I.S.F.
Improved Systems Factors

AN INTRODUCTION TO STRUCTURED TRANSACTIONAL MESSAGES

BY MALCOLM HAMER

Introduction

This paper presents an account of the history of the emergence of standards for "structured transactional messages" and reviews the various standards that have evolved to serve different industries.

A transactional message is a message that accomplishes a specific business or administrative goal, such as:

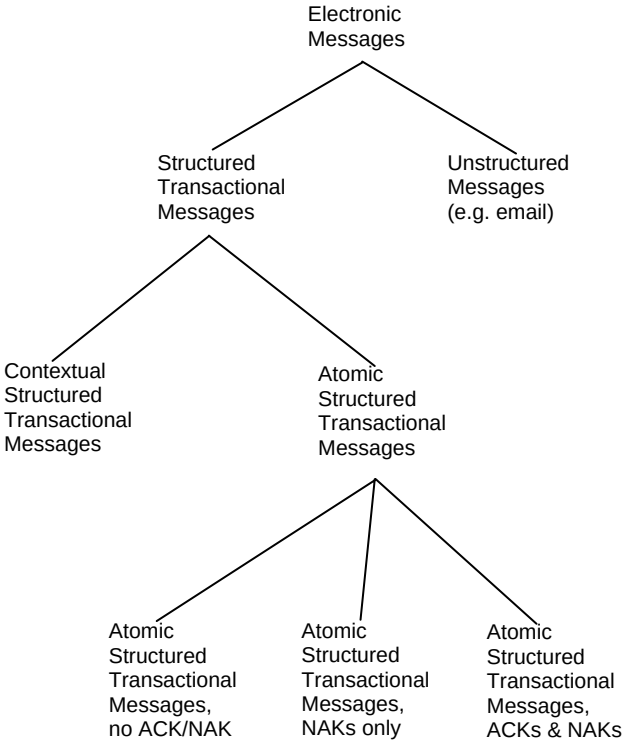
- Transferring money between accounts at two different financial institutions
- Sending instructions from a funds manager to a broker for a stock purchase
- Placing an order for goods to be supplied to a buyer (such as a hospital) by a supplying organization (such as a hospital-supplies distributor).

This paper discusses only structured transactional messages. A structured message is one that is "machine-readable". The message is sent from one computer system to another on a peer-to-peer basis and the computer at the receiving end can, by virtue of the strict format that the message follows, easily extract from the message all the various pieces of information required to execute the transaction. The standards that define such messages include rules about what goes in the message and how the message is formatted. By contrast, an unstructured message is a message, such as an email message, composed by a human being and read by another human being. Although it may be passed from one computer system to another in the process of being delivered to the addressee, an unstructured message lacks any rules-based structure that would allow a computer program to easily and reliably extract actionable information from it.

To be classified as a true structured-message standard, a standard must be defined in a way that allows it to be used in a peer-to-peer situation, between two different organizations using their own implementations of the standard. Excluded from this definition is any arrangement where the system that executes the transaction sends any kind of "blank form" to the system that is requesting the transaction. This is not a message-based transaction. All that is happening in this arrangement is that the transaction-initiating system is effectively "filling in the form". This method of working is often used with devices attached to mainframes, where the transaction-initiating device behaves like a surrogate "dumb terminal". This is not true peer-to-peer message-based communication.

Types of structured transactional message

Structured transactional messages fall into a number of categories. The distinctions between these categories are shown in the following diagram and are described below.



(a) Structured messages may be atomic or contextual:

Structured transactional messages are said to be atomic if they are self-contained and capable of being interpreted, and acted upon, without taking into account the contents of any prior message sent from the same sending system to the same receiving system. This is the most common type of structured transactional message.

By contrast, contextual messages, which are less common, are messages that form part of a pre-defined sequence of messages, exchanged between the sending system and the receiving system. A single message within such a sequence does not, by itself, contain sufficient information for the receiving system to execute the transaction. Also, in some cases, the meaning of a message may vary depending on its position within the sequence.

Contextual structured messages are used in situations where the transaction as a whole involves several distinct steps, or involves some kind of negotiation. For example, when you try to get cash from a cash machine, the cash machine has to first send a message to the bank's computer system saying that you wish to withdraw a certain amount of cash. The bank's computer system checks that you have enough money in your account and, if you do, sends an "OK, go ahead" message to the cash machine. The cash machine then starts dispensing the cash and sends an "I have done it" message to the bank's computer system. The computer system then debits your account. This kind of sequence is called a two-phase commit, because the transaction is first set up and approved, and then it is executed and confirmed. The "I have

done it" message cannot be sent by the cash machine as a stand-alone message. The receiving system can act upon that message only within the context created by the prior message.

In this paper we will be largely concerned with atomic messages.

(b) Atomic structured message standards may use NAKs; ACKs and NAKs; or neither:

When an atomic message is sent from one system to another, the receiving system executes the transaction based on the content of that message. The message is self-contained. However, there may, nevertheless, be a message sent in the reverse direction (from the receiving system to the sending system) to acknowledge the transaction. This does not affect the classification of the main message as "atomic".

There are three possible acknowledgement arrangements:

- No acknowledgement. The sending system has no information about whether or not a transaction was successfully executed.
- Only negative acknowledgements ("NAKs") are sent. A NAK is sent if the transaction could not be executed. The sending computer assumes that all has gone well in the absence of a NAK.
- An acknowledgement is sent in all cases. This may be a positive acknowledgement (an "ACK"), confirming that the transaction has been executed, or a negative acknowledgement (a "NAK"), advising the sending system that the transaction could not be completed for some reason.

Acknowledgements are, by their nature, contextual: there must have been a main message, sent earlier, for an acknowledgement to have any meaning to the system that receives it. However, even where acknowledgements are used, the overall message scheme (the definition of the structure of the main message, the rules for acknowledgement-sending, and the definition of the structure of acknowledgements) is still described as an "atomic message standard".

