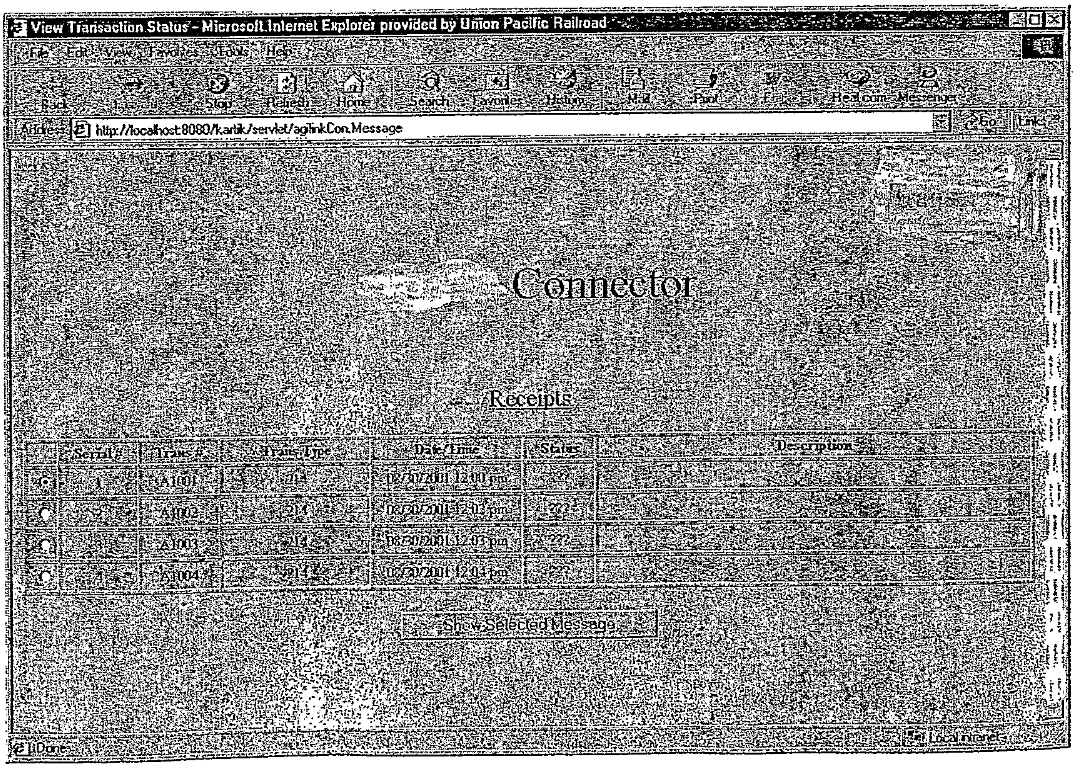

View Transaction Status - Microsoft Internet Explorer provided by Union Pacific Railroad
File Edit View Favorites Tools Help
Back Stop Refresh Home Search Favorites History Mail Print Real.com Messenger
Address http://localhost:8080/kartik/servlet/agilinkCon.Redirect
Go Links
Connector
View Transaction Status
● Receipts
○ Outbound Messages
○ Error Messages
Get Messages
Done
Local intranet

