Date: September 2010

Model Driven Message Interoperability (MDMI)

Version 1.0

OMG Document Number: formal/2010-09-01
Standard document URL: http://www.omg.org/spec/MDMI/1.0
Associated File*: http://www.omg.org/spec/MDMI/20090901
http://www.omg.org/spec/MDMI/20090902

---

* original file: dtc/2009-09-15 (EMOF), dtc/2009-09-16 (mdxml)

[Supersedes formal/2010-03-01: added associated file]

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 140 Kendrick Street, Needham, MA 02494, U.S.A.

TRADEMARKS

MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™ , Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™ , IMM™ , MOF™ , OMG Interface Definition Language (IDL)™ , and OMG SysML™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page *http://www.omg.org*, under Documents, Report a Bug/Issue (http://www.omg.org/technology/agreement.htm).

Table of Contents

Preface v

1 Scope 1

2 Conformance 1

3 Normative References 2

4 Terms and Definitions 2

5 Additional Information 3

5.1 Acknowledgements 3

6 Overview 5

6.1 Relationship to ISO 20022 5

6.2 Different Ways to Use the Current Specification 6

6.2.1 Moving Data from One Message to Another 6

6.2.2 Versioning 6

6.2.3 Moving Data from an Internal Enterprise Message Format to an External Standard 7

6.2.4 Bilateral Mapping 7

6.3 Basic Approach for the Use of This Specification 7

6.3.1 Stage 1 7

6.3.2 Stage 2 7

6.4 Future Benefits of the Specification 8

6.4.1 Dealing With (Near) Synonyms 8

6.4.2 Mapping between Data Dictionaries 8

6.4.3 Handling Lossless Conversion 8

7 Use of MDMI Artifacts Overview 9

7.1 Informal Overview of Artifacts 9

7.1.1 Step 1 - Remove the Syntax 10

7.1.2 Step 2 - Mapping a Source Semantic Element to a Target Semantic Element through the use of a Unique Identifier acquired from a central dictionary 11

8 UML Semantics - Normative Definition 13

8.1 MessageModels, MessageGroup, MDMIDictionaryReference 13

8.1.1 Overview 13
8.1.2 Abstract Syntax 13
8.1.3 MessageModel - Detailed Semantics 13
8.1.4 MessageGroup - Detailed Semantics 14
8.1.5 MDMIDomainDictionaryReference 15

8.2 MessageSyntaxModel, Node, Bag, Choice, LeafSyntaxTranslator 15

8.2.1 Overview 15
8.2.2 Abstract Syntax 16
8.2.3 MessageSyntaxModel - Detailed Semantics 16
8.2.4 Node - Detailed Semantics 17
8.2.5 Bag - Detailed Semantics 18
8.2.6 Choice - Detailed Semantics 18
8.2.7 LeafSyntaxTranslator 19

8.3 SemanticElementSet, SemanticElement SimpleMessageComposite, MessageComposite, Keyword 19

8.3.1 Overview 19
8.3.2 Abstract Syntax 20
8.3.3 SemanticElementSet - Detailed Semantics 20
8.3.4 SemanticElement - Detailed Semantics 21
8.3.5 Keyword - Detailed Semantics 23
8.3.6 SimpleMessageComposite - Detailed Semantics 23
8.3.7 MessageComposite -- Detailed Semantics 23

8.4 MDMIDatatype, DataRules 24

8.4.1 Overview 24
8.4.2 An Example of Complex Datatype 24
8.4.3 MDMIDatatype, DataRules - Abstract Syntax 26
8.4.4 MDMIDatatype - Detailed Semantics 26
8.4.5 DataRules - Detailed Semantics 26

8.5 MDMIBusinessElementReference, Conversion Rule, ToSemanticElement, To BusinessElement, MDMIBusinessElementRule 27

8.5.1 Overview 27
8.5.2 Abstract Syntax 28
8.5.3 MDMIBusinessElementReference - Detailed Semantics 28
8.5.4 ConversionRule - Detailed Semantics 29
8.5.5 ToSemanticElement - Detailed Semantics 29
8.5.6 ToBusinessElement 30

8.5.7 MDMIBusinessElementRule 30
8.6 SemanticElementRelationship 31
8.6.1 Overview 31
8.6.2 Abstract Syntax 31
8.6.3 SemanticElementRelationship - Detailed Semantics 31
8.7 SemanticElementBusinessRule 32
8.7.1 Overview 32
8.7.2 Abstract Syntax 33
8.7.3 SemanticElementBusinessRule - Detailed Semantics 33
8.8 Summary of Complete Metamodel 34
8.8.1 Overview 34
8.8.2 Abstract Syntax 34
Annex A - List of Acronyms 35
Index 37

Preface

About the Object Management Group

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at *http://www.omg.org/*.