The most common acknowledgement scheme is the use of NAKs only. This provides for recovery from the hopefully rare situations where a received message cannot be processed by the receiving system, without burdening the sending system with keeping track of the successful processing of every transaction (as happens when ACKs are used).

(c) Structured messages may be printable or bit-oriented:

This distinction is not shown on the earlier diagram: it applies to all four categories of structured message. A printable structured message is a message composed of normal keyboard characters (generally ASCII-encoded). Even though structured messages are normally "read" by a computer program, printable structured messages may be printed out as they stand and read by a human being. The message contains no strings of bits outside the ASCII coding scheme and no non-printable ASCII values (such as control-G) within the ASCII coding scheme. For the purposes of this definition, formatting characters such as carriage return and line-feed are treated as "printable" characters; but tab characters and backspaces are not, because their presence cannot be detected in a printed rendering of the message.

Many message standards for printable structured messages restrict the ASCII printables that are permissible in a message, rather than allowing the full range of keyboard characters. Common restrictions placed on the characters that are allowed in a message (some of which arose for historical reasons that no longer apply) include:

- Restricting the allowed characters to those that appear on a telex keyboard – permitting only upper case letters, digits, and basic punctuation marks.

- Forbidding the use of ACSII values associated with keyboard keys that have different labeling in different countries, such as the ASCII value that corresponds in the US to the "$" sign.

The fact that a message may be encrypted just before transmission, and then decrypted at the receiving end, does not affect its classification as a "printable" message. The description "printable" refers to the message as it appears "in the clear", before encryption and after decryption.

In contrast to a printable message, a bit-oriented structured message is one that, either in whole or in part, consists of strings of bits that the receiving system interprets as units of information other than printable keyboard characters. For example, individual bits in some bytes in the message may have specific meanings. As a result, the message cannot be studied by a human being unless a program is used that parses the message, bit by bit, and translates it into a readable form.

Most structured transactional message standards are based on printable-message formats. There is a good reason for this. Using printable formats allows messages to be printed out and studied by programmers. This is very useful when the software that sends and receives the messages is being tested and debugged, or when problems are being investigated after the system has gone live. Programmers can much more easily spot errors in the structure and content of messages if they can study examples of messages on a screen or on a piece of paper. In the early days of structured transactional messages, printable formats were adopted for two further reasons that do not often apply today: the need to be able to store archive copies of messages in microfiche format, and the need to be able to transmit the messages over the public telex network (where only five-bit Baudot-encoded characters can be used). Although these requirements no longer apply, the ability to print out and study sample messages is a good enough reason, in itself, to continue the practice of using printable formats.

Bit-oriented message formats were used in the early days of structured transactional messages in situations where the bandwidth available to convey the messages was extremely limited. In these situations messages had to be kept as short as possible. Longer messages would have slowed down the business process for which they were being used. For example, the first cash machine networks used a message format that was bit-oriented. This was because most of the connections between the cash machines and the data center operated at only 75 bps. The use of printable-format messages under these circumstances would have caused the cash-dispensing process to take several seconds longer, reducing the efficiency of the then-costly cash machines.

Bit-oriented message formats have also been used where transaction volumes were expected to be extremely high. The processing of a bit-oriented message has been found (at least up until recently) to put a lighter load on the computer or server doing the processing – because the extraction of information from a bit-oriented message is a simpler task: bits and bytes of transaction data are in fixed positions within a message and are therefore quickly located and extracted. This consideration has been one of the several obstacles to replacing established bit-oriented structured message standards with printable message standards. For example, in the realm of credit card transaction networks, the bit-oriented ISO 8583 standard is firmly

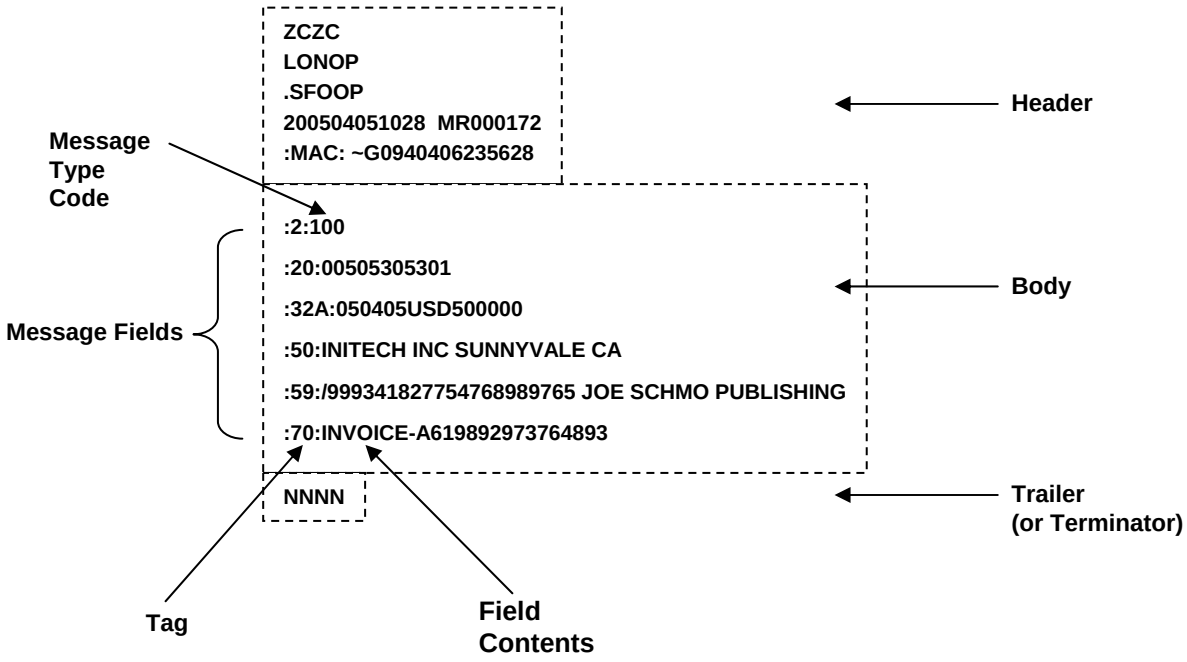
entrenched and unlikely to be replaced in the near future. This is partly because (aside from the complexity of implementing a global change in the standard) a change to a printable message standard would have demanded an increase in the processing power of every server and every embedded processor chip in millions of devices globally.

