

Introduction to MDMI

The global financial industry exchanges huge amounts of electronic information. Differences in understanding and interpretation of exchanged electronic information form an important barrier to the effective and efficient execution of financial business processes. This leads to processing errors and delays that may have a serious impact on the business and operational capacity of financial institutions. The potential impact within a financial institution includes reduced STP rates, missed business opportunities due to longer time-to-market, and limited agility and higher costs for new product development and testing. On the scale of the entire financial industry, the risks are even higher as misunderstanding between institutions may lead to unnoticed transactional errors – a “semantic coherence” problem.

Because there is little hope that all semantic coherence problems can be overcome with any natural language, the financial industry has created other, more formal ways to exchange business information, such as standardised message types¹. It can reasonably be expected – and history has proven – that the use of these message types results in more predictable behaviour which promotes automation, lowers error rates, increases straight-through-processing (STP), and hence reduces costs for the global financial industry and their clients. This partially explains the success of industry standards like SWIFT, X.9, FIX and FpML.

While standardised messages reduce the semantic coherence problem, these have not totally solved it. Interpretation issues still exist, especially between different standards (e.g., *what is the equivalent FIX data field for the SWIFT data field 32A*), or within a standard (e.g., *is the meaning of data field X exactly the same in all versions of all message types where it is used*). A standard like ISO 20022 is another valuable step toward solving the semantic coherence problem because it establishes a single data dictionary. This dictionary gives each data field item a business-oriented definition. Even so, there are still gaps as most existing standards are not (yet) linked to ISO 20022 and as the detailed meaning of a dictionary item may be further impacted by the context of the message type it is used in.

The next step may come from work undertaken by the Finance Domain Task Force (FDTF) of the Object Management Group (OMG). The task force has recently developed and approved a specification called "Model Driven Message Interoperability" (MDMI). This specification proposes a formal approach to establish semantic equivalence (“semantic interoperability”) between dictionary elements and corresponding fields in international and domestic standards and even proprietary internal formats. The specification builds upon existing financial and technical standards, such as ISO 20022 and OMG’s Model Driven Architecture (MDA), and Unified Modelling Language (UML). MDMI also applies concepts from other standards like ISO 11179 and UN/CEFACT Core Component Technical Specification (CCTS).

Although the basic principles of MDMI are related to the approach that individual financial institutions are taking, there are good reasons to believe that the introduction of

¹ A standardised message type is the result of an agreement on the content of the message (i.e. the message elements) and on the structure and representation of the information (i.e. the message format).

MDMI can bring added value to the financial industry by resolving the semantic coherence problem.

What are the basic concepts of the MDMI specification?

The MDMI specification is based on establishing equivalence relationships between the basic concepts (Figure 1). All relationships will be formally expressed. This will either be done through modelling techniques (e.g., use UML associations), or through the use of rules², that are expressed in a formal language and linked to UML artefacts.