OMG Specifications

As noted, OMG specifications address middleware, modeling, and vertical domain frameworks. A catalog of all OMG Specifications is available from the OMG website at:

http://www.omg.org/technology/documents/spec_catalog.htm

Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM)

Platform Specific Model and Interface Specifications

- CORBAservices

- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications.

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
140 Kendrick Street
Building A, Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult *http://www.iso.org*

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

Note – Terms that appear in *italics* are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to *http://www.omg.org/technology/agreement.htm*.

1 Scope

Complete financial transactions often involve multiple steps that require the transmission of information across financial systems in multiple enterprises. Each step of a transaction usually relies on the transmission of information via standardized messages. Some examples of standardized message formats utilized in financial services are MDDL, FIX, FpML, IFX, TWIST, SWIFT messages, Visa messages, RosettaNet, OAGi, ACORD, and CIDX. Each of these standards provides a particular type of functionality within the financial service industry. For example, FIX deals with front-office transactions in the securities sector, while a certain group of SWIFT messages will deal with back-office security transactions, such as clearing and settling, in the same sector. Each set of financial message standards is usually supported by a separate industry standards body, e.g., SWIFT for SWIFT messages, Visa for Visa Messages, the FIX Protocol committee for the FIX standard, etc. The messages created by these groups have evolved over many years with little or no coordination between groups.

To get true Straight Through Processing (STP), information must be correctly interpreted and processed by each involved financial system at each step of the financial transaction. This implies, among other things, that information must be accurately moved from one system to the next. This may require moving information from one message format to another, e.g., from a FIX pre-trade message into a SWIFT settlement message. In addition, a financial institution will often have its own internal data elements used either in internal data stores or in internal messages. These internal data elements must also be appropriately mapped to and from the industry standard messages if information is to be transmitted from one institution to another. Currently, the mapping of financial data from one format to another is not standardized. The mappings are usually done in an ad hoc procedural manner. The complicated and complex maze of existing formats and hard-coded transformations has created an environment where every introduction of new message formats, and even changes to older messages, is very expensive. The goal of the current specification (MDMI) is to provide a declarative, model-driven mechanism to perform message data transformation - not only to handle the movement of data between different message formats, but also to support versioning by providing a mechanism to map information between a new and an older version of the same message. Thus, the current standard can help reduce the barriers that prevent the introduction of new versions of messages and thereby greatly reduce the cost of change.

The Finance Domain Task Force wishes to emphasize that this specification is intended for use by the financial services community, and has been developed with its specific needs and requirements in mind. While it can certainly be envisioned that the concepts, models, and mechanisms described in this specification can be applied or adapted to other application domains, it is not the intent of this document to cover other than the financial services domain.

2 Conformance

To be compliant with the specification, an implementation would need to be able to create the artifacts that are shown in the model specification (OMG document dtc/2009-09-09); to utilize expression languages that are consistent with the constraints described in Section 8.1.4, “MessageGroup - Detailed Semantics” to utilize MDMIDatatypes that are consistent with the description and constraints in section 8.4, and to utilize a central dictionary that provides a function delivering a unique identifier as described in section 8.1.5. In addition, an implementation needs to support a runtime application, as described in Figure 7.1 and Figure 7.2. See Section 7.1, “Informal Overview of Artifacts” that can consume the generated maps and match unique identifiers to provide a transformation of a Semantic Element from a source message to a target message.

3 Normative References

This specification references ISO 20022. A complete reference for ISO 20022 can be found at www.ISO20022.com.

4 Terms and Definitions

Business Element

A Business Element is the smallest semantic unit in an external dictionary. For example, in ISO20022 Business Elements are the attributes of Business Component (or their related Message) classes and represent a "business concept."

Composition

A configuration of related entities that results in a new entity at a different level of abstraction that is, a composition is a grouping of two or more entities that can be referred to as a single entity at a different level of abstraction from its component entities.