However, recent improvements in processor designs that incorporate sophisticated string-handling functions on the processor chip have improved the efficiency of parsing operations for printable-format messages to the point where the processing-load difference between bit-oriented and printable message standards is now small. As a result, *newly* created bit-oriented standards are now very rare. They would be introduced only (a) where the means of conveying the messages has an unusual bandwidth limitation or (b) where the designer is an idiot.

Message Structure Terminology

Having covered the basic categorization of different structured message standards we will now look at the terminology used in defining a particular standard. The following terms are used when describing the rules according to which a structured message is assembled.

A structured message is said to consist of a "message header", a "message body", and a "message trailer".



The header is the first part of the message. The header contains some or all of the following:

- The address of the message, which defines exactly where the destination system is located; which of the systems at that location the message is intended for; and, in some cases, which particular piece of software in that system should act upon the message.
- The address of the sending system.
- Other information about the message as a whole, such as the time and date it was sent, and a reference number that uniquely identifies the message.

- Security-related information, such as a message authentication code (MAC), used by the receiving system to check that the message has not been altered since it was created at the sending system and/or to validate the identity of the sending system. (In some standards the MAC may be part of the trailer, rather than the header.)

Following the header is the body of the message, which contains the actual transaction. The final part of a message is commonly described as the message trailer or message terminator. This marks the end of the message.

In most message-standards documents, the header and trailer are typically defined and described in the first section of the document. The reason for them to be dealt with together in this way is that, taken together, they are thought of as being the "envelope" (analogous to a paper envelope in which a letter is mailed). The envelope "contains" the body of the message. In some standards the header and trailer may be covered in a document that is completely separate from the larger document that deals with the message body. Such a separate header/trailer definition is most likely to be created where the organization that publishes the standard makes available a choice of two or more different "enveloping" standards for use with a single message-body standard. Different header/trailer standards may be used, for instance, on different physical networks over which the messages may be conveyed.

Within the body of the message the contents are typically composed of a number of "fields" and "subfields". A field is a distinct piece of information that forms part of the details of the transaction, such as an account number or an amount. Subfields are finer levels of detail within a field. The number of "tiers" (field, subfield, subsubfield, and so on) varies from standard to standard. Two tiers (that is, fields and subfields) is the most common scheme. Another term that may be used in some standards instead of "field" is data element.

In some of the standards that allow for three or more structural tiers within the message body, the highest tier is called a "data segment" (rather than a "field"), the next tier (or tiers) is called a "composite data element", and an element at any level that contains no other elements is called simply a "data element".

Tagged and Untagged Message Fields

The example above shows message fields that start with "tags". This is by far the most common arrangement in modern message standards and I will describe this first. However, some of the earliest standards adopted a different approach to field-identification, namely, positionally identified fields, which I will describe second.

In standards based on the use of field tags, each field (or "data element") is composed of a "tag", followed by the field contents. A tag (also sometimes called a "field identifier code", "field label", or "data element identifier") tells the receiving system what piece of information is contained in the field. For example, one tag value may mean "this is the account number of the account to be credited" and another tag value may mean "this is the amount of the transaction". In a printable-format message, a tag is typically a fairly small number of characters, with a distinctive format such as ":36B:". (In a bit-oriented message the tag may be very short, such as eight bits, or sometimes even fewer.)

However, in *untagged* message standards, the fields are identified by their exact position in the message body. The fields always appear in a defined order and all fields must be present in a given type of message. In this way, the receiving system can determine which field is which. In some standards the start of each field is rigidly defined as starting an exact number of bytes from the start of the message body. In this case each field must be of a pre-determined number of bytes and, if the actual data in a field is less than that number of bytes, it must be "padded out" to that length using spaces (or another padding character), either before or after the actual data. In less rigid standards, the fields must still be in a specific order but can contain a variable number of bytes. In these standards a special character (which cannot be allowed to appear in the actual data) is used to mark the end of each field. In either case, fields cannot be omitted, even if they contain no data. Empty fields must be included and must be full of padding characters or contain only the end-of-field character, depending on whether the standard uses fixed-length fields or an end-of-field character.

An example of an untagged format is the message standard (mentioned earlier) for credit and debit card transaction authorizations, ISO 8583. This standard is based on a 200-byte message format containing 28 fields with lengths of between 1 and 40 bytes. All information inserted into a field must be either already equal to the length of the field or padded to the field length with trailing space characters.

The need for most message standards to evolve is now widely appreciated. Once an untagged standard is in operation it is extremely hard to implement upgrades to the standard because this typically requires making the change simultaneously network-wide, across all interconnected systems. In the future, we should therefore expect new untagged message standards to become extremely rare – although entrenched standards, such as ISO 8583, are unlikely to be displaced, because of the huge investment in current point-of-sale terminals, cash machines, and authorization-system software.

Parsing messages in the receiving system

In order for the receiving system to be able to decode (or "parse") an inbound message, all the rules set out in the standard must be programmed in the software that analyzes inbound messages in order to extract the information or instruction contained in the message. These rules specify:

- The meaning of the information contained in each field.

- The format according to which the data is presented, for example, "an alphanumeric string of up to 16 characters" or "a decimal value with up to ten digits to the left of the decimal point and two digits after the decimal point".