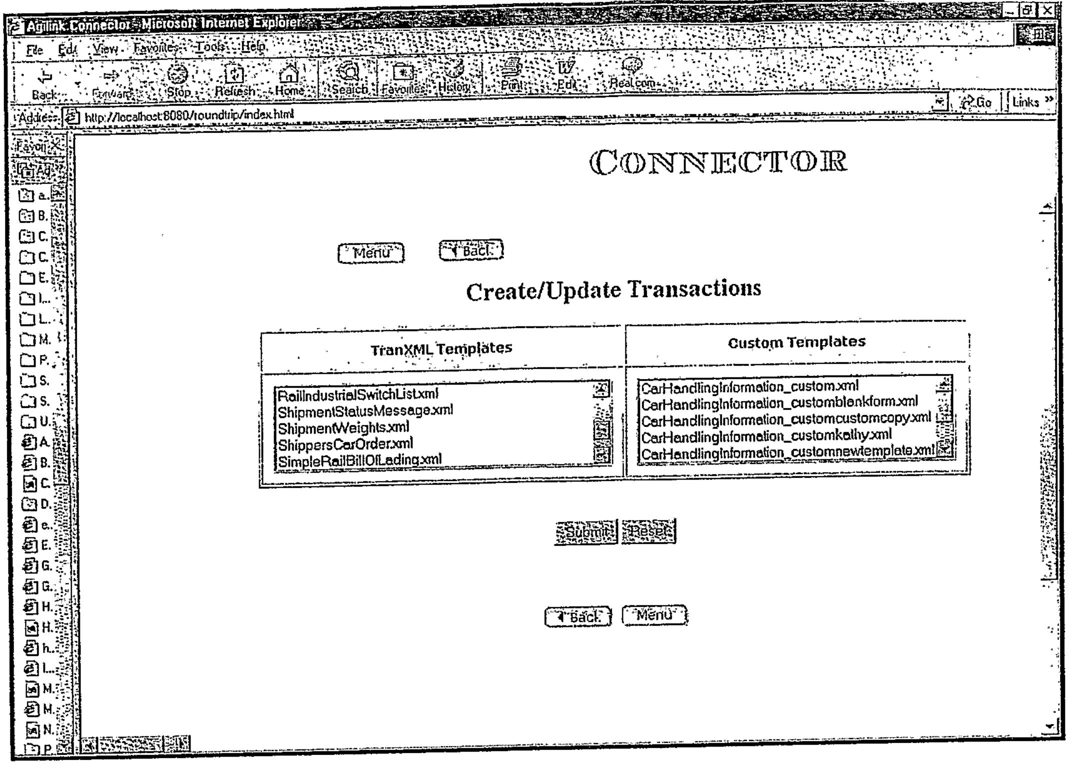
Fig. 46


View Transaction Status - Microsoft Internet Explorer provided by Union Pacific Railroad
File Edit View Favorites Tools Help
Back Stop Refresh Home Search Favorites History Mail Print Real.com Messenger
Address http://localhost:8080/kartik/servlet/agilinkCon.Message
Go Links
Connector
Receipts

	Serial #	Trans #	Trans Type	Date/Time	Status	Description
●	1	A1001	214	08/30/2001 12:00 pm	???	
○	2	A1002	214	08/30/2001 12:02 pm	???	
○	3	A1003	214	08/30/2001 12:03 pm	???	
○	4	A1004	214	08/30/2001 12:04 pm	???	