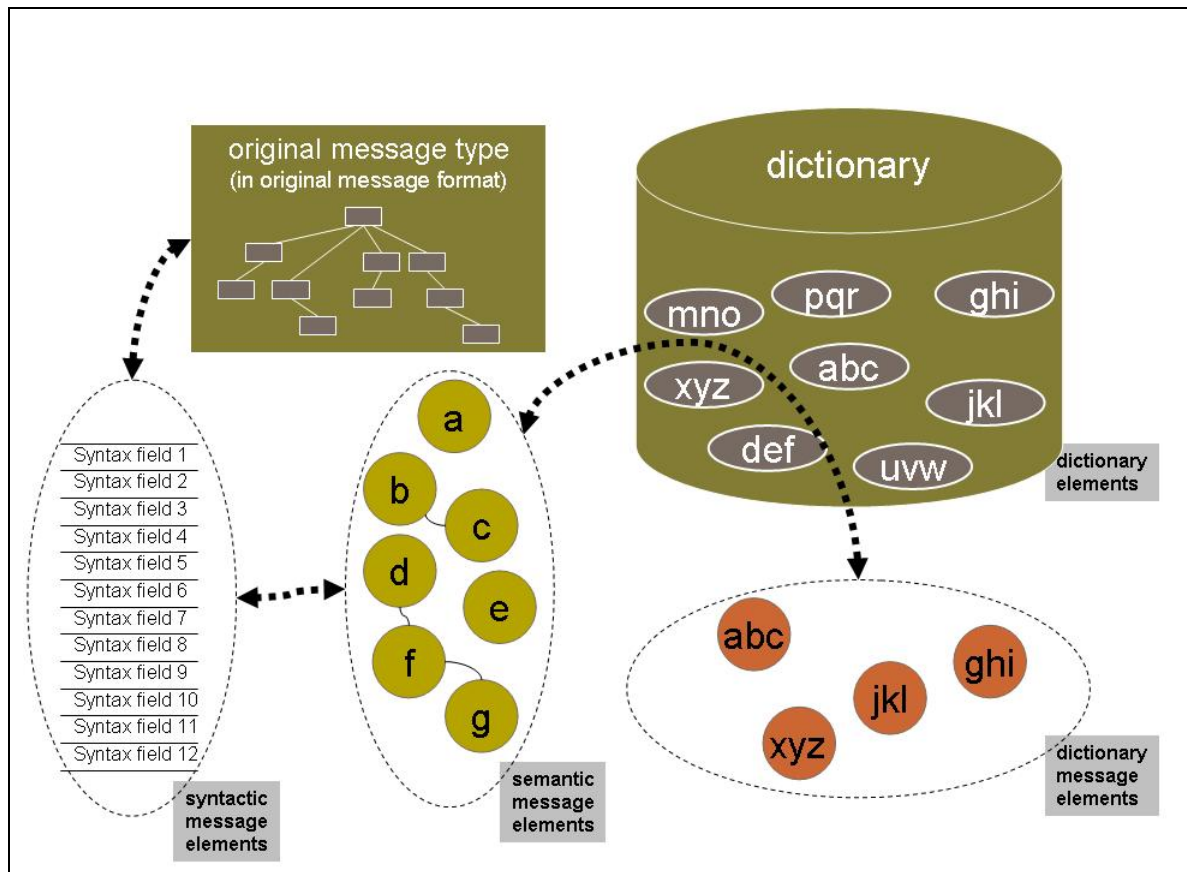


Figure 1 - MDMI basic concepts

The MDMI specification creates uniformly consistent financial information by rigorously defining and applying these concepts:

- **Syntactical message elements:** The syntactical message elements are defined by making the position, representation and other relevant format information of each leaf field (= field at the lowest level in the original message type) explicit. This conversion is based solely on the message format; that is, it excludes any meaning of data contained in each leaf field.

For example, a syntactical message element is technical information about a data field in a message. Suppose the message is a retail card payment message defined by ISO standard 8583. One such data field in these messages is the nature of a merchant's business who accepted a retail payment card. This data field is a numerical code. It

² The MDMI specification does not include the specification of rule languages. Instead it will rely on the reuse of suitable existing standardised rule language(s) by describing the requirements that must be met.

has length four bytes. Each byte must be an integer value taken inclusively from the range 0 through 9. These four integers must be coded using the EBCDIC or ASCII electronic character data sets. The four character value thus provided in the data field must be one of a list of ISO approved codes.

The result is a set of elements that reflect the physical content of the original message type. The stored information makes it possible to reconstruct the original message.

- **Semantic message elements:** The semantic message elements are created by combining syntactical message elements into the smallest possible semantically meaningful items (= items that make business sense). This is done without taking into account the content of the dictionary. For each semantic element the relevant information about this element is made explicit. This includes for instance the value and the related syntactical message element(s).

For example, the retail payment message code discussed above describes the nature of a merchant's business. This includes department stores, automobile fuel filling stations, jewellery stores, hotels, restaurants, or airlines, to name only a few of possibly hundreds of business types.

The result is a set of elements that reflect the full business information contained in the original message type. The stored information makes it possible to reconstruct the related set of syntactic message elements.

- **Dictionary elements:** The dictionary must contain all elements that cover the semantic meaning of each identified semantic message element. Where necessary, new dictionary elements will be created – according to the dictionary rules – to cover missing semantic meaning. It will therefore be possible to link each semantic message element to a dictionary element, although the relationship will not necessarily be a one-to-one relationship (i.e., multiple semantic message elements may be combined in a single dictionary element).

For example, the dictionary would include definitions for several of the terms used in describing the nature of a merchant's business in a retail payment message. Hence, the dictionary must define, "merchant", "type of business", "services business", "retail business", and related terms.

The role of the dictionary³ is to form the "glue" between equivalent semantic elements coming from different message types (regardless of the original message format). To this end the dictionary elements must have a sufficient semantic precision and must be – as unambiguously as possible – distinct from one another⁴.

The result is a set of dictionary elements and the relationships with all semantic message elements.