It is possible, although unusual, to encode the field-format definition within the field itself. For example, in between the tag and the actual field contents it is possible to include characters that define the format of the field content – telling the receiving system whether to treat the field as a string of characters (like an account number) or a decimal value (like an amount). However, this arrangement is redundant. The software in the receiving system must anyway store information about what each field means. It makes much more sense to store the format definition in the receiving system, along with the information about what each field means, rather than to constantly re-send the format definition with every message. However, there is one aspect of the format definition which can usefully be included in the message itself. This is a "length

descriptor", placed between the tag and the field contents. A length descriptor tells the receiving system how many characters to expect to find in the field contents. A length descriptor can be useful for certain fields because it simplifies (for the receiving system) the task of parsing message fields that contain information consisting of a large, variable numbers of characters.

One very important field is the message type field. This is the field that defines which class of message the message belongs to (within the full set of possible message classes). For example, the message type code may mark the message as a "funds transfer" message or a "securities transaction" message. A message type field typically consists of a message type tag followed by a message type code, which may consist of a short string of digits such as '100' or a short string of characters such as 'MT838'.

Other things that standards specify about messages

When you start to read the main part of a message standard, covering the message body, you will generally find that the following subjects are addressed.

Field inclusion and repetition:

When you read the definition of the composition of each specific message type (such as a "funds transfer message") you will generally find that the section that describes the message type starts by listing all the tag values that may appear in a message of that type.

Then, for each of these tag values, the standard will typically assign each field to one of the following four categories:

- Single, Mandatory (SM). This means that there can be only one field in a message with this tag value; and every message of the type in question must contain this field.

- Single, Optional (SO). This means that the field may, or may not, be used in the type of message in question; but if it is used there can only be one such field.

- Multiple, Mandatory (MM). This means that there can be one or several fields with this tag value; but there must be at least one.

- Multiple, Optional (MO). This means that this field may, or may not, be used in a message; and there may be one or several fields with this tag.

In some standards all fields are defined as single fields (SM or SO). In such standards, multiple items are handled within multiple subfields contained within a field – one subfield for each item. Some standards also define "Single, Conditional" (SC) and "Multiple, Conditional" (MC) fields. In this context, "conditional" means "mandatory if certain other fields are present".

Field separation:

Separation of one field from the next field within a message can be done in one of several ways. In printable-format messages a common method of indicating the end of each field is with a carriage return. Another common end-of-field character is the ASCII 'SOH' character. In other standards, the appearance of the next tag may mark the end of the previous field: rules are applied that prohibit any string of characters in the field contents that could be mistaken for a tag. This ensures that each new tag unambiguously marks the end of the prior field.

Field order:

Some standards require that fields appear in a certain order within the message body, such as in ascending numerical or alphanumerical value of the tag; some require certain fields (such as the message type code) to be first, but allow the rest to be in any order; and others allow fields to appear in any order. In general, it is better to design a standard so that the order of the fields does not affect the meaning of the message. However, even though the standard may be written to make it able to operate with any field order, the standard may strongly recommend a certain order, so that the message-parsing software can operate more efficiently, or so that messages are more easily read by a human being during testing and debugging of software.

Placement of the Message Type Field:

The message type code is the first piece of information that the receiving system must have in order to be able to parse the message. It tells the receiving system what to expect in terms of what other fields may be present. Because of this, the message type field is either (a) required to be the first field in the body of the message, or (b) is physically located within the header, rather than in the message body (although, conceptually, it should be regarded as part of the message body).

The most common reason for defining the message type field as part of the header is because, in some networks, the message type code may be used by the network in the process of transporting the message to the destination system. The transport network may do this in order to make a choice between two logically or physically separate systems at the destination. Because the network is typically designed to examine only the headers of messages, the placement of the message type field in the header allows the network to route messages directly to the required system. However, the receiving system will then need to parse the header in order to extract the message type code from the header and use this along with the message body when processing the message. This is why it is important to regard the message type field as being logically part of the message body, even when it is placed in the header for reasons of selective routing of messages to different receiving systems.

Where a message standard uses in-the-header placement of message type fields, you will typically find that the message-standard document places the description of the message type field and the allowed values and meanings of the message type code in the “header” section of the standard, rather “body” part of the standard.

It should be noted that the placement of the message type field in the body of the message does not rule out the possibility of the transport network taking account of the value of the message type code when routing the message to the correct receiving system. The software in the network can parse the whole message and locate the message type field wherever it is located. In fact, some networks parse the whole message as a matter of course, in order to take account of the contents of fields in the message body when routing the message. This practice is called content-based routing. For example, the network may selectively route messages with certain characteristics, such as account numbers starting with a particular first digit or transactions denominated in certain currencies, to a different system. Thus, the distinction between the header and the body may become blurred in certain “intelligent” transport networks. Nevertheless, the distinction is worth maintaining when defining a standard. This is because, as mentioned earlier, different forms of header may be attached to the

message body depending on which transport network is to be used. It is therefore important to clearly specify what constitutes the body of the message and what constitutes the header.

The history of structured transactional messaging

Historically, the move towards structured transactional messaging can be thought of as having taken place in two stages. In the first stage, only the header of the message was changed to a structured format, so that the transport network could route the message from its source to its destination without human intervention. In the second stage, the body of the message was structured so that the message could pass from a sending computer, via the transport network, to the receiving computer, where it would be executed without any human intervention. Looking back at this evolution of message standards we would not describe the first stage as representing true "structured messages". However, the first stage was a pre-requisite for the later development of fully-structured messages; and so it is important to understand how that first stage was reached.