Show Selected Message
Done
Local intranet

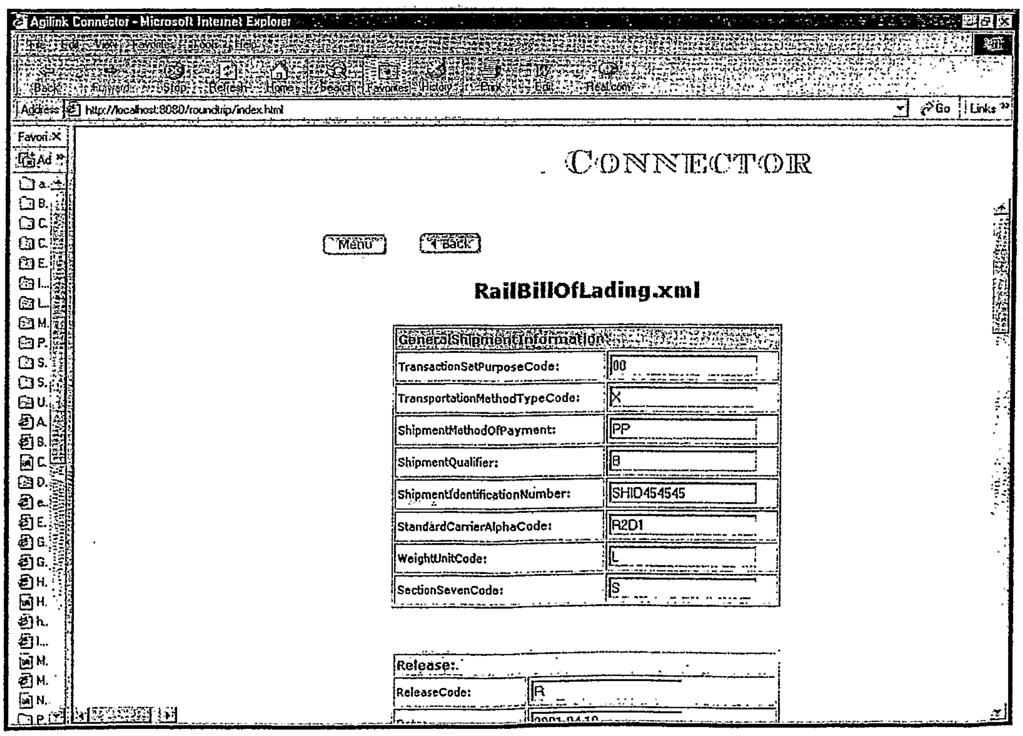
Fig. 47


Agilink Connector - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Print Edit Real.com
Address http://localhost:8080/roundtrip/index.html
Go Links
CONNECTOR
Menu
Back

Create/Update Transactions

TranXML Templates	Custom Templates
RailIndustrialSwitchList.xml ShipmentStatusMessage.xml ShipmentWeights.xml ShippersCarOrder.xml SimpleRailBillOfLading.xml	