Conversion Rule

A rule that describes the conversion of a value of a source Semantic Element into a value of a target Business Element or a target Semantic Element.

Datatype

A reusable prescription of the format of a data value that has no specific message format related semantic content, for example an address, a date, etc.

Federated Dictionary

A collection of physical Domain Dictionaries, whereby each Domain Dictionary contains Business and Semantic Elements that are relevant to a particular domain of the financial industry and whereby the collection of all Business and Semantic Elements represents a single logical Domain Dictionary for the financial industry.

Message Format

Definition of the syntax and semantics of a class of messages. Can be defined in many ways including paper documentation.

MXxx

Message format developed according to the ISO 20022 specification.

MTxx

Message format developed according to the SWIFT EDI specification, including the ISO 15022 messages.

Near Synonym

A Semantic Element that can be derived using prescribed mapping rules from a set of other Semantic Elements, thus lying within a clearly bounded semantic distance from those Semantic Elements.

Physical Message Instance

An instance of a message that is used to transmit information from a source to a target application.

Semantic Element

An entity in a message format that represents a “smallest” business concept specific to that message format. The easiest way to describe is by analogy. If the information in a message were used to define a denormalized table in a database table, then the Semantic Elements would represent the columns of that table.

Semantic Element Set

A set of SemanticElements, MessageComposites and Simple MessageComposites and SemanticElementRelationships that represent the semantics contained in a message format.

Semantic Map

A map that describes the relationship between a Semantic Element in a Semantic Element Set and a Business Element in a Domain Dictionary or between a Semantic Element in one Message Model and a Semantic Element in another Message Model.

Synonym

A Semantic Element that can be mapped to another Semantic Element by simple equivalence, i.e., A=B.

TCxx

Message format developed according to the VISA EDI specifications for retail banking applications.

5 Additional Information

5.1 Acknowledgments

The following companies submitted and/or supported this specification:

- Adaptive
- FireStar Software, Inc.
- IBM Corporation
- Informatica Corporation
- IP Commerce
- Visa International, Inc.

The OMG Finance Domain Taskforce wishes to acknowledge the contributions of the primary author of the specification, Mark Eisner and the support given him by Joseph Bugajski; and the key contributors, Said Tabet, Gabriel Oancea, David Frankel, and Christian Nentwich for their work in refining the specification. We would like to acknowledge Kris Ketels and Frank Vandamme for their careful and constructive review of the materials. The authors also would like to make a special acknowledgement for Pete Rivett for very careful review and suggestions for both the documents and the specification and for Sridhar Iyengar for his patience and guidance.

6 Overview

Given the lack of a financial industry-mapping standard, data is usually mapped directly from one message format to another. It is a well-known principle in the field of system architecture that as the number of interfaces in a “system” increases linearly the cost of maintaining point-to-point mappings increases geometrically. In addition, since many of these mappings are done locally and procedurally, errors are easily introduced. All financial organizations face this situation. Certainly, financial organizations spend a good deal of their software development budget on developing new interfaces and mappings or extending existing ones. In addition, it is very hard to introduce any changes into existing message formats or introduce new formats because of the tremendous cost of changing applications that process the older message formats.

The goal of the MDMI specification is to provide a standard framework and methodology for the financial services industry, which will alleviate the mapping problem.

This specification will:

- Reduce significantly the cost and time needed to define conversion rules to map data from one message format to another.
- Handle versioning issues as particular message formats evolve over time.
- Allow the expedited adoption of new standards - as mapping the new standard to the existing standard will allow applications to continue to use the legacy standards thus greatly reducing the introduction cost of new standards.
- Improve the interoperability and STP in end-to-end financial transactions that are based on multiple message formats.

The MDMI specification’s framework is based on two concepts:

1. First, removing any syntax associated with a message format, revealing the set of core “Semantic Elements” contained in that message format. A Semantic Element is the smallest semantic unit defined in a message format.
2. Second, specifying a semantic map of those Semantic Elements to an industry accepted Domain Dictionary made up of “Business Elements.” A Business Element is the smallest semantic element that is an entry in the dictionary. Business Elements represent a business concept for the industry sector.

The easiest way to recognize Semantic Elements or business elements is that they cannot be constructed from other Semantic Elements or business elements, respectively, i.e., they are represented by a class, whose value property is a data type.

