

A MODULE AND METHOD FOR TRACKING MESSAGES SENT BETWEEN DISPARATE BUSINESS APPLICATIONS AND HANDLING RELATED ERRORS

Field of the Invention

This invention relates to a module and method for tracking messages sent between disparate business applications and for handling errors that may occur during the transfer of the messages.

Summary of Prior Art

Existing enterprises have an increasing number of disparate business applications for different purposes, typically from different vendors. These systems may comprise e-commerce platforms, CRM systems, ERP systems, SFA systems, accounting systems, call-center systems, etc. This results in a massive requirement for integration in order to allow internal business processes to run fully automatically.

In addition to these internal requirements of a company, the requirements for integration with external business partners are growing, drastically pushed by growing legal requirements (e.g. Nota Fiscal) and the increasing pressure from large companies (e.g. Walmart) to carry out business transactions completely electronically. In existing enterprises, as more and more vendors, clients and/or customers are brought on board, data or information will have to be shared seamlessly between the enterprise and the various other different business applications. As an example, the enterprise may utilize an internal accounting business applications while a vendor may utilize a commercial accounting business application such as Oracle, SAP, etc. As expected, each of these business applications will store data in their own unique formats, have their individual primary key logics, their individual data customizations and different business logics.

Hence, in order for data to be shared seamlessly between these two disparate accounting systems, the internal accounting system will have to be integrated with the commercial accounting system. This may be done using various types of data-integration methods, for example, using an application integration technique, a legacy migration approach, a data source consolidation method, a master data consolidation approach, a server consolidation approach, or any other software based techniques. Regardless of the technique adopted, the initial stage of any data-integration between two disparate business applications typically involves the integration and organization of data. This is usually performed by individual coded logic with the help of a mapping tool. However, a mapping tool covers structural mapping only. In other words, it is targeted at the different field names and

the arrangement of information inside the message. But even here, there are limits and this often requires additional intervention through extra coding steps.

Furthermore, existing data-integration methods (with the exception of possibly project based development and message-based protocols like Electronic Data Interchange) also do not allow or provide for the shared data to be tracked by the business application that sent it or for notifications relating to the successful delivery/receipt of the data to be issued by the business application that received the data.

Existing integration platforms only provide rudimentary support for error handling whereby technical errors are typically manually controlled by a technical integration specialist while semantical errors have to be manually addressed by the integration consultant who will then have to develop individual solutions to address the individual errors. Overall, as the errors have to be addressed individually, this results in a lot of time being wasted and overall costs increasing beyond the original budget. As a result, these errors are typically ignored. When this happens, there will be a lack of transparency and/or information about any errors, e.g. semantic errors that may have occurred during the data exchange and as such, the users of the integration platforms will be unaware of the need for remedial measures to be taken.

For the above reasons, those skilled in the art are constantly striving to come up with a module and method that is capable of automatically generating data-delivery confirmations and end-to-end tracking of sent data messages between all integrated business applications. Further, the module and method would also be able to handle technical errors and semantical errors automatically or based on “on-demand retrials”, including the sending of alerts to registered users.

Summary of the Invention

The above and other problems are solved and an advance in the art is made by systems and methods provided by embodiments in accordance with the invention.

A first advantage of embodiments of modules and methods in accordance with the invention is that the invention is that once a data message has been delivered, a delivery confirmation may be generated and stored.

A second advantage of embodiments of modules and methods in accordance with the invention is that the invention allows for an automatic end-to-end tracking of data messages between integrated business applications.

A third advantage of embodiments of modules and methods in accordance with the invention is that any errors that occur during the transfer of data messages between integrated business applications will be automatically handled and logged via an error handling service.

A fourth advantage of embodiments of modules and methods in accordance with the invention is that any semantical errors that occur during the transfer of data message between integrated business applications will be automatically handled by an incomplete inbox service.

The above advantages are provided by embodiments of a module in accordance with the invention operating in the following manner.

According to a first aspect of the invention,

Brief Description of the Drawings

The above and other problems are solved by features and advantages of a system and method in accordance with the present invention described in the detailed description and shown in the following drawings.

Figure 1 illustrating a block diagram of a system for integrating information across disparate business applications in accordance with embodiments of the invention;

Figure 2 illustrating a block diagram representative of processing systems providing embodiments in accordance with embodiments of the invention;

Figure 3 illustrating a comparison between an exemplary message in the format of a first business application and the same message in the format of a second business application;

Figure 4 illustrating exemplary unique values that may be used as primary keys, secondary keys, links to other business objects, a determination result and the mapping of a customized value in the exemplary message in accordance with embodiments of the invention.

Figure 5 illustrates a process flow of the linking of a message that is in the format of a first business application to a message that has the format of a second business application in accordance with embodiments of the invention;

Figure 6 illustrating a process flow of the updating of an existing message that has the format of a second business application based on the content of a message that is in the format of a first business application in accordance with embodiments of the invention;

Figure 7 illustrating a process flow of the delivery confirmation process in accordance with embodiments of the invention;

Figure 8 illustrating a process flow of the end-to-end tracking service in accordance with embodiments of the invention;

Figure 9 illustrating a process flow of the error handling service in accordance with embodiments of the invention;

Figure 10 illustrating a process flow of the incomplete inbox service in accordance with embodiments of the invention;

Figure 11 illustrating a process flow of the integration of two disparate business applications whereby one system comprises a sender's business application and the other system comprises a receiver's business application in accordance with embodiments of the invention;

Figure 12 illustrating a process flow of the generation of a local primary key for the receiver's business application in accordance with a first embodiment of the invention;

Figure 13 illustrating a process flow of the generation of a local primary key for the receiver's business application in accordance with a second embodiment of the invention; and

Figure 14 illustrating a process flow of the generation of a local primary key for the receiver data management system in accordance with a third embodiment of the invention.

Detailed Description

This invention relates to a module and method for integrating information or data across disparate business applications. In particular, the module is configured to receive a message from a business application, enhance the message such that any integration logic may easily transform the enhanced message into a format that can be easily processed by another disparate business application. In embodiments of the invention, the processing of the message may comprise the retrieval of the message from the other business application or may comprise the insertion of the message into the format of the other business application.