Before any kind of transactional message could be carried over a distance, it was necessary to have a network capable of carrying digital information. The first such network was the telegraph network. Telegraph networks first appeared in 1837. (By the way, although we tend to think of "digital" techniques as a late-twentieth-century development, digital long-distance communication by electrical means appeared on the scene over 40 years before analog telephone communication. Earlier non-electrical communication systems, using smoke signals and semaphore, were also digital.) The first telegraph networks were based on the manual transmission of messages using the Morse Code. The first step in automating the telegraph networks occurred 38 years later, with the introduction of the teleprinter. The teleprinter was invented by Émile Baudot in 1870, and its deployment started in 1875. The Baudot teleprinter used five-bit binary codes to represent printed characters.

Initially, public telegraph networks operated on the basis of telegraph operators manually retyping messages at each intermediate relay center, as the message made its way to its destination over a series of point-to-point links. But the operation of relay centers was soon streamlined (and errors were reduced) by the introduction of five-hole punched paper tape printers and readers, also invented by Baudot, in 1881. With these tape printers and readers attached to the teleprinters, operators in the relay centers could receive an inbound message on a tape, tear off the tape, dash over to the appropriate teleprinter for the outbound route, and insert the tape into the tape reader there. This was known as a "torn-tape" relay system.

The torn-tape system continued to be the standard method of telegraphic communication for the next fifty years. Then, in the early 1930s, the first telex networks started to be deployed around the world. In these networks, the torn-tape centers were replaced by automatic exchanges. These exchanges were based on an adaptation of Strowger switching technology, which had been used from 1892 onwards to automate the telephone network.

In the telex network, telex terminals were installed by the network provider in customers' offices. These terminals consisted of a teleprinter plus a telephone-style dial. The dial allowed the customer to make an automatic telex "call" direct to the telex machine of another customer. Thus, there was no longer any need for the torn-tape relay centers to route messages to their destination.

Telex terminals could be used interactively for “instant messaging” (as we call it today); but in many business sectors the most common practice was to prepare punched tapes on an off-line terminal and proof-read messages before sending them. Most telex machines were left to receive messages unattended: someone would check them periodically to collect incoming messages and make sure that the paper roll had not run out.

Although the public telex networks, operating in dial-up mode (that is, circuit-switched mode), started to go into operation in the 1930s, the majority of private telegraph networks, operated by large companies and cooperative groups, continued to use their own versions of the public torn-tape relay system well into the 1970s. However, some of them, particularly the large banks, did start to use the public telex network for *external* communication, for example, to deliver messages to other banks and to corporate customers. Banks with international torn-tape private networks also used these private networks to get messages to a destination country, and then used the local telex network to deliver the message to the correspondent bank or customer within that country. This practice was called “telex refile”. It was frowned upon by some national telex network carriers, who were being deprived of international telex revenues by this practice.

The owners of the private torn-tape telegraph networks, having ignored telex as a technology that might be used for their internal networks, finally accepted the need to do something about their growing torn-tape relay centers, which were becoming costly and, in some cases, chaotic. In choosing the technology to do this, they decided to break away from the public-network approach of a dial-up network, like telex. They decided, instead, to stick with the “message switching” approach of the torn-tape relay, and automate the torn-tape function as it stood. To do this they set about writing software to run on the new generation of “minicomputers”. (Minicomputers appeared in the late 1960s as an alternative to the very costly mainframes of the 1950s and 1960s.) The resulting systems, consisting of the message-switching software and the minicomputers on which this software ran, became known as “store-and-forward” switching systems.

This approach had three advantages over adoption of telex technology. First, it allowed the senders of messages to get their messages out without waiting, as you have to do in a telex network, for the destination terminal to become free. Second, it allowed the network owners to maximize the traffic loading on expensive leased circuits, particularly international leased circuits, by queuing messages in the switching systems and pumping them down the links, one after another. And third, it made it easy to keep copies of messages, stored on the computers’ hard drives, for future reference. These copies were then easily accessible in the hours after messages were sent, in case a delivery problem occurred. The copies would be kept on the system for a few days and then archived to microfiche, creating a permanent record that does not exist in a telex-based network.

The first industry to introduce minicomputer-based “store-and-forward” switches, in place of torn-tape relay, was the airline industry. The International Air Transport Association (IATA), which was founded in 1919, had created a separate operational arm in 1949 called SITA (Société Internationale de Télécommunications Aéronautiques). SITA operated a large international torn-tape network, which the industry used for inter-airline and intra-airline communication. In the early 1960s, SITA started a project to automate its network, using store-and-forward switching systems. They started the deployment of these systems with the replacement of the Frankfurt torn-tape center in 1966. By 1970, the store-and-forward network was complete, with switching systems in New York, London, Hong Kong, Amsterdam, Brussels, Madrid, Paris, and Rome.

Initially, all messages sent via the SITA network had unstructured bodies, but the headers had a well-defined format (generally referred to as “the IATA format”). This header format allowed the switching systems to parse the header and act upon it, routing the message to its destination. The structure of the header was based on standards that were already being used in the SITA torn-tape operation, adopted from the earlier public torn-tape networks. The header began with a start-of-message code, which was ‘ZCZC’, followed by an alphabetic destination address, a source address (which identified the address of the terminal from which the message was sent), and various other elements. All messages had a terminator code, which was ‘NNNN’.

It was a rule that the body of the message must never contain either ‘ZCZC’ or ‘NNNN’, since these would abort or prematurely terminate the message. (‘ZCZC’, by the way, is pronounced “zed see zed see”, even if you are an American English speaker. You should avoid the mistake of pronouncing it “zee see zee see” if you are talking to “old hands” from the industry. They will laugh at you – as they laughed at me the first time I tried to Americanize the pronunciation of ‘ZCZC’ for an American audience at a presentation in New York in 1980.)

The format of the IATA destination address was based on using the IATA airport codes (for example, BRU for Brussels, or LAX for Los Angeles), followed by additional letters to indicate the airline or airport department.