- **Dictionary message elements:** The end goal is to use the information of the semantic message elements and related dictionary elements to build a set of "dictionary message elements". This set will cover the full business information contained in the

³ The MDMI specification does not include the specification of a data dictionary. Instead it will rely on the reuse of a suitable existing data dictionary by describing the requirements that must be met. It is the hope and intention to reuse the ISO 20022 data dictionary for this purpose.

⁴ This property is called "semantic distance" in the MDMI specification. It must ensure that MDMI users will systematically select the correct dictionary elements for all identified semantic message elements.

original message type and have a one-to-one correspondence with the set of dictionary elements. This means that each dictionary message element covers the same semantic meaning as a dictionary element. Compared to the dictionary element the dictionary message element contains additional information extracted from the original message type, such as the value, relationships with other dictionary message elements and constraints.

For example, the dictionary message element for the retail payment message would be the combination of ...

The result is a set of elements that reflect the full business information contained in the original message type in a fully dictionary-compliant way.

How does MDMI work?

Very simply, the MDMI specification relies on each owner of a set of message types to develop “MDMI sets”, i.e., sets of syntactical message elements, semantic message elements, and dictionary message elements for all the message types it owns. The term “message type” should be understood in its broadest context and can refer to an international or domestic message type, or internal messages used by a financial institution.

MDMI sets typically will be used according to the following models:

1. The MDMI sets can be combined to support end-to-end conversions; e.g., one can translate FIX pre-trade information into ISO 15022 post-trade information by using the MDMI sets provided by FPL (for FIX message types) and by SWIFT (for ISO 15022 message types).
2. The MDMI sets can be used to ease versioning issues by allowing information translation between different versions of the same message type.
3. The MDMI sets can be complemented with internal MDMI sets; i.e., MDMI sets created by a financial institution to describe its proprietary internal format(s), hence allowing conversion between external and internal formats.

Is MDMI different from traditional middleware?

The basic principles of the MDMI specification are not (very) different from middleware-software that is used today by most financial institutions. Indeed, the problem of dealing with multiple formats is not new to the financial industry. For decades financial institutions have been exposed to various domestic and international message formats. They have built custom solutions to deal with these many and varied formats. In the end custom solutions always boil down to the definition of “conversion rules” between the external and internal formats. As most financial institutions have multiple internal data formats, many institutions try to develop customized intermediate data formats that are supposed to buffer external and internal message formats. These intermediate data formats are supposed to follow the “hub-and-spoke” model, where each message format is converted only once, from the original format into a common intermediate format. Contrast this hub and spoke model with the “peer-to-peer” model where each format is translated as often as it is used.

MDMI takes hub-and-spoke model and standardizes it by introducing some important differences that add value versus either the peer-to-peer or the hub-and-spoke model:

1. The “technology” of the custom hub-and-spoke model is developed internally by each financial institution or middleware vendor, leading to a multitude of proprietary structures and languages to describe formats, positions, constraints and conversion rules.

MDMI is based on industry standards, like MDA (Model Driven Architecture) and UML (Unified Modelling Language). It defines a common structure and open, standardised ways to express formats, positions, constraints and conversion rules. This make it possible for owners of message types to create MDMI sets once and to reuse them globally.

2. In the custom hub-and-spoke model, each financial institution creates the conversion rules for all formats it uses. It must therefore rely on in-house knowledge or (expensive) consulting regarding the various external standard formats.

In MDMI each owner of a message format can define its conversion rules and make these publicly available. In combination with the use of standardised technology, that is “machine readable”, MDMI will allow each financial institution to import and reuse an official set of rules for each external message format. This approach will improve message interpretation and reduce time and costs to implement changes to external formats.

3. Custom hub-and-spoke models are often geared towards “transient conversions”; i.e., information is directly transformed from a source format into the internal format for immediate consumption by target applications. The intermediate format – which has been used for the definition of the conversion rules – may or may not be created during the actual conversion but is never available to other potential users.

The transient source-to-target conversion methodology can also be used with MDMI and may even be required for performance reasons. In addition, MDMI can easily support a two-step conversion allowing persistent storage of the information in the intermediate format. This opens the door to reuse and other value-added processing, like enrichment (e.g., use a received identifier to extract and store name and address information from an external source) and a business transaction approach (e.g., collect and combine information coming from multiple input messages).

4. The custom hub-and-spoke model is often based on a full-message approach; i.e., the rules are defined for complete message types and rely on availability of a complete message instance.

MDMI takes an element-oriented approach. This makes it possible to extract useful information from a partially available message that adds flexibility.

5. Financial institutions will often create internal message types for specific purposes especially when no standardised message types exist. It is however very hard to use these message types for other (external) purposes.