Providing semantic maps to a central Domain Dictionary creates a “hub and spoke” approach to mapping as each standards body need only develop maps to the standard Domain Dictionary. A complete mapping for a Semantic Element then will have two maps, a map from a source to a Business Element in the Domain Dictionary and a map from the same Business Element in the Domain Dictionary to the target. Thus, the mapping process is reduced from being geometric to being linear with the number of message formats.

6.1 Relationship to ISO 20022

To be effective, there needs to be an industry-wide consensus on the semantic content of the business elements in the Domain Dictionary and there needs to be an organization that will take on responsibility for maintaining its integrity. In the financial services industry the responsible organization is TC68 and its working groups as outlined in part 2 of the ISO 20022 standard.

In the ISO 20022 Domain Dictionary, Message Elements, which are properties of Message Components (where Message Components, in turn, are related to Business Components), are the equivalent of the Business Elements as defined in this specification. Thus, Semantic Elements can be mapped to the Message Elements in ISO_20022.

Examples of Message Elements:

- The amount in a client's retail bank account
- The name of a bank branch
- The name of the sender of a wire transfer

6.2 Different Ways to Use the Current Specification

6.2.1 Moving Data from One Message to Another

The primary focus of the MDMI specification is moving some information from a source message in a message format that has been defined by one standards body to a target message in a message format independently defined by another standards body utilizing an industry defined central Domain Dictionary.

For example:

One message format may define a "client address" field while another message format may have separate fields for "client street," "client city," "client state," etc.

One message format may define a bank ID number as a BIC number while another message format may define a bank ID as an ABA routing number.

The key is that the fields in each message are mapped to the same central dictionary element. There are two important benefits of mapping to a central Domain Dictionary such as the ISO 20022 Repository:

1. The central dictionary creates a hub and spoke architecture for transformations. Therefore, only a linear set of transformations must be created among different message format groups instead of the n2 mappings required for bilateral transformations. For example, by using a central Domain Dictionary for payments, only six maps need to be created to map payment information among SWIFT MT messages, SWIFT MX messages, FIX messages, Visa TC messages, RosettaNet messages, and ACH messages, whereas 15 bilateral conversion maps would be needed.
2. Given that a standards body or enterprise takes responsibility for creating standard conversion maps to a central dictionary, it need only be expert in its own message formats and the well-defined semantics of the central Domain Dictionary, rather than needing to understand the semantics and syntax of many other message groups if the bilateral element method is employed.

6.2.2 Versioning

A second costly problem in the financial services space is versioning. The market continually requires changes in message formats. Given the legacy of existing software, even a small change in a message format can be prohibitively costly to implement. Thus, required changes are often implemented very slowly and, in the worst case, not implemented at all. By providing MDMI maps between new versions and older versions, new message formats can be introduced without requiring that existing message formats be abandoned or that legacy applications be re-coded, as long as the legacy applications do not utilize the new information in the new version.

6.2.3 Moving Data from an Internal Enterprise Message Format to an External Standard

Another important value of MDMI is moving information from an enterprise's internal message or data formats to an external message standard. It is important to note that a record definition in a database schema can be considered to be a "message format" and maps can be generated that transform data from that internal database to an external standard. Currently large staffs are devoted to creating bilateral maps between their internal standard and the external standard. Whenever either message format changes, these maps must be changed. With MDMI maps, the Semantic Elements in internal message formats are mapped to a central dictionary, such as the ISO 20022 Domain Dictionary. Given that a standards body, such as SWIFT, distributes new MDMI maps to account for the change in their standard, then the internal enterprise maps do not have to be changed. This will result in very significant savings.

6.2.4 Bilateral Mapping

MDMI can be used to model and define conversion maps directly between two message formats. In this case, the semantic mapping is between the Semantic Elements in a source message format and the Semantic Elements in a target message format. (The ConversionRules, which define the relationship between Semantic Elements must be as complicated as required to accomplish a mapping whereas conversion rules mapped to a central dictionary will have a restricted set of operators.)

6.3 Basic Approach for the Use of This Specification

The artifacts defined for this specification are designed to map data (i.e., sets of Semantic Elements) from one message format to another rather than the wholesale conversion of a complete message in one message format to another message format. With this focus, each data field conversion needs to be atomic, containing all the meta-data necessary to move the data in the field to a target field (or fields) with as little reference to additional meta-data such as a complete model of the message format.