The present invention will now be described in detail with reference to several embodiments thereof as illustrated in the accompanying drawings. In the following description, numerous specific features are set forth in order to provide a thorough understanding of the embodiments of the present invention. It will be apparent, however, to one skilled in the art, that embodiments may be realised without some or all of the specific features. Such

embodiments should also fall within the scope of the current invention. Further, certain process steps and/or structures in the following may not have been described in detail and the reader will be referred to a corresponding citation so as to not obscure the present invention unnecessarily.

Further, one skilled in the art will recognize that many functional units in this description have been labelled as modules throughout the specification. The person skilled in the art will also recognize that a module may be implemented as circuits, logic chips or any sort of discrete component. Still further, one skilled in the art will also recognize that a module may be implemented in software which may then be executed by a variety of processors. In embodiments of the invention, a module may also comprise computer instructions or executable code that may instruct a computer processor to carry out a sequence of events based on instructions received. The choice of the implementation of the modules is left as a design choice to a person skilled in the art and does not limit the scope of this invention in any way.

Figure 1 illustrates a block diagram of the system for the integration of disparate business applications in accordance with embodiments of the invention. System 100 comprises of business application 105, business application 115 and integration module 110. It should be noted that business applications 105 and 115 both comprise disparate business application which are sitting on top or installed on top of database systems whereby each business application stores data according to their own proprietary format, have their data customized in their own way, have business objects tailored to their unique methods and process data according to their own proprietary rules. Business applications 105 and 115 provide their own logic, and may be exposed using various User Interfaces (UIs) and Application Programming Interfaces APIs such as, but are not limited to, SAP S/4, Oracle E-Business Suite, salesforce.com, QuickBooks, Shopify, Amazon, Magento, Xero , and all other types/forms of business applications. It should be noted that the designations used to describe the business applications throughout this description are not to be construed as limiting the invention to specific numbers and/or types of business applications and these designations are selected purely for the convenience of the reader.

As for integration module 110, this module comprises of various components and a processing system for implementing embodiments of the invention. Figure 2 illustrates an exemplary processing system 200 that may be provided within module 110. One skilled in the art will recognize that the exact configuration of processing system may be different, the exact configuration may vary and Figure 2 is provided by way of example only.

In embodiments of the invention, each of the modules may comprise controller 201 and user interface 202. User interface 202 is arranged to enable manual interactions between a user and each of these modules as required and for this purpose includes the input/output components required for the user to enter instructions to provide updates to each of these modules. A person skilled in the art will recognize that components of user interface 202 may vary from embodiment to embodiment but will typically include one or more of display 240, keyboard 235 and track-pad 236.

Controller 201 is in data communication with user interface 202 via bus 215 and includes memory 220, processor 205 mounted on a circuit board that processes instructions and data for performing the method of this embodiment, an operating system 206, an input/output (I/O) interface 230 for communicating with user interface 202 and a communications interface, in this embodiment in the form of a network card 250. Network card 250 may, for example, be utilized to send data from these modules via a wired or wireless network to other processing devices or to receive data via the wired or wireless network. Wireless networks that may be utilized by network card 250 include, but are not limited to, Wireless-Fidelity (Wi-Fi), Bluetooth, Near Field Communication (NFC), cellular networks, satellite networks, telecommunication networks, Wide Area Networks (WAN) and etc.

Memory 220 and operating system 206 are in data communication with CPU 205 via bus 210. The memory components include both volatile and non-volatile memory and more than one of each type of memory, including Random Access Memory (RAM) 220, Read Only Memory (ROM) 225 and a mass storage device 245, the last comprising one or more solid-state drives (SSDs). Memory 220 also includes secure storage 246 for securely storing secret keys, or private keys. One skilled in the art will recognize that the memory components described above comprise non-transitory computer-readable media and shall be taken to comprise all computer-readable media except for a transitory, propagating signal. Typically, the instructions are stored as program code in the memory components but can also be hardwired. Memory 220 may include a kernel and/or programming modules such as a software application that may be stored in either volatile or non-volatile memory.

Herein the term “processor” is used to refer generically to any device or component that can process such instructions and may include: a microprocessor, microcontroller, programmable logic device or other computational device. That is, processor 205 may be provided by any suitable logic circuitry for receiving inputs, processing them in accordance with instructions stored in memory and generating outputs (for example to the memory components or on display 240). In this embodiment, processor 205 may be a single core or multi-core processor with memory addressable space. In one example, processor 205 may

be multi-core, comprising—for example—an 8 core CPU. In another example, it could be a cluster of CPU cores operating in parallel to accelerate computations.

With reference to Figure 1, when business application 105 generates a message 120 which is in the format of business application 105, message 120 will be retrieved and processed into a suitable format for business application 115 by integration module 110 as module 110 would be constantly monitoring application 105 for changes such as the generation of new messages and/or module 110 may be configured to monitor application 105 for such changes based on a predetermined schedule using a specific API. If message 120 were to be directly sent to business application 115 by module 110, business application 115 would not be able to process message 120 as its format would not be compatible with the format utilized by business application 115.

An exemplary format of a message generated by business application 105 is illustrated as message-format 305 in Figure 3 while the format adopted by business application 115 is illustrated as message-format 310. As can be seen from these two message-formats, they both utilize different data structures, they tailor data differently, and they assign and handle primary keys in their own unique styles. Therefore, it is widely accepted that it would be a challenge to generate message-format 310 directly from message-format 305.

One skilled in the art will recognize that a primary key/ local primary key is defined as a unique identifier that is associated with a particular record. Its main use is to uniquely identify a record or specific content within a message or a table.

Figure 4 illustrates local primary keys 410a-d as found in invoice 400 (i.e. message format 310 in Figure 3). In this example, although the message is shown to be in the format of an invoice, one skilled in the art will recognize that the message may comprise of any other types of documents without departing from the invention.