CarHandlingInformation_custom.xml CarHandlingInformation_customblankform.xml CarHandlingInformation_customcustomcopy.xml CarHandlingInformation_customkathy.xml CarHandlingInformation_customnewtemplate.xml

Fig. 48

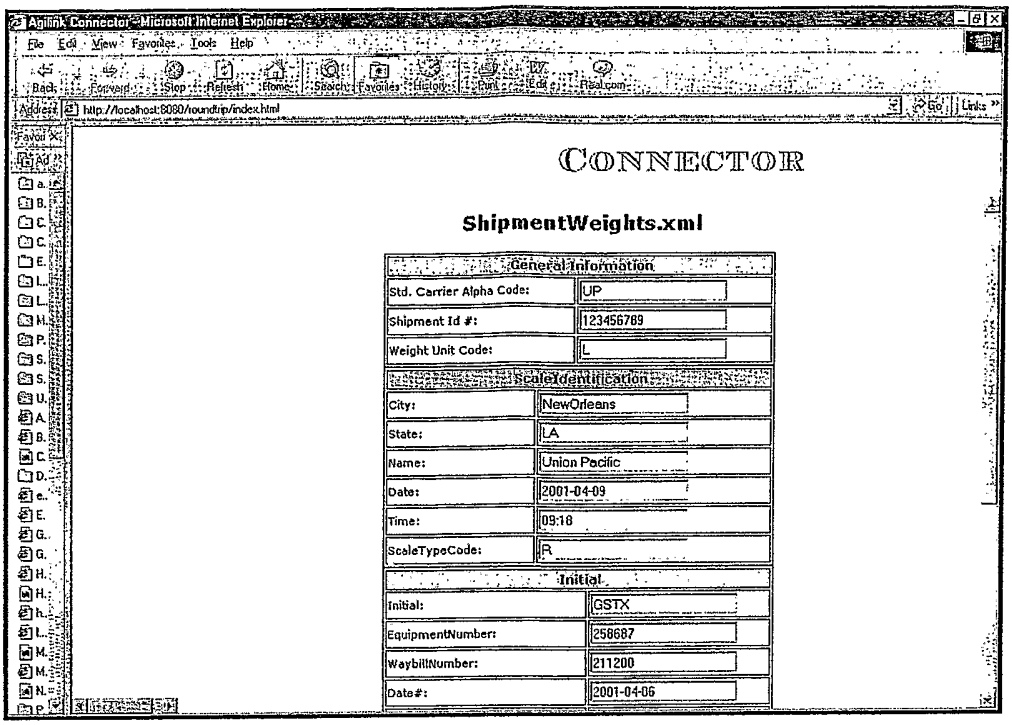

Agilink Connector - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Print Edit Real.com
Address http://localhost:8080/roundtrip/index.html
Go Links
CONNECTOR
Menu
Back