The standard is a declarative standard based on a UML model that defines the artifacts necessary to define a standardized conversion. These artifacts represent a two-stage process, as described below.

6.3.1 Stage 1

The first stage artifacts utilize a Message Syntax Model to create a syntax-neutral set of Semantic Element classes. Semantic Elements are the smallest semantic entities contained in a message format, for which further parsing would lose semantic meaning leaving only generic data-type values.

6.3.2 Stage 2

The second stage provides semantic mapping to a central Domain Dictionary. It does this by specifying To and From Conversion Rules for source Semantic Elements to target Semantic Elements in another message format to Business Elements in a central Domain Dictionary such as the ISO 20022 Repository.

In many cases, this mapping will amount to a simple isomorphic mapping; in other cases, simple transformations will be required, such as defining an arithmetic expression, doing a table lookup, or splitting or concatenating a string. Separate transforms are defined for the mapping 1) from a source Semantic Element to a Business Element and 2) from a Business Element to a Semantic Element.

For example:

- Mapping "Primary Client Identifier" element in the source message to the two elements, "Primary Client Name" and "Primary Client BIC" in the dictionary.

- Mapping “Primary Account Beginning Balance” and “Primary Account Ending Balance” in the dictionary to “Primary Account Beginning Balance” and “Primary Account Debited Amount” in the target.

Note: There may be no simple or reasonable Conversion Rule between a source Semantic Element and a Business Element in an industry Domain Dictionary, such as the ISO 20022 repository. This indicates that the Semantic Element represents a concept not yet included in the industry Domain Dictionary. In this case, a submission should be made to the governing body to enhance the industry Domain Dictionary, rather than include a complex or convoluted mapping.

6.4 Future Benefits of the Specification

There are a number of extensions to the MDMI specification that should enhance its value.

6.4.1 Dealing With (Near) Synonyms

A key feature of the MDMI specification is the semantic mapping that is carried out between the Semantic Elements and Business Elements contained in an industry Domain Dictionary, such as the ISO 20022 Domain Dictionary. These conversions should be restrictive to a small set of direct conversion rules, e.g., only allowing arithmetic and logical expressions and limiting external functions to table lookups.

In effect, establishing such a set of rules can be used to define the semantic proximity between the Business Elements (or in the case of ISO 20022 the Message Elements) in a Domain Dictionary. This semantic proximity can be characterized as defining synonyms and “near synonyms.” This is accomplished because terms that can be mapped to the dictionary using the conversion rules must be synonyms or “near synonyms.” Only terms that are not synonyms or near synonyms of other Business Elements would be allowed in the basic dictionary itself. The synonyms and near synonyms with their mappings could be kept in an auxiliary catalog. The allowable rules established for the Conversion Rules in effect define the minimum semantic distance that is allowed for dictionary entries, resulting in a “measurable” well-structured dictionary. The future work would involve defining appropriate sets of conversion rules and understanding their implication on the dictionary structure.

6.4.2 Mapping between Data Dictionaries

The current specification is focused on supporting Semantic Element conversions among one or more message standards within Financial Services by mapping Semantic Elements to one large, central Domain Dictionary. However, the MDMI Semantic mappings could be applied to create maps between data dictionaries. Thus, the MDMI specification could be used to effectively support federated dictionaries. This, in turn, will allow content aware standards groups to manage dictionaries for specific subsections of the financial services industry, as opposed to one group being responsible for a large Domain Dictionary. A federated set of dictionaries might be more effective to maintain.

6.4.3 Handling Lossless Conversion

An important need in messaging is dealing with the loss of information when performing Semantic Element conversions. While this problem can never be completely solved improvements in lossless conversions will be a great benefit. The artifacts for the MDMI specification can provide a strong basic framework for creating lossless conversions, e.g., syntax incompatibilities can be traced and accommodated; auxiliary storage for lost information can be created with additional Semantic Elements, etc.

7 Use of MDMI Artifacts Overview

The focus of the MDMI specification is to create a template for machine-readable maps that standardize the conversion of data from a source message instance based on one message format to data in a target message instance based on another message format. This may involve the movement of as little as one data element or it may involve the conversion of a complete message. The specification can be used to map data for message formats within a Message Group or across Message Groups.