It can be seen that in invoice 400, primary key 410a comprises (A8B226576C); primary key 410b comprises (4789990.A); primary key 410c comprises (itm013668); and primary key 410d comprises (itm003445).

In this example, the function of primary key 410a is to provide a document identifier to invoice 400 such that this invoice 400 may be identified simply by calling primary key 410a. Similarly, primary key 410b functions as a link to an identity of a customer of invoice 400, e.g. location 425a and address 425b, such that when primary key 410b is called, it will return location 425a and address 425b. As for primary key 410c, in this example, it acts as a link to the details of a particular item, e.g. item details 430; and similarly, primary key 410d acts as a

link to the details of another item, e.g. item details 435. Hence, when primary keys 410c and 410d are called, it will return the details of items 430 and 435 respectively.

In other embodiments of the invention, 410b, 410c and 410d may be identified as secondary keys whereby each of these secondary keys are associated with individual business objects. Reference 415 illustrates a mapping of a customization value while 425a and 425b represent enhancements that were added to the message.

The 'action=update' feature 450, illustrates the determination result - which is the result of the invention's logic to detect if the message is already available in receiver system (update) or if the message is not available in the receiver system (insert).

Figure 5 illustrates a process flow showing the linking of message 501 that is in the format of first business application 105 to a message (not shown) that has the format of second business application 115 in accordance with embodiments of the invention.

For all messages, that are sent from application 105 to application 115, all the information about the various local primary keys (and their functions) will be first generated and stored within module 110. With the next appearance of the already passed messages, sent from application 105 to application 115, these stored information will be used.

In operation, message 501 will be initially intercepted and received by integration module 110. For this example, as the intention is to link message 501 to a message in application 115, module 110 will retrieve a local primary key, L1-Key (i.e. "43557C0A8B2269715"), associated with the document identity of message 501 from message 501. It should be noted that at this initial stage, the data in message 501 only comprises the local primary key L1-Key. Module 110 then determines at step 552 whether the retrieved local primary key L1-Key has been assigned with a global key GKey and as a result has previously been paired with a local primary key associated with a message of second business application 115.

In this example, at step 552, as the local primary key L1-Key has not been assigned with a global GKey (and has not been paired with a local primary key associated with a message of the second business application), module 110 will proceed to generate a new GKey and will link this new GKey to the L1-Key. Module 110 will also retrieve the unique identifier (SysId1) of the Business Application 105 and link this SysId1 to the GKey-L1-Key pair resulting in the entry GKey-SysId1-L1-Key being entered into module 110. All this information is then appended to message 501 at this step. This enhances the information contained within message 501. In this embodiment of the invention, module 110 proceeds to

generate, at step 554, a local primary key L2-Key (that is associated with a message of the second business application) for local primary key L1-Key and will store the key, that is paired with the previously generated GKey into database 505. The enhanced message 501 is then provided to any integration logic to prepare and send the message to application 115.

In an embodiment of the invention, the local primary key L2-Key may be generated directly by module 110. In this embodiment the local primary key L2-Key may be generated based on a Global Unique Identifier (GUID); based on a random number of a particular format and length; based on a primary key generation algorithm followed by a number range having a specific format and length; or based on a primary key generation algorithm followed by a defined logic.

In another embodiment of the invention, the local primary key L2-Key may be generated by business application 115 and then subsequently provided to module 110. In this embodiment, module 110 is only made aware of the local primary key L2-Key that is to be paired with the local primary key L1-Key only after it has received it from business application 115. As such, in this embodiment, the local primary key L2-Key may be determined by a “response” approach, by “a post call with timestamp” approach or by a “post call with filter” approach.

Under the “response” approach, module 110 will perform a synchronous insert-API call of message 501 (transformed to 115 format) to application 115. Upon receiving this API call, system 115 will generate an appropriate local primary key L2-Key and subsequently provide this L2-Key to module 110 as part of the response to the API call. Upon receiving the response from application 115, module 110 is then able to identify the L2-Key inside the response data, based on definitions, describing from where to pick the primary key in the response data of this document type.

When the “post call with timestamp” approach is adopted, module 110 will similarly perform an insert-API call of message 501 to system 115. However, under this approach, application 115 will not respond to such an API call with a generated primary key. After a predetermined period of time has lapsed, module 110 will perform another API call to application 115 to retrieve all document types along with their corresponding documents that were attempted to be “inserted” into application 115 over a particular period. This means that if this particular period were to be set to the period/time-range that the original insert-API call of message 501 was sent to application 115, this would result in application 115 retrieving all the documents that are of the same document-type as message 501. The bundle of documents having the same document-type as message 501 would then be provided to module 110. Upon receiving these documents, module 110 will then identify the recently

inserted document and will pick the corresponding primary key L2-Key from this document, using definitions which will describe where the primary key is to be obtained from this document type.

Under the “post call with filter” approach, as application 115 will not respond to an insert-API call from module 110 with a generated primary key and does not support the retrieval of all document types by timestamps along with their corresponding documents that were attempted to be “inserted” into application 115 over a particular period. Module 110 will have to perform another API call to application 115 to retrieve all documents that are of the same document type as message 501, filtered to before inserted identifier fields (e.g. name, total amount, last updated, etc. (based on definitions for this system)). The documents having the same document-type as message 501, filtered by identifier fields would then be provided to module 110. Upon receiving these documents, module 110 will then identify the recently inserted document and will pick the corresponding primary key L2-Key from this document, using definitions which will describe where the primary key is to be obtained from this document type.

One skilled in the art will recognize that other methods and/or techniques may be used at step 554 to generate a suitable local primary key L2-Key for local primary key L1-Key and that the generation of the L2-Key is not specifically limited to the approaches and/or embodiments described above.