RailBillOfLading.xml

GeneralShipmentInformation:	
TransactionSetPurposeCode:	00
TransportationMethodTypeCode:	X
ShipmentMethodOfPayment:	PP
ShipmentQualifier:	B
ShipmentIdentificationNumber:	SHID454545
StandardCarrierAlphaCode:	R2D1
WeightUnitCode:	L
SectionSevenCode:	S

Release:	
ReleaseCode:	R

Fig. 49

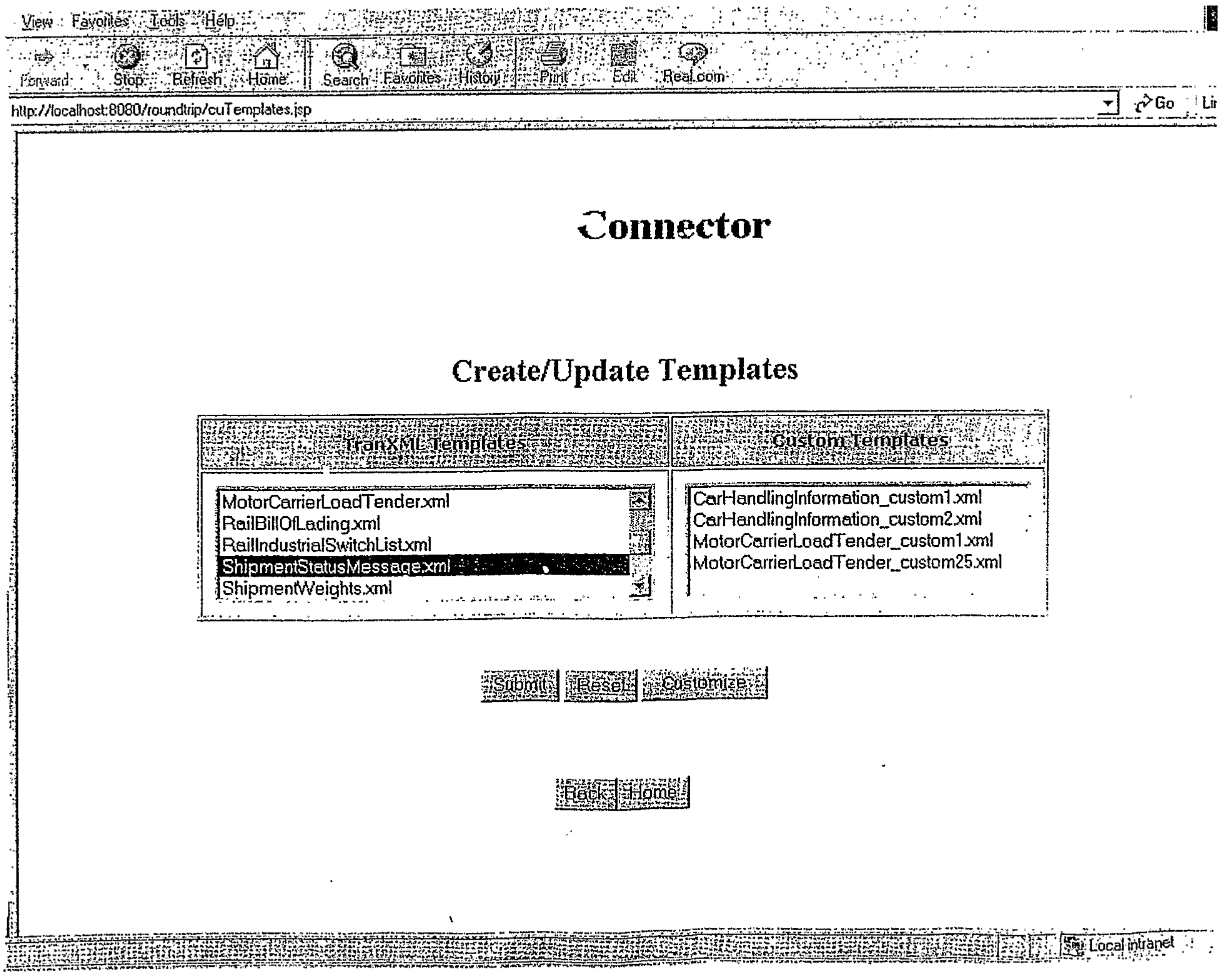

Agilink Connector - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Print Edit Real.com
Address http://localhost:8080/roundtrip/index.html
Go Links »
Favori ×
Ad

CONNECTOR

ShipmentWeights.xml

General Information	
Std. Carrier Alpha Code:	UP
Shipment Id #:	123456789
Weight Unit Code:	L
ScaleIdentification	
City:	NewOrleans
State:	LA
Name:	Union Pacific
Date:	2001-04-09
Time:	09:18
ScaleTypeCode:	R
Initial	
Initial:	GSTX
EquipmentNumber:	258687
WaybillNumber:	211200
Date#:	2001-04-06