7.1 Informal Overview of Artifacts

Before presenting the artifacts in the MDMI specification, an overview and example of the use of the key artifacts in performing a conversion may be helpful.

Figure 7.1 and Figure 7.2 present an implementation of a conversion utilizing the key artifacts in the MDMI specification. The rectangles in the diagram represent these artifacts. In addition, it should be understood that the Business Elements in Figure 7.1 are the same Business Elements as in Figure 7.2 and that these Business Elements are defined in a central dictionary.

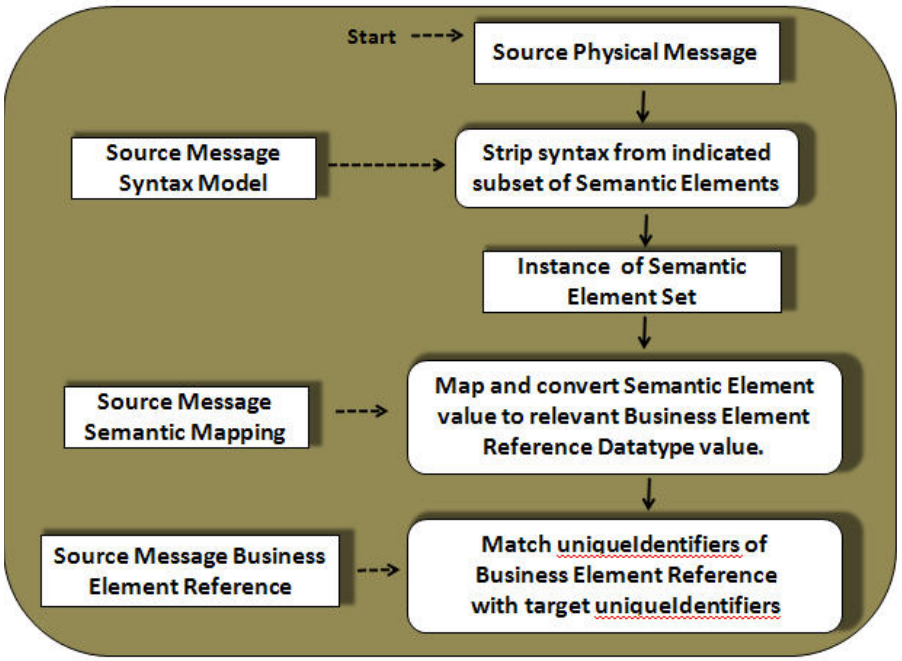

Figure 7.1 - Overview of run-time conversion methodology from Source

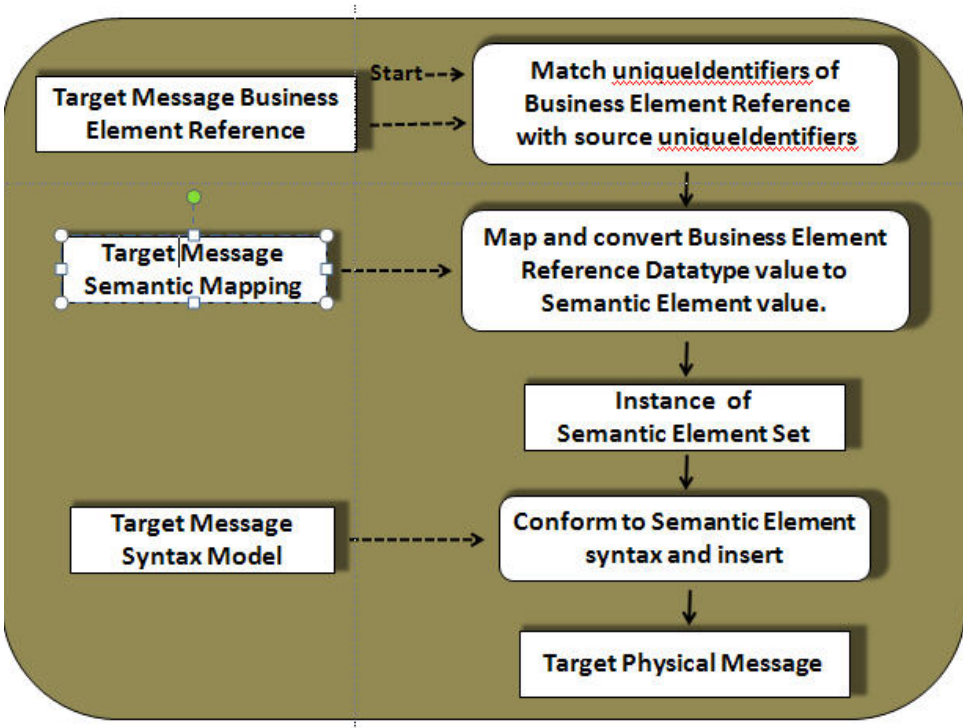

Figure 7.2 - Overview of run-time conversion to Target

The following step descriptions annotate this conversion example.