In 1971, SITA proceeded to the next stage of its automation plan by encouraging its member airlines to start using the network to transport structured message bodies for various common transactions, for example, dispatch-release messages (which authorized each flight to take off, giving route, weather, and aircraft data), and reservations messages reporting "free sale" seats (that is, seats sold by stations without access to the appropriate reservations system). The airlines connected a number of computer systems directly to the network and these systems were able to start exchanging structured messages. Thus, the era of structured transactional messaging began.

The next industry to start using structured messages was the banking industry. The large international banks were quick to copy the SITA network. In fact, many of them copied the message header formats in almost every detail, including use of the IATA airport codes to identify their branches. So, for example, the foreign exchange department in Brussels might be assigned an address of ‘BRUFX’. The IATA codes were adopted by the large banks partly because this was a good idea – the airport codes are short, and all the cities in which the banks have branches have airports, and therefore IATA codes – and partly because some of the programmers who worked on the SITA network formed the core of the new messaging software development teams in the banks.

One of the first attempts in the banking industry to create a structured format for the message body was initiated at Citibank (at that time called First National City Bank). Starting in 1973, Citibank developed a standard covering common transactions, such as funds-transfers. The standard was known as MARTI (short for Machine-Readable Telegraphic Input). In 1974 Citibank started using MARTI for internal transactions between its international branches and for transactions with some of its correspondent banks. (These banks were, up until then, exchanging instructions for interbank transactions with Citibank, and with one another, via the public telex network, using unstructured or semi-structured messages, handled by human beings at the receiving end.).

Following these initial successes with MARTI, Citibank wrote to all its correspondent banks telling them that, as of March 31, 1975, Citibank would accept transactions only in MARTI format. Well-intentioned though this was, unfortunately it backfired. Although some of Citibank's correspondent banks had willingly adopted MARTI, many not only refused to use MARTI but also started to move as much of their correspondent business as they could to the other large banks. This led to a major climb-down by Citibank. A newly appointed head of the correspondent-bank business, in an attempt to smooth ruffled feathers, went as far as to welcome a loosening of existing informal agreements about semi-structured formats for human-readable messages, reportedly saying, "We'll even take your instructions on toilet paper, if that's what it takes to win back your business." Nevertheless, for its own international branch-to-branch transactions, Citibank continued to use the MARTI format, thus piloting structured transactional messages in the banking industry.

It should be noted that the idea of eliminating the human element in the processing of interbank transactions pre-dated MARTI by several years. For example, transactions between the US banks and the Federal Reserve Bank, under the "FedWire" system, were already based on the exchange of magnetic tapes, sent via courier, to transmit transactions. These tapes contained machine-readable data. However, sending tapes for batch processing does not fall within the normal definition of transactional messaging. FedWire eventually converted to a message-based approach.

Cooperation between banks in Europe got off to a less bumpy start than in the US. As in the US, interbank transactions were being done using free-format or semi-formatted messages, sent via public telex, and acted upon by human beings at the receiving bank. In the early 1970s, a number of European banks started exchanging ideas about performing interbank funds transfers (on behalf of their corporate customers) by means of structured messages. This exchange of ideas led to an agreement, in 1973, to found a society that would (a) define the standards for the messages, and (b) operate a network that would be shared by all its members. This society was S.W.I.F.T. (S.W.I.F.T. stands for Society for Worldwide Interbank Financial Telecommunication. Note that the "T" stands for "Telecommunication", without an "s", not "Transaction". Also note that S.W.I.F.T. treats its name as an acronym, pronouncing it "SWIFT", but always writes it with the five periods.)

The first S.W.I.F.T. message standards were agreed, and the physical S.W.I.F.T. network was built and ready for service, by 1977. (Reportedly, work on the S.W.I.F.T. message standards drew, in part, on the work done at Citibank on MARTI.) From going live in 1977, the S.W.I.F.T network grew rapidly, becoming a global network. The larger US banks, including Citibank, joined the network. Also, Citibank, accepting the inevitable, scrapped the MARTI format and converted all its systems to use the S.W.I.F.T. standards, both for external and, ultimately, for internal branch-to-branch transactions.

Throughout the 1980s and 1990s, S.W.I.F.T. extended its standards to cover a wider range of cash and securities transactions. By the late 1990s, "The S.W.I.F.T. Standards" consisted of over thirty bound manuals, occupying about two feet of shelf space.

In the same year that S.W.I.F.T. went live (1977), Citibank launched its cash machine network in the New York area. Although this was not the first cash machine network, it was the largest and technically most advanced. The design of the network was based on a bit-oriented contextual-message scheme that revolutionized thinking about cash machine networks in the banking industry. (Incidentally, the basic informational unit in Citibank's original message standard was four bits, that is, half a byte, which the Citibank technology team referred to as a

"nybble".) This was the first application of structured transactional messages in consumer banking (as distinct from corporate banking). Since then, structured-message standards have emerged for the exchange of cash machine transactions between banks (through cooperative groups like Cirrus), and also between banks and credit card networks (so that credit cards can be used for cash advances at cash machines).

Recent developments

After the airlines and the banks, other industries started taking an interest in structured transactional messaging. During the 1980s and 1990s the list of structured transactional messaging standards grew steadily. Today there are many different standards covering many industries.

The table below summarizes some of the most significant examples of structured transactional messaging standards that evolved since 1971. Note the different formats that have been used for tags in different standards. One might even suggest that the designers of each new standard deliberately tried to use a different arrangement to that of earlier standards in order to leave their mark on the world of structured transactional messaging standards.