Once a local primary key L2-Key associated with a message of the second database has been generated at step 554, module 110 then proceeds to add a new entry into database 505, pairing the already available GKey (which is paired with L1-Key) with the new L2-Key. At step 554, module 110 then retrieves the unique identifier (SysId2) (which is used to identify Business Applications used by system 115) and will link this with the L2-Key. At this stage, it is useful to note that the entry for the primary key of the incoming message (L1-Key-SysId1-GKey) has been previously entered into module 110 and now, an entry for the primary key of the inserted message (L2-Key-SysId2-GKey) is created in database 505, which may be provided within module 110 or may be located in a remote location and communicatively connected to module 110. All this information is then appended to message 501 at step 558. This enhances the information contained within message 501 for further processing.

It should be noted that although the description above only discloses the appending of a single set of local primary keys L1-Key and L2-Key, the retrieved system identities and the assigned GKey, one skilled in the art will recognize that two or more such sets (comprising of local primary keys, the associated retrieved system identities and the assigned global keys GKeys) may be generated and processed as described above. The above steps ensure that

the pairing of all the relevant local primary keys in message 501 (which is in the format of system 105) would be performed by module 110 before appended message 501 (which is enriched with all the sets of information) are provided to an integration logic for further processing into the appropriate format for business application 115. With the use of the key entry format (GKey-SysId-LKey), any number of systems may be logically linked.

Figure 6 illustrates a process flow showing the updating of the content in message 601 (that is in the format of first business application 105) to the content in a corresponding message (not shown) that has the format of second business application 115 in accordance with embodiments of the invention.

In operation, message 601 will be initially intercepted and received by integration module 110. For this example, as the intention is to update the contents of message 601 to contents in a corresponding message in application 115, module 110 will retrieve a local primary key, L1-Key (i.e. "43557C0A8B2269715"), associated with the document identity of message 601 from message 601.

Module 110 then determines at step 652 whether the retrieved local primary key L1-Key has been assigned with a global key GKey. In this example, as the local primary key L1-Key has been assigned with a global Gkey, i.e. GKey '2000' and its SysID1 "0010000101", module 110 then proceeds to identify the local primary key L2-Key via its SysID2 that has been previously paired with the local primary key L1-Key via the GKey '2000'. In this example, the paired local primary key L2-Key would comprise "A8B226576C" and SysID2 is "0010000102". It is useful to note that in this example, the L2-Key would be the primary key that is associated with the document identity of a message of the second business application.

Set 656 is then appended by module 110 to message 601 at step 658. This enhances the information contained within message 601.

Integration logic (not shown) will leverage this information to translate the message into the format and conventions, required by business application 115. Module 110 then provides the transformed message 601 to business application 115. Upon receiving the transformed message 601, application 115 may then proceed without any further logic related to integration as the message is completely in the proper format of application system 115. It should be noted that although the description above only discloses the appending of a single set of local primary keys L1-Key and L2-Key, the retrieved system identities and the assigned GKey, one skilled in the art will recognize that two or more such sets (comprising of local primary keys, the associated retrieved system identities and the assigned global keys GKeys) may be generated and appended to the message, sent to any integration logic, and processed

accordingly, afterwards handover to application 115 so that multiple contents may be mapped simultaneously.

In yet another embodiment of the invention, if module 110 determines that certain information is missing from a message that is sent from system 105 to system 115 (e.g. only business partner primary key (43557C0A8B2269715) is in the message, but address of business partner is missing, but is needed to handover to system 115), module 110 may obtain this missing information by an API call to the system 105 based on the business-partner-primary key. Module 110 obtains the information to retrieve the data using the primary key and the primary key may be obtained based on definitions that inform module 110 where the primary key is to be found in the message and from the definitions of the API. The required data is then retrieved and added in a special section in the appended message.

Figure 7 illustrates a process flow that shows a message delivery confirmation process in accordance with embodiments of the invention. The delivery confirmation process may take place during the insertion/linking of a new message by module 110 (as illustrated by the process flow in Figure 5) or during the updating of a message by module 110 (as illustrated by the process flow in Figure 6).

In an embodiment of the invention, in response to a message insertion request or a message updating request 702 by module 110 (i.e. the process illustrated in Figure 5 or 6 respectively), in addition to the usual response to the insertion or updating request (that passes through an integration logic module), application 115 will also generate a response 703 indicating that the message has been received and accepted. The response may then be used to validate the delivery status of the message, i.e. whether it was delivered or not. In particular, contained within module 110 will be definitions that will allow module 110 to determine from the API response whether the transfer to application 115 was successful or not. For example, the definitions could contain instructions for module 110 to monitor the response for the existence of a certain field having a specific value (e.g. //status='ok') or for a negative determination response (field 'soapfault' exists) or any other such determination. The response may also include the local primary keys (LKeys) linked to or appended to the message that was received by application 115 (i.e. for an update process) or new local primary keys linked to the message received by application 115 (i.e. for an insert process) and, the global key associated with the message. The insertion request may be in the form of an API call as discussed in the previous sections. The response is then stored by module 110 in delivery confirmation log 706 and all this takes place in the background through the use of generic adapters provided within module 110.

Conversely, once the insertion/updating process has been completed, module 110 will then generate, in the background through the use of generic adapters, an additional retrieval call 704 which is then sent to application 115. Explicit get call 704 comprises a function, which includes the local primary keys associated with the message that was supposed to be received by application 115. When get call 704 is executed by application 115, application 115 will determine whether a recently sent message (as defined in call 704) is available in application 115 (i.e. for an insert process) or will determine whether data contained in the recently sent message is found in application 115 (i.e. for an update process). Additionally, it will also determine whether the latest message that has been updated in application 115 corresponds to the time of the previous performed API call (i.e. the data transfer). Call 704 utilizes the local primary keys linked to the recently sent message to cause application 115 to search through messages recently received by application 115 to determine if the required message has been received by application 115. The outcome of this is then sent from application 115 to module 110 as response 705. All this information is then stored in delivery confirmation log 706 by module 110 and this takes place in the background of module 110. It should be noted that log 706 may comprise, but is not limited to, a database stored within module 110, a remote server or a cloud data centre.