Fig. 50

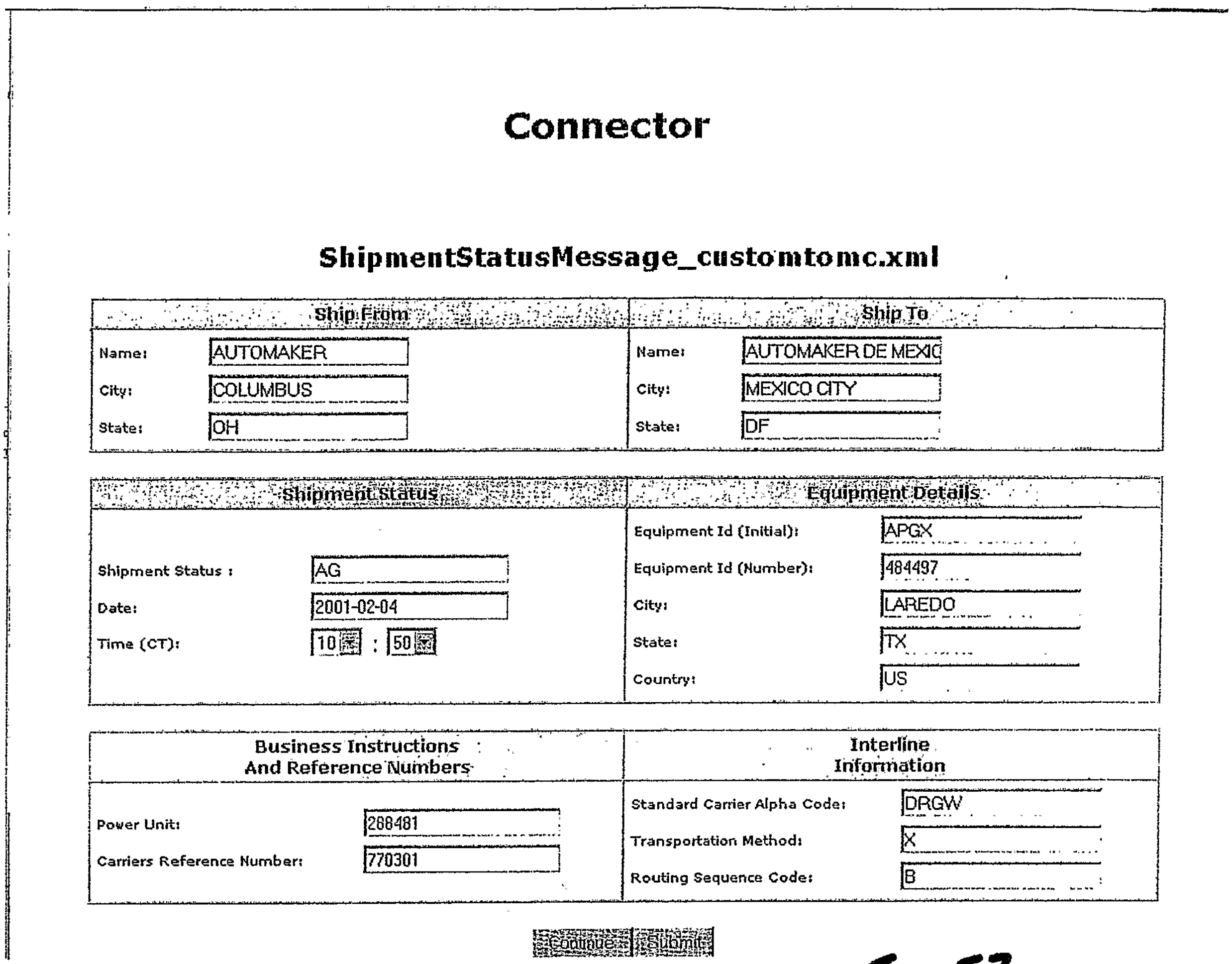

View Favorites Tools Help
Forward Stop Refresh Home Search Favorites History Print Edit Real.com
http://localhost:8080/roundtrip/cuTemplates.jsp
Go
Submit
Reset
Customize
Back
Home
Local intranet

Connector

Create/Update Templates

TranXML Templates	Custom Templates
MotorCarrierLoadTender.xml RailBillOfLading.xml RailIndustrialSwitchList.xml ShipmentStatusMessage.xml ShipmentWeights.xml	CarHandlingInformation_custom1.xml CarHandlingInformation_custom2.xml MotorCarrierLoadTender_custom1.xml MotorCarrierLoadTender_custom25.xml

Fig. 51



Connector

ShipmentStatusMessage_customtomc.xml

Ship From		Ship To	
Name:	AUTOMAKER	Name:	AUTOMAKER DE MEXIC
City:	COLUMBUS	City:	MEXICO CITY
State:	OH	State:	DF

Shipment Status		Equipment Details	
		Equipment Id (Initial):	APGX
Shipment Status :	AG	Equipment Id (Number):	484497
Date:	2001-02-04	City:	LAREDO
Time (CT):	10 : 50	State:	TX
		Country:	US

Business Instructions And Reference Numbers		Interline Information	
Power Unit:	268481	Standard Carrier Alpha Code:	DRGW
Carriers Reference Number:	770301	Transportation Method:	X
		Routing Sequence Code:	B