Standard	Industry	Year of first use	Number of structural tiers	Example of the tag format
IATA	Airline	1971	1 – "field"	n/a – the standard is largely based on positionally defined fields (with some elementary tagging)
S.W.I.F.T.	Banking	1977	2 – "field", "subfield"	:36B:
ISO 8583	Financial Services (for credit/debit card transaction authorization)	1987	1 – "field"	n/a – this is an untagged standard, using positionally defined fields
EDIFACT	International Goods Trading (ordering, shipping confirmation, invoicing)	1988	2 – "segment", "component data element" A segment may contain several "data elements" which are identified positionally within the segment	QTY+
FIX	Securities Trading	1995	1 – "field"	44=

As more industries started to look at the possibility of exchanging transactions electronically, the implementation problems started to get more difficult. The early adopters of structured messaging, such as the airlines and banks, had approached the problem from the point of view of *automating existing electronic information-exchange processes*, where unstructured or semi-structured messages were transmitted via public and private telegraph and telex networks.

Another characteristic of the businesses of the early adopters was that their transactions were such that fairly simple message structures could be used: each transaction involved a single action, such as crediting an account. By contrast, most of the later adopters faced the task of automating processes that were carried out by the *exchange of paper documents*, such as purchase orders, bills of lading, packing lists, invoices, letters of credit, certificates, and so on. Also, many of the documents in these other industries dealt with transactions that contained multiple actions or parts, such as the supply of several different products under a single purchase order or shipment. This made the development of structured message formats a much more complex task.

To convert the processes of the later adopters from paper to electronic form, it was first necessary to agree, in principle, to exchange the electronic equivalents of these documents. It was then necessary to find standardized ways of capturing and encoding the information contained in these documents. (When you look at the standards developed by the later adopters, the underlying documentary nature of the data that the messages represent is often apparent from the structure of the messages.) Only then could the final step of designing a message standard take place.

In some industries it was also necessary, in developing standards for structured messages, to gain agreement on standardizing, or at least codifying, terms and conditions, so that transactional information could be detached from the small print that normally accompanied it in many documents.

Because of these important differences between the early adopters and the later adopters, progress in moving to structured transactional messaging was a lot slower for the later adopters.

X.409

The X.400 message standards, of which the X.409 standard is a part, deserve mention here, not because you will come across them today (you won't – their life as a standard was a short and unglamorous one), but because they serve as a good counter-example to show how a poorly-designed message standard hampers adoption of the standard. The X.400 standards were defined by the CCITT (Comité Consultatif International Téléphonique et Télégraphique, a standards-definition body which is part of the International Telecommunications Union, or ITU, and which has since been renamed the ITU-T). The X.400 standards dealt with message-based communication and were part of a larger 'X' series of standards (or "recommendations", as the CCITT referred to them) dealing with all aspects of data communication. As the standards-definition process progressed, the 'X' series committees worked towards the goal of defining a complete seven-layer "stack" of data communication protocols, based on the seven-layer OSI model (Open System Interconnection model), which was promoted by the International Standards Organization (ISO), and was thus given the potentially confusing full title of "the ISO OSI model".

Although some of the ideas embodied in the OSI model are still useful, the distinctions between the seven layers have become blurred over time, as the world of data communications has come to be dominated by Internet-oriented protocols and addressing standards, such as TCP/IP, http, ftp, ethernet, and DNS. In fact, this has happened to such an extent that it is now hard for many people to imagine what exactly the problem was in the early 1980s that the CCITT "X-series" committees thought they were trying to solve.

The messages to be covered by the X.400 series included unstructured messages (such as email messages) and structured messages. Recommendation X.409 was the recommendation that specifically addressed structured messaging. It was also conceived as defining the sixth protocol layer of the OSI protocol stack – the "presentation" layer. This was immensely confusing because, at the time of its publication, X.409 was the only layer-6 protocol defined by the CCITT and it was therefore assumed by many people to be *the* layer 6-protocol of the OSI stack. Because of its complete unsuitability for non-message-based communication (for example, for simple file-transfer operations), many designers and programmers working on system-to-system communication problems ignored X.409, even when they were working on applications where it might have had some value.

The confusing dual role of X.409 (as both a structured-message standard and "presentation" layer of the OSI protocol stack) was not the only reason for X.409's lack of adoption. Other reasons included the following:

- The X.409 standard as a whole was appallingly badly written and almost impossible to understand.

- The standard was based on bit-oriented messages, rather than printable messages, for no obvious reason. This made it difficult to lay out useful examples of messages in the standard.

- The example used in the standard – a message containing an employee's personal record – was very poorly presented (even allowing for the difficulties of presenting a bit-oriented format on the printed page). It was also not a very good choice, given that the most promising applications of the standard were likely to be for commercial transactions such as electronic orders, financial transactions, electronic invoicing, bills of lading, and so on.

- The designers seem to have become very confused about the purpose of tags. Rather than treat tags as parts of a message whose values and meanings were to be defined by industry groups (as the overall standard was adapted to different uses), the standard allowed tags to be either normal tags or format descriptors (like "integer", "character string"), *but not both at the same time*. Readers were thus left unsure about how they should be using the tags.

- The X.400 suite of recommendations as a whole was ultimately a failure. The major factor that led to the demise of X.400 was competition from other industry standards. In particular, the addressing concepts contained in X.400 were trampled to death by the stampede to adopt the DNS-based approach for email addressing, with addresses of the form username@companyname.com. This caused vendors who had initially bet on X.400 as the future basis of all messaging to either go out of business or terminate support of their X.400 software products. This, in turn, caused the few customers who had adopted X.409 to discard it along with their X.400-based networks.

The confusion about the role of tags in X.409 was perhaps a result of the dual role which the standard was required to play. One of the goals of the OSI "presentation" layer that seemed important at the time was to create a *linga franca* for computer applications, so that applications running on different hardware, using different operating systems, and written in different programming languages, could reliably exchange data. In the early 1980s, the software industry was very heterogeneous, with the result that the need for format conversion of data

transmitted between two systems was a common problem. While the problem persists to a degree today, it is now much less of a problem. The emergence of PCs, and the subsequent dominance of Microsoft's software, has removed many of the obstacles that existed up until the early 1990s to transferring data values between different applications and systems. Also, the general acceptance of printable-message standards has removed most of the problems that X.409 created for itself in trying to transfer numerical values as bit-strings. As a result, today we are left wondering "What was this presentation-layer thing anyway?"