7.1.1 Step 1 - Remove the Syntax

The first step of a conversion is to convert the targeted data in a physical message instance (e.g., a SWIFT MT103, a Visa TC05, etc.) from its existing format to a syntax-neutral format. The conversion involves the extraction of data from the existing Message using a syntax translation process. This process utilizes the MDMI specification artifact, "Message Syntax Model." The Message Syntax Model provides a syntactic description that contains the necessary information to extract or insert any particular data item from/to a physical message instance.

A data item in a message is defined as the smallest semantic unit in a message for which further parsing would lose semantic meaning leaving only generic datatype values. For example, in a SWIFT MT102 there is a field representing a Settlement Date. If further parsing was done, the value left would simply be a date and indistinguishable, in a business semantic context, from any other date. Therefore, Settlement Date is a data item that is the smallest semantic unit. The data item "Settlement Date" has a datatype of date.

Normally the smallest semantic unit in a message is a field, but in many overloaded message formats a semantic unit can be a sub-field or a combination of fields. In existing message formats, many "fields" have been subdivided into numerous semantic units. For example, a field may contain a list of "Primary Account IDs" separated by commas. In that case, each "primary account ID" is a separate data item even though they appear in one field.

When the data is stripped of its specific message format syntax, its value will be represented by an instance of the artifact "Semantic Element." There will be a Semantic Element class defined for every semantic unit contained in a message's message format. All of these Semantic Element classes are contained in the "Semantic Element Set" by composition.

7.1.2 Step 2 - Mapping a Source Semantic Element to a Target Semantic Element through the use of a Unique Identifier acquired from a central dictionary

The second step for the conversion leverages a central dictionary to define the relationship between a Source Semantic Element and one or more Target Semantic Elements.

The Source and Target Semantic Elements are associated with a central dictionary Business Element through a Business Element Reference class. That association may be a simple isomorphic mapping or it may involve a more complex map utilizing various artifacts in the MDMI specification such as a computed Semantic Element or a Conversion rule. Each element in the central dictionary has to provide a unique identifier for its Business Elements. That unique identifier will be stored in the Business Element references that are associated with Semantic Elements. The appropriate Unique Identifiers will have been stored in the MDMI map for all Semantic Elements in both the Source and Target message formats.

An MDMI runtime application can locate a complete definition of a transformation by lining up the Source and Target maps for the Business Element References that have matching Unique Identifiers. However knowing the direct mapping instructions is often not enough information to insert a value into a Target message, as the validity of that insertion often depends on other Semantic Elements in a message. For example, it may be invalid to store a “Primary Account Balance amount” if there is no value for a “Primary Account ID.” Therefore, the maps for each Semantic Element include a set of Semantic Element Relationships that describe the relationship of a particular Semantic Element with all other Semantic Elements in the message. A runtime application uses the Semantic Element Relationships in its target mapping to make sure that no constraints are violated and that the inserted value is valid in relationship to other elements in the Message.

8 UML Semantics - Normative Definition

The following is the formal Meta-Object Facility (MOF) model of the Conversion Models for Payment Messages Standards. It is first presented as a set of annotated views followed by the presentation of all the “elements” brought together in a single view.

8.1 MessageModels, MessageGroup, MDMIDictionaryReference

8.1.1 Overview

This view presents the MessageModel, the MessageGroup, and the MDMIDictionaryReference. A MessageModel is a formal representation of a message format. A MessageGroup is composed of a set of Message Models that are usually grouped together because they focus on a particular messaging domain. For example, the set of SWIFT MTxx payment messages, the set of SWIFT MXxx fund messages, the set of Visa TCxx retail payment messages. An MDMIDictionaryReference provides a reference to the central dictionary to which the Semantic Elements for all MessageModels in the MessageGroup will be mapped.