Once log 706 has been updated, a Web User Interface may be used to determine if sent messages have been successfully received and accepted by any business applications. This can be done as the global and/or local primary keys of all linked messages and their corresponding business applications are known at this stage. In summary, the global keys associated with the receiving business application may be used to retrieve local primary keys linked with the message. These retrieved local primary keys are then used to determine if the sent messages (that contain the retrieved local primary keys) have been successfully received and accepted. If such messages (that contain the retrieved local primary keys) can't be found in log 706, this implies that the messages were not received and/or accepted by the intended business application. In other embodiments of the invention, the "Delivery Confirmation" process may comprise the step of providing a set of APIs to external parties so that these external parties may gain access to log 706 to determine if sent messages have been successfully received and accepted. These API's would also contain the global keys associated with the receiving business application so that the APIs may be used to retrieve local primary keys linked with the message and in turn determine the delivery status of the sent messages.

In another embodiment of the invention, if a message that should have been delivered is not found in log 706, an "End-to-End Tracking" service may be used to address this. Figure 8 illustrates the process flow of the end-to-end tracking service in accordance with

embodiments of the invention. The main purpose of this service is to track the message as it is sent and received by the various business applications.

As the message is sent from one business application to the next, each connection point 802 (comprising an API call from the sender to the receiving business application) is logged into tracking table 806 by module 110. The logging of the connection points or API calls are not executed directly by the integration logic but by module 110. In other words, each time an API call is carried out to request a message insertion or update, module 110 will also at the same time initiate a call to a generic adapter provided within module 110 to activate the logging of the connection points. When activated, connection points with the current timestamp, application system number, the action and the local primary key involved will all be logged into tracking table 806.

In embodiments of the invention, the end-to-end tracking service is enabled through the use of a semantic key (SKey). The SKey of a business application represents the logical key of the data (e.g. the order number 2022/1556) as compared to the local primary key (LKey) which represents the technical key (e.g. a GUID da3f3f26-6097-4201-a9fe-ccefca291468) associated with specific data points in the message. Typically, LKeys are not visible on the user interface of the business application instead, only the semantic keys SKeys are displayed and are identifiable by all disparate business applications.

In operation, in order to ensure that an end user may query the run of a specific job (2022/1556) through all connected business applications, during the key mapping operations (as described with reference to Figures 5 and 6), in addition to the generation and/or use of the LKey, GKey, and the System ID, semantic keys (SKeys) are also generated and/or determined by module 110 for the message. The SKeys are then linked to its corresponding (L1-Key-SysId1-GKey) / (L2-Key-SysId2-GKey) entries, similar to how the SysIds were assigned. The linked semantic keys are then stored in the extended (already existing) key mapping table. By doing so, a link is established between the system-specific / business application-specific SKeys to the LKeys and its corresponding records.

When SKeys are generated and linked during the key mapping operations, the delivery confirmation log 706 may be extended to include an additional timestamp field to record the exact time of each action (e.g. delivery of the message or receipt of the message) that occurs in module 110. As such, when a query is made for the end-to-end tracking of a message, the SKeys of the message may be leveraged to access the stored information in the key mapping table and the delivery confirmation log. For example, once the SKey/SysID of a message is obtained, the corresponding LKey may then be determined from the Key Mapping table along

with the associated GKey. Once the GKey has been obtained, all the LKeys associated with the GKey may then be determined as well.

The determined LKeys may then be used with the delivery confirmation log 706 to identify all entries in the respective disparate business applications associated with these LKeys. As the entries in log 706 are sorted by their timestamps, through the use of the timestamped entries, the entire transfer of the message may then be effectively tracked and the delivery status of the messages may then be validated.

It should be noted that presently, the diverse identities of the data used by the various business applications prevents the possibility of the end-to-end tracking of messages from one business application to the next.

This problem has been addressed through the use of the key linkages described in the previous sections (i.e. the key-mapping service) together with the delivery confirmation service and use of the semantic keys (SKeys) which allows for an effective and efficient end-to-end tracking service to be implemented. It should be noted that these definitions may all be implemented by generic services and adapters.

Web User Interface or API may be used to gain access to the content within to track any Business Object / Data Message using the global key / local key linked to the object / message.

During the integration process, technical errors may occur due to various problems such as network errors, loss of connectivity, hardware limitations (memory, CPU, disk space, etc.) and this may jeopardize the integration process. To address this problem, an error handling module is provided within module 110 and this module is configured to perform an error handling service. In embodiments of the invention, the error handling module is configured to run in the background of module 110 whereby it is also configured to monitor all integration processes that take place through module 110. Once such errors have been identified, the error handling module then pauses the integration process, issues a queue indicator which functions like a virtual bookmark, and proceeds to resolve the error based on predefined solutions associated with the identified error. Among some of the solutions that may be performed by the error handling module include, but are not limited to, the immediate reactivation of the integration process, the reactivation of the integration process after a predefined amount of time has passed, the removal of the affected message from the integration process and flagging the message for further processing, prevent the sending of the affected message and etc.

In embodiments of the invention, if a technical error occurs during the processing of a message, this process is terminated by an 'Exception' (i.e. an interrupt) which may be generated by the processor itself or the error handling module. It should be noted that integration processes are normally triggered by a series of messages that are queued in a queuing system. The integration process control of the error handling module immediately detects the occurred error when an 'Exception' occurs and the module then ensures that the message that triggered the exception is rolled back into the inbound queue using the queue indicator associated with the message and the integration process is temporarily deactivated whereby depending on the predetermined definitions of error actions, the integration process may then be activated again.

At the same time, the error handling module will identify the error based on the thrown error message. If the error message is known to the error handling module, more specific error handling actions can then be initiated. Each time the error is detected, an error counter associated with the specific error will increase.

If the cause of the error message is not known, the error along with the message that triggered the error will be inserted into the error database with a newly initialized error counter (i.e. count = 1) and with the default error actions. The user of the error handling module may then analyse the frequency of particular errors, specify individual error actions which can be automatically processed or which may be handed over to other users of other business applications (to address the possibility that it is the inconsistencies in the other application's data that is causing the error). Through the analysis of the errors, weak points in the integration processes may then be determined. These flaws may then be addressed and the integration processes may then be optimized.