XML

A more recent development in transactional message standards is the adoption of XML as a basis for defining the structure of messages. XML, which stands for eXtensible Markup Language, was originally developed to meet the needs of large-scale electronic publishing, particularly publishing on the World Wide Web. XML is a simplified version of SGML (Standard Generalized Markup Language), an international standard (ISO 8879) for defining the structure of different types of electronic document. XML allows organizations to create their own customized markup applications for exchanging information within their particular discipline (music, chemistry, electronics, geology, linguistics, genealogy, and so on). XML does this by allowing the user to assign names to the various data elements in the document. These names are the equivalent of tags in traditional structured messages.

XML is particularly useful in the World Wide Web as an alternative to HTML, because it allows a website to make information available to its visitors in a way that allows the visitor to use software that programmatically extracts information from a page. For example, suppose that the user wants to use a program that compares the prices of several items across many different websites. If the pages of those websites are encoded using XML, the user's program can reliably extract the values of the fields "product name" and "price", because these values are "tagged" with the names "product name" and "price". This is not possible with an HTML-encoded page, which simply assigns places on the screen to each string of characters. While it is technically possible to create programs that "guess" which character strings represent the product name and which character strings represent prices in an HTML page, the operation of such programs can never be completely reliable: sometimes they will extract the wrong character strings.

Hearing that XML is now a big thing, a number of standards bodies have either started adapting their message standards to XML or have declared their intention to do so. However, it is not clear why this is useful. Essentially, all that is achieved by conversion to XML is the replacement of already-agreed tags by XML names. In most cases the original tag values will simply be declared to be XML names. This does not in any way increase the usefulness of the structured messages, nor does it reduce the amount of work that needs to be done to get a system ready to handle a new additional message type that has been added to an existing standard by the standard-enhancement committee.

The biggest problem with XML is the inflated expectations that software vendors have generated for XML. It is an excellent idea to tag data on a website, so that it can be easily read by a program, as well as by a human being. However, a common misconception about an XML document or webpage is that it can be "self-defining". Tagging the data does not make the data self-defining. To illustrate this point, suppose that I send someone an XML document in which I assign, and use, the names "SloduShok", "FroobleNawk", and "KoolPukak", for three kinds of data element. It really does not help the recipient of the document that every data element

value is tagged with one of these three names – unless I have talked to the recipient beforehand and come to an agreement about what these terms mean. Of course, this is an extreme example. You could argue that using XML names like “product name”, “manufacturer”, and “price” is unlikely to lead to a misunderstanding. However, when you start to look at information that might be exchanged between organizations doing business with one another, it becomes apparent that things are never as clear-cut as you think they are going to be. The S.W.I.F.T. standards are an excellent illustration of this point. As mentioned earlier, the complete set of S.W.I.F.T. message standards occupies about two feet of shelf space and has been developed over a period of three decades. This enormous set of documents is mainly required *not* to assign tag values and define message types (a fairly simple task), but rather to unambiguously record the results of years of analysis and debate as to exactly what each field means, and what business rules apply to the handling of each type of transaction. The non-trivial nature of these aspects of a comprehensive standard is obvious if you pick any section of the S.W.I.F.T. standards at random and start to read it.

The same need for agreement between parties, on what each field or data element really means and on what business rules apply to its use, is present in all industries. The complexity of the task in many industries is even greater than in banking. This is not to say that simple pilots of a single transactional process cannot be started fairly quickly. However, for a particular industry or profession, the complete task of gaining agreement on, and recording in a document, (a) the meaning of all transactional data, and (b) the way it is to be processed, is one that will take many years. In some industries, the resulting document may well occupy *several* bookshelves, and not just one bookshelf like the S.W.I.F.T. standards.

In summary, the move away from short tags like “:36B:” to XML-style tags, composed of descriptive text like “SecuritiesOrderToBuyQuantityOfFinancialInstrument” certainly does no harm. With bandwidth so plentiful and cheap today, it cannot be objected to on the grounds of wasting bandwidth. However, it could be objected to on the grounds that it may give the false impression that the exact meaning of the data element, in all relevant transactions, has been agreed by all parties and properly recorded, where this may not, in fact, be the case.

Conclusion

The use of standardized structured transactional messages is critical to a broad range of commercial activity today. The volumes of business that are conducted electronically using such standards is now so great that a return to manual processing of unstructured messages, even if transmitted electronically, is both impossible and unimaginable.

The principles underlying all structured transactional message standards as established in the 1970s underlie even the most recent standards. These principles include:

- The concept of message envelopes (comprising message headers and message trailers)
- Message bodies
- Message type codes
- Field tags (or positionally identified fields)
- Field formats
- Field inclusion and repetition rules
- Message acknowledgement rules

The recent XML-driven movement towards using tags consisting of one or several words strung together (for example, “SecuritiesOrderToBuyQuantityOfFinancialInstrument”), instead of compact numeric or alphanumeric strings (such as “:36B:”), does not obviate the fundamental requirement that sender and receiver agree on the precise meaning of the information that will be inserted in the field, and the business rules that govern the processing of that information as part of the transaction. Reaching such agreements between the many global stakeholders in each business sector remains a formidable task – particularly for sectors that have taken no prior steps towards elimination of processes based on exchange of pieces of paper. The notion that XML allows field tags to be created that are “self-defining” is a seductive and potentially dangerous notion if it is believed by those responsible for gaining agreement on a new set of structured transactional message standards for a critical business activity. Belief in the “self-defining” notion will certainly lead to the amount of hard work required to reach an understanding between all interested parties being grossly underestimated and is likely to doom the project from the start.