8.1.2 Abstract Syntax

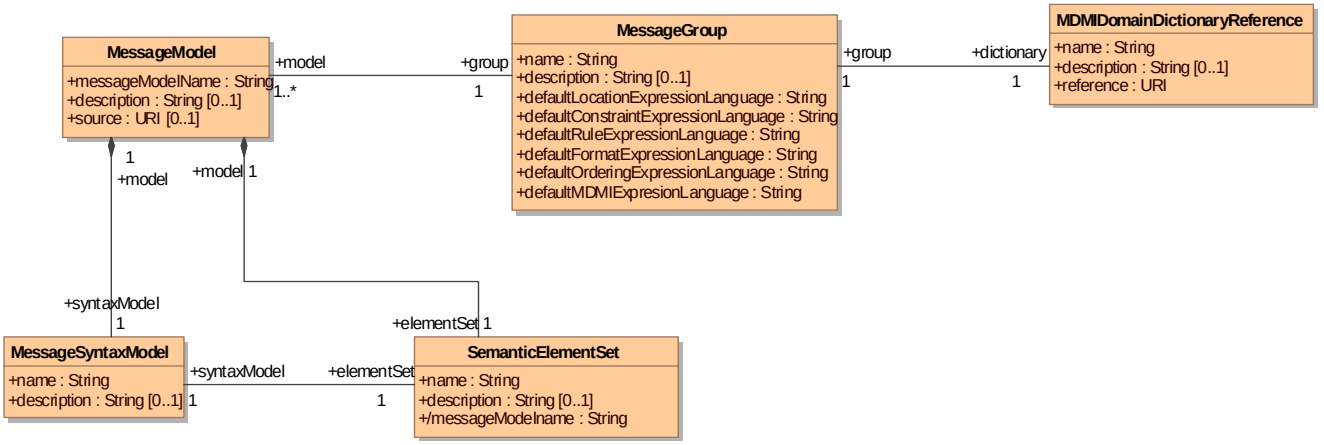

Figure 8.1 - Message Model, MessageGroup, MDMIDictionaryReference

8.1.3 MessageModel - Detailed Semantics

Description

The MessageModel is the parent class that contains the MDMI model of a message format. The database schema of a record in a table can also be considered a message format as well as most XML documents.

Properties

1. A “messageModelName” property, of type String, names the model of the message format being modeled. For example, the value of a messageModelName for an MT103 MessageModel could undoubtedly be “MT103.”

2. An optional “description” property, of type String, contains a description of the message model.

3. A “source” is a property, of type URI, which contains a reference to the definition of the message format whose elements are being mapped. This reference can take many forms; for example, the reference might be to a machine-readable definition, such as the location of the message definition in the ISO 20022 repository, or it might reference a paper document.

Associations

1. A MessageModel has a MessageSyntaxModel by composition.

2. A MessageModel has a SemanticElementSet by composition.

3. A MessageModel is associated with a MessageGroup.

8.1.4 MessageGroup - Detailed Semantics

Description

The MessageGroup class contains a set of message models that are considered in the same grouping (e.g., SWIFT MX messages, SWIFT 15022 messages, FIX security messages, etc.). The MessageGroup class is useful for setting various defaults for closely related message formats.

Properties

1. The property “name” of type String, names the MessageGroup.

2. The optional property “description,” of type String, provides a description of MessageGroup.

3. The property “defaultLocationExpressionLanguage” of type String identifies the location language to be used as a default for specifying location for all the messages in the MessageGroup. The value must be recognized by a runtime transformation application. The location of any field or sub-field in a message must be expressible in the chosen locationExpressionLanguage. For example, a location language for an XML message format would be “XPath 2.0.”

4. The property “defaultConstraintExpressionLanguage” of type String identifies the constraint language to be used as a default for specifying the constraints in the Choice class for all the messages in the MessageGroup. The constraintExpressionLanguage must be able to reference nodes. An appropriate language, which has been used in an example implementation, is NRL 1.0.

5. The property “defaultRuleExpressionLanguage” of type String identifies the rule language to be used as a default for specifying rules in all classes with the property “rule” for all the messages in the MessageGroup. This rule language must be able to access the values of any SemanticElement and thus it must be able to access the fields in complex