The error handling service performed by the error handling module comprises a generic framework that is set up with a rich set of optional error actions and the ability for anyone to register new actions to address newly discovered errors. In embodiments of the invention, the error handling service may be configured to trigger one or multiple error actions. An in-built error action comprises an action that removes the message from the inbound queue which allows further processing of other messages in the queue to avoid jams or blockages from occurring in the queue. In addition, the message that was processed prior to the problematic message that triggered the 'Exception' may be stored in an error inbox (where a retrieval counter is maintained). The error inbox comprises a database table with links to the original messages (or through the use of the queue indicator) so that once the problematic message has been addressed, the amended message may then be returned to its proper place in the queue to ensure consistency of the data in the integration processes.

Figure 9 illustrates a process flow for an error handling service in accordance with embodiments of the invention. As illustrated, it is shown that messages sent between application 920 and 925 are handled as previously described by module 110, e.g. through a message insertion or update process or at any point of the processing. When a message containing an error, i.e. message 901, is sent from application 920 to application 925, this error is detected by error handling module 905 as described above. The error handling module 905 then pauses the integration process for message 901, and proceeds as described in a predefined definitions list, e.g. to move message 905 into error inbox 910 to be resolved based on predefined actions associated with the identified error.

In addition to technical errors occurring at any point during the integration process, semantical errors may also occur during insert/update operation. When semantical errors occur, the integration processes will not be interrupted. Typically, existing integration platforms will continuously attempt to send these messages, causing delays to occur in the overall integration system. To address this problem, an incomplete inbox module is provided within module 110. The incomplete inbox module is configured to detect messages that contain semantical errors and to remove such messages from the integration process so that these errors may be addressed manually thereby allowing the other messages to be processed by module 110 first. In particular, messages that are handed over to a business application via API calls will usually be processed by the individual business logic of the business application, which usually comprise unique requirements associated with the business application. When plausibility checks associated with the business logic fail, the messages will be rejected and typically communicated to the API-caller via the response of the call. The continuous rejection and return of such messages typically tend to cause delays to occur.

As described above, it has been noted that the continuous rejection and return of messages that contain semantical errors are a big problem. For this reason a 'Tolerant System' business application has been proposed whereby it comprises a business application that runs through its business logic, but also accepts messages that may contain semantical errors. Such messages (e.g. an incomplete, inconsistent order) are then stored at a separate place and a special user interface (such as the 'Incomplete Inbox 1005') is utilized to allow the administrator to correct the message. Once corrected, the message is then returned to its inbound queue.

Figure 10 illustrates a process flow for an incomplete inbox module that performs the function of a 'Tolerant System' as described above in accordance with embodiments of the invention. As shown, messages sent between application 1020 and application 1025 are processed by module 110 accordingly. When incomplete inbox module 1005 detects a

message containing a semantical error such as message 1001, module 1005 will immediately remove message 1001 from the processing queue and move it into incomplete inbox 1010 for further processing. In embodiments of the invention, module 110 may host incomplete inbox modules for each business application connected to module 110. Through the use of appropriate user interfaces and APIs linked to these plurality of incomplete inbox modules, the errors associated with the messages may then be corrected and further processed normally.

Figures 11-14 sets out exemplary processes that may be performed in module 110 for integrating information, e.g. a first-message, shared between a first business application and a second business application, wherein the first and second business applications comprise disparate business applications.

Process 1100 begins at step 1102 with process 1100 receiving a first-message from the first business application. Process 1100 then retrieves local-first primary keys L1-Keys (this includes the actual Primary Key of the main business object (410a) and also all Secondary Primary Keys which are in the message, linking to other object (410b, 410c, 410d)) from the received first-message at step 1104.

Process 1100 will proceed to step 1106 for each of the retrieved local-first primary keys L1-Keys that has been assigned a global key GKey. Conversely, process 1100 will proceed to step 1112 for each of the retrieved local-first primary keys L1-Keys that has not been assigned a global key GKey

At step 1106, process 1100 will proceed to retrieve global keys GKeys associated with all the local-first primary/second keys L1-Keys as obtained from step 1104. At step 1108, process 1100 will identify and retrieve all existing L-Keys of other systems (i.e. other than the local-first primary key L1-Key) which are associated with the existing GKeys. Process 1100 also retrieves the first business application's relevant SysIds, i.e. SysId1s, and also the second business application's relevant SysIds, i.e. SysId2s. These retrieved SysIds are then appended to the GKey-L1-Key pairs and GKey-L2-Key pairs to form (GKey-SysId1-L1-Key) and (GKey-SysId2-L2-Key) entries. This takes place at step 1110. These entries are then appended to the first-message and subsequently provided to any integration logic to transform the message accordingly and handover to the second business application.

Conversely, at step 1112 (where each of the retrieved local-first primary keys L1-Keys that has not been assigned a global key GKey), process 1100 will generate new GKeys for each of these retrieved L1-Keys that do not have an associated GKey. The new GKeys are then linked with their associated L1-Keys. SysID1 is also retrieved and added to these GKey-L1-Key pairs. The GKey-SysId1-L1-Key entries are then appended to the first message.

In an embodiment of the invention, at step 1116, process 1100 receives local-second primary keys L2-Keys that are generated by the second business application for each of the (L1-Key-GKey-SysID1) entries after the GKey-SysID1-L1-Key entries have been appended to the first message and sent (after transforming by any integration logic) to the second business application. Process 1100 then pairs the received entries accordingly with the associated GKey to form (L2-Key-GKey-SysID2) entries.

In another embodiment of the invention, at step 1116, process 1100 receives local-second primary keys L2-Keys that are generated by module 110 for each of the (L1-Key-GKey-SysID1) entries as the second business application has authorized module 110 to generate the local-second primary keys L2-Keys for each of the entries. Process 1100 then pairs the entries accordingly with the associated GKey to form (L2-Key-GKey-SysID2) entries. The (GKey-SysID1-L1-Key) entries and the (L2-Key-GKey-SysID2) entries are then appended to the first message and sent to an integration logic to be transformed. With these actions the transformed message was sent to the second business application.