Fig. 52

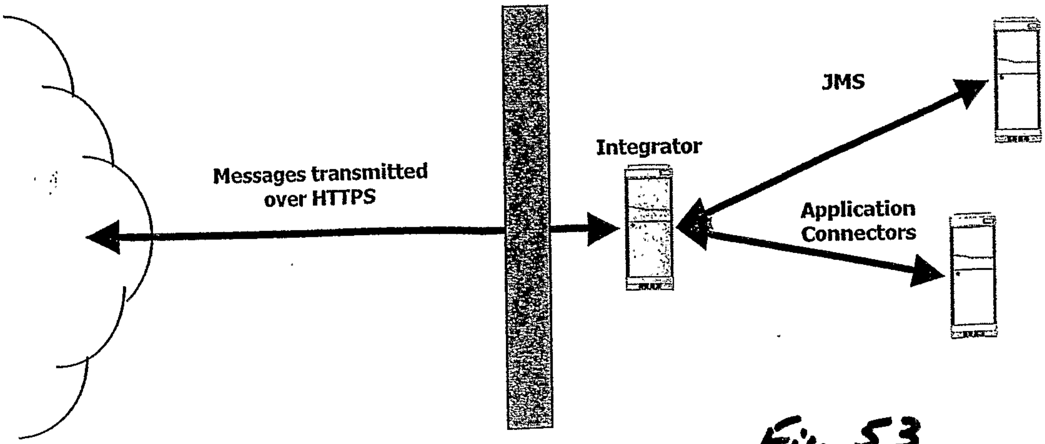


Fig. 53

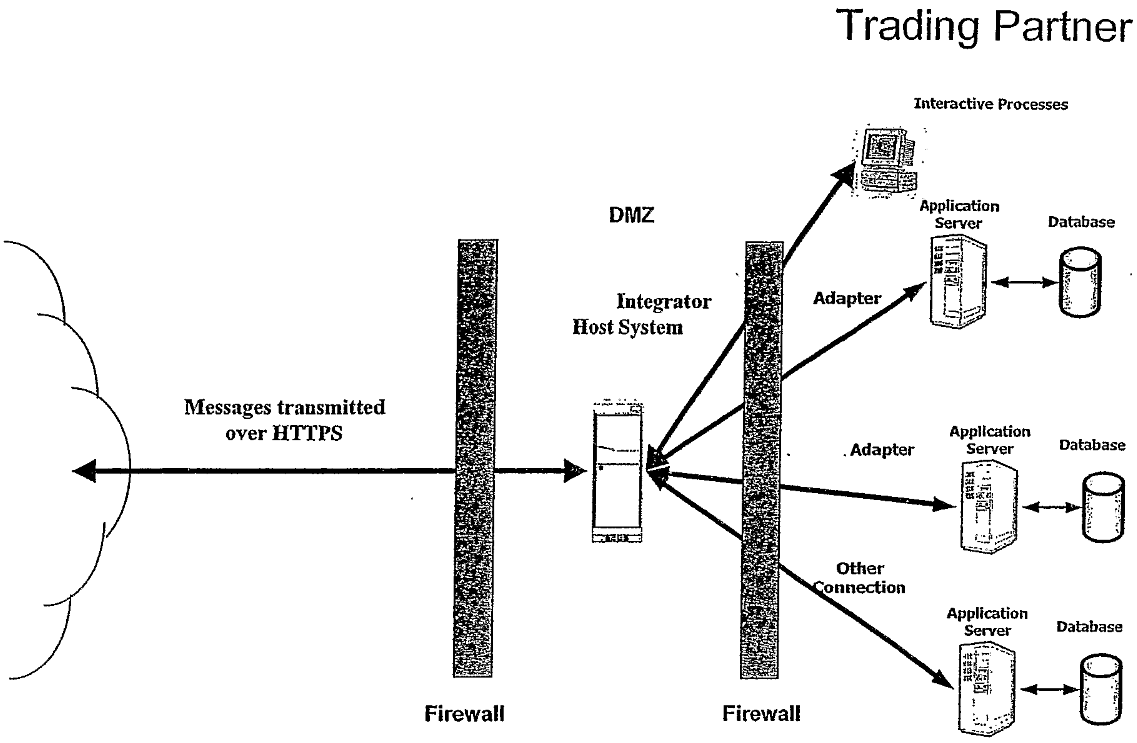

Trading Partner
Interactive Processes
DMZ
Application Server
Database
Adapter
Integrator Host System
Messages transmitted over HTTPS
. Adapter
Application Server
Database
Other Connection
Application Server
Database
Firewall
Firewall

Fig.54

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US02/29351

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) : GO6F 15/16
US CL : 709/201
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 709/201

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

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 6,144,988 A (KAPPEL) 07 November 2000 (07.11.2000), see column 2, line 64 to column 10, line 55.	1-42
Y	US 6,125,391 A (MELTZER et al.) 26 September 2000 (26.09.2000), see column 9, line 8 to column 32, line 20.	1-42
A	US 5,982,983 A (HUGHES) 09 November 1999, see the whole reference.	1-42

□ Further documents are listed in the continuation of Box C. □ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent published on or after the international filing date

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

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

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

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

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

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

"&" document member of the same patent family

Date of the actual completion of the international search 19 December 2002 (19.12.2002)	Date of mailing of the international search report 14 JAN 2003
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703)305-3230	Authorized officer Meng Ai An Telephone No. 703-305-3900