MDMI makes it possible to externalise customised message types by providing – in addition to the message type – the related MDMI sets.

Some examples of using MDMI

1. One-to-one message translation

MDMI can be used to translate a complete message instance into another (equivalent) message instance. The other message instance may have another message format or may be another version of the same message type (in the same message format).

Example:

SWIFT is gradually moving from its traditional message format to ISO 20022 XML. It has recently developed a set of translation rules for credit transfer messages: e.g., MT 103 to/from MX pacs.008, and for investment funds; or MT 502 to/from MX setr.010. These translation rules have been defined in a human readable format (in Excel tables) and in a machine readable format (based on XSLT script and Java code).

These rules can also be defined in MDMI⁵ in which case there will be a separate MDMI-set for each MT and MX. The actual translation can be done in a two-step approach; i.e., MT 103 → intermediate format → MX pacs.008. This is a straightforward implementation of the defined MDMI sets for MT 103 and MX pacs.008. Alternatively – and preferably – the translation can be done in a single step; i.e., MT 103 → MX pacs.008, by implementing the combined information from the two MDMI sets.

The preference for a single step approach is purely driven by performance considerations and by the fact that the ultimate goal of this particular use case is to have a complete message translation. This single step approach will mimic the current set of direct MT 103 to MX pacs.008 translation rules and it is thus easier to derive the single step translation from these rules. Direct translation rules can however not be reused for other purposes, e.g., when we also need to develop the MX pacs.008 to Fedwire CTR translation rules.

2. Dealing with multiple versions of a message type

MDMI can be used to protect internal applications from changes in subsequent version of message types. This protection is limited and will not totally protect a financial institution from the need to provide or use new information (but even in the latter case the impact will be reduced by the use of MDMI).

Example:

There are multiple versions of FIX message types in use and financial institutions that are FIX users may need to deal with counterparts using different versions. Suppose that a user receives Indication of Interest (IOI) messages from various counterparts. Initially all are based on FIX version 4.2 but due to the release of FIX version 5.0 some counterparts want to start using the IOI v5.0. According to the FIX documentation there is a significant difference between these versions of the IOI, so there could be a significant impact on the financial institution.

The financial institution will have a single application that deals with incoming IOI messages and it will originally have two MDMI-sets: one for the FIX v4.2 IOI and one for the internal format used by its application⁶.

⁵ Note that this assertion still needs to be verified. It is the subject of a Proof of Concept that is currently executed by the MDMI consortium.

⁶ Note that – in theory – the application might also use FIX IOI as its internal format.

In order to deal with the new incoming version the institution needs an additional MDMI-set for the FIX v5.0 IOI. As the MDMI sets for both FIX versions of the IOI are developed by FPL (as owner of these message types) the workload to include this new MDMI-set is limited. If the financial institution does not need new information from the new IOI version there will be no impact on the internal MDMI-set. Otherwise the institution will only have to adapt its MDMI-set with respect to the new information to be used. In all cases the workload under these circumstances is far less than in the traditional model where the financial institution has to develop the conversion rules between the new FIX version and its internal format.

3. Multiple-to-one message translation

MDMI can be used to support the behaviour where one needs to extract information from multiple incoming messages before creating an outgoing message or before using the information in an internal application. It should be noted that this will require some additional logic beyond MDMI to trigger – at the correct time – the generation of the outgoing message or the internal application.

Example⁷:

Suppose that an institution decides to send some type of information based on input coming from multiple data providers. The data providers are using various message formats and the related MDMI-sets have been defined by the owning standards organisations.

The institution will use these MDMI-sets to store the received information in a persistent way and to group information that is related to the same subject, even when it is coming from various sources.

In the end the relevant application will have access to all received information.

4. Value-added processing

MDMI can make it easier to do value-added processing on information that has been extracted from incoming message instances.

Examples:

Suppose that an institution receives an MT 103 that contains the BIC code of the creditor's bank. The MDMI-set can be used to store this BIC code in a persistent way and additional information, eg, name and address, can be extracted from an internal database and stored with the BIC code. This will make it easy for any application or outgoing message to include – where needed – the name and address information, without requiring every application to access the same database.

Suppose that information in the source message is provided in a character set that is not supported by the internal applications of an institution. In that case the original information can still be stored and forwarded to a next party in the chain, while the internal applications use a converted form of the information. This solution is being considered in the SEPA regain to deal with Greek and Cyrillic characters (but can easily be extended to other characters).

⁷ A better (more realistic / more concrete) example would be useful