Regardless of either embodiment, these (L2-Key-GKey-SysID2) entries are then stored in the module for further processing and this takes place at step 1118.

Process 1100 then ends.

Figure 12 illustrates an exemplary flowchart of process 1200 for generating a local-second primary key L2-Key for a retrieved local-first primary key L1-Key in accordance with an embodiment of the invention, i.e. step 1116 of process 1100. Process 1200 begins at step 1202 with process 1200 performing an API call to the second business application. Upon receiving the response from the second business application at step 1204, process 1200 then proceeds to step 1206. At this step, process 1200 will select an appropriate local-second primary key L2-Key from the response to be paired with the local-first primary key L1-Key. Process 1200 then ends.

Figure 13 illustrates an exemplary flowchart of process 1300 for generating a local-second primary key L2-Key for a retrieved local-first primary key L1-Key in accordance with another embodiment of the invention, i.e. step 1116 of process 1100. Process 1300 begins at step 1302 by performing an API call to the second business application. However, in this embodiment, as the second business application did not respond the generated primary key to the API call performed at step 1302, process 1300 then proceeds to perform another API call to the second business application to retrieve target documents having a particular timestamp. This takes place at step 1304. This causes all the documents in the second business application that are of the same document-type as the first-message and in the

requested timeframe to be sent to module 110 and these documents are received by process 1300 at step 1306. Process 1300 then identifies a suitable document from the received response and based on the identified document, process 1300 then retrieves and selects the suitable local-second primary key L2-Key to be paired with the local-first primary key L1-Key. Process 1300 then ends.

Figure 14 illustrates an exemplary flowchart of process 1400 for generating a local-second primary key L2-Key for a retrieved local-first primary key L1-Key in accordance with yet another embodiment of the invention, i.e. step 1116 of process 1100. Process 1400 begins at step 1402 by performing an API call to the second business application. However, in this embodiment, as the second business application did not respond to the API call performed with a generated primary key at step 1402, process 1400 then proceeds to perform another API call to the second business application to retrieve target documents having the same document-type, and filtered by identifier fields (e.g. name, total amount, last updated, etc. (based on definitions for this system)). This takes place at step 1404. This causes all the documents in the second business application that are of the same document-type as the first-message and fitting to the defined filter to be sent to module 110 and these documents are received by process 1400 at step 1406. Process 1400 then identifies the suitable document from the received response and based on the identified document, process 1400 then retrieves and selects the suitable local-second primary key L2-Key to be paired with the local-first primary key L1-Key. This takes places at step 1408 Process 1400 then ends.

Numerous other changes, substitutions, variations and modifications may be ascertained by one skilled in the art and it is intended that the present invention encompass all such changes, substitutions, variations and modifications as falling within the scope of the appended claims.

CLAIMS:

1. A module for tracking information shared between a first business application and a second business application, wherein the first and second business applications comprise disparate business applications, the module comprising:

- a processing unit; and

- a non-transitory media readable by the processing unit, the media storing instructions that when executed by the processing unit cause the processing unit to:

 - receive a first-message from the first business application;

 - retrieve local-first primary keys L1-Keys associated with the first-message;

 - for each of the retrieved local-first primary keys L1-Keys that has been assigned a global key GKey, identify a local-second primary key L2-Key and a semantic key SKey that have been paired with the local-first primary key L1-Key, and retrieve system identities associated with the local-first primary key L1-Key and the local-second primary key L2-Key;

 - for each of the retrieved local-first primary keys L1-Keys that has not been assigned a global key GKey, generate and assign a global key GKey and a semantic key SKey to the local-first primary key L1-Key to form a (Gkey, L1-Key, SKey) entry, receive a local-second primary key L2-Key generated for the (Gkey, L1-Key, SKey) entry, and retrieve system identities associated with the local-first primary key L1-Key and the local-second primary key L2-Key;

 - append all the retrieved local-first primary keys L1-Keys, their associated global keys GKeys and semantic keys SKeys, and the respective retrieved system identities to the first-message, wherein the semantic keys SKeys are identifiable by the first and second business applications;

 - provide the appended first-message to an integration logic module, whereby the integration logic module is configured to process the appended first-message and provide the processed first-message to the second business application;

 - receive a response from the second business application, wherein the response is stored in a delivery log;

 - determine, from the response, a delivery status of the appended first-message.

2. The module according to claim 1 wherein the instructions to determine the delivery status of the appended first-message comprises:

- instructions for directing the processing unit to:

 - for each of the retrieved local-first primary keys L1-Keys that has been assigned a global key GKey,

identify from the response, the retrieved local-first primary keys L1-Keys that were appended to the first-message, and

validate the delivery status of the appended first-message when all the local-first primary keys L1-Keys that were appended to the first-message are identified from the response.

3. The module according to claim 1 wherein the instructions to determine the delivery status of the appended first-message comprises:

instructions for directing the processing unit to:

for each of the retrieved local-first primary keys L1-Keys that has not been assigned a global key GKey,

identify from the response, local-second primary keys L2-Keys, and

validate the delivery status of the appended first-message when all the local-second primary keys L2-Keys identified from the response match the received local-second primary keys L2-Keys generated for the (Gkey, L1-Key, SKey) entries.

4. The module according to claim 1 wherein the module further comprises:

instructions for directing the processing unit to:

assign timestamps to all instructions executed by the processing unit; and

determine from the delivery log, using a semantic key SKey associated with the appended first-message, a delivery status of the appended first-message.

5. The module according to claim 4 wherein the instructions to determine from the delivery log, using the semantic key SKey associated with the appended first-message, the delivery status of the appended first-message comprises:

instructions for directing the processing unit to:

retrieve (Gkey, L1-Key, L2-Key) entries associated with the SKey from the delivery log;

arrange the retrieved (Gkey, L1-Key, L2-Key) entries according to their timestamps; and

validate, using the timestamps, the delivery status of the appended first-message.

6. The module according to claim 1 wherein the module further comprises:

instructions for directing the processing unit to:

receive, from the integration logic module, errors, messages associated with the errors, and a queue indicator linked with each of the messages;

rectify the errors associated with the messages; and

return each rectified message to its original queue location in the integration logic module based on the queue indicator linked with the message.

7. The module according to claim 1 wherein the local-second primary key L2-Key generated for the (Gkey, L1-Key) entry is generated by the module based on a Global Unique Identifier (GUID), a random number, a primary key generation algorithm in combination with a number range, or a primary key generation algorithm in combination with a defined logic.

8. The module according to claim 1 wherein the local-second primary key L2-Key generated for the (Gkey, L1-Key) entry is generated as part of a response by the second business application in response to an Application Programming Interface (API) call from the first business application, whereby the generated response is provided to the first business application.

9. The module according to claim 1 wherein the local-second primary key L2-Key generated for the (Gkey, L1-Key) entry is generated by the module based on a bundle of documents received from the second business application, the bundle of documents being sent by the second business application to the first business application in response to a second continuous API call from the first business application, whereby the bundle of documents comprise documents having a same document-type as the first-message and having a similar time-stamp as a first API that was sent from the first business application to the second business application.

10. The module according to claim 1 wherein the local-second primary key L2-Key generated for the (Gkey, L1-Key) entry is generated by the module based on a bundle of documents received from the second business application, the bundle of documents being sent by the second business application to the first business application in response to a second continuous API call from the first business application, whereby the bundle of documents comprise documents having a same document-type as the first-message and having fields that match predefined requirements.

11. A method for tracking information shared between a first business application and a second business application, wherein the first and second business applications comprise disparate business applications, the method comprising:

receiving, using a module, a first-message from the first business application;
retrieving, using the module, local-first primary keys L1-Keys associated with the first-message;

for each of the retrieved local-first primary keys L1-Keys that has been assigned a global key GKey, identify a local-second primary key L2-Key and a semantic key SKey that have been paired with the local-first primary key L1-Key, and retrieve system identities associated with the local-first primary key L1-Key and the local-second primary key L2-Key;

for each of the retrieved local-first primary keys L1-Keys that has not been assigned a global key GKey, generate and assign a global key GKey and a semantic key SKey to the local-first primary key L1-Key to form a (Gkey, L1-Key, SKey) entry, receive a local-second primary key L2-Key generated for the (Gkey, L1-Key, SKey) entry, and retrieve system identities associated with the local-first primary key L1-Key and the local-second primary key L2-Key;

appending, using the module, all the retrieved local-first primary keys L1-Keys, their associated global keys GKeys and semantic keys SKeys, and the respective retrieved system identities to the first-message, wherein the semantic keys SKeys are identifiable by the first and second business applications;

providing, using the module, the appended first-message to an integration logic module;

receiving, using the module, a response from the second business application, wherein the response is stored in a delivery log;

determining, using the module, from the response, a delivery status of the appended first-message.

12. The method according to claim 11 wherein the step of determining the delivery status of the appended first-message comprises:

for each of the retrieved local-first primary keys L1-Keys that has been assigned a global key GKey,

identifying, using the module, from the response, the retrieved local-first primary keys L1-Keys that were appended to the first-message, and

validating, using the module, the delivery status of the appended first-message when all the local-first primary keys L1-Keys that were appended to the first-message are identified from the response.

13. The method according to claim 11 wherein the step of determining the delivery status of the appended first-message comprises:

for each of the retrieved local-first primary keys L1-Keys that has not been assigned a global key GKey,

identifying, using the module, from the response, local-second primary keys L2-Keys, and

validating, using the module, the delivery status of the appended first-message when all the local-second primary keys L2-Keys identified from the response match the received local-second primary keys L2-Keys generated for the (Gkey, L1-Key, SKey) entries.

14. The method according to claim 11 wherein the method further comprises the steps of: assigning, using the module, timestamps to all instructions executed by the processing unit; and

determining, using the module, from the delivery log, using a semantic key SKey associated with the appended first-message, a delivery status of the appended first-message.

15. The method according to claim 14 wherein the step of determining from the delivery log, using the semantic key SKey associated with the appended first-message, the delivery status of the appended first-message comprises the steps of:

retrieving, using the module, (Gkey, L1-Key, L2-Key) entries associated with the SKey from the delivery log;

arranging, using the module, the retrieved (Gkey, L1-Key, L2-Key) entries according to their timestamps; and

validating, using the module, based on the timestamps, the delivery status of the appended first-message.

16. The method according to claim 11 wherein the method further comprises the steps of: receiving, using the module, from the integration logic module, errors, messages associated with the errors, and a queue indicator linked with each of the messages;

rectifying, using the module, the errors associated with the messages; and

returning, using the module, each rectified message to its original queue location in the integration logic module based on the queue indicator linked with the message.

17. The method according to claim 11 wherein the local-second primary key L2-Key generated for the (Gkey, L1-Key) entry is generated by the module based on a Global Unique Identifier (GUID), a random number, a primary key generation algorithm in combination with a number range, or a primary key generation algorithm in combination with a defined logic.

18. The method according to claim 11 wherein the local-second primary key L2-Key generated for the (Gkey, L1-Key) entry is generated as part of a response by the second business application in response to an Application Programming Interface (API) call from the first business application, whereby the generated response is provided to the first business application.

19. The method according to claim 11 wherein the local-second primary key L2-Key generated for the (Gkey, L1-Key) entry is generated by the module based on a bundle of documents received from the second business application, the bundle of documents being sent by the second business application to the first business application in response to a second continuous API call from the first business application, whereby the bundle of documents comprise documents having a same document-type as the first-message and having a similar time-stamp as a first API that was sent from the first business application to the second business application.

20. The method according to claim 11 wherein the local-second primary key L2-Key generated for the (Gkey, L1-Key) entry is generated by the module based on a bundle of documents received from the second business application, the bundle of documents being sent by the second business application to the first business application in response to a second continuous API call from the first business application, whereby the bundle of documents comprise documents having a same document-type as the first-message and having fields that match predefined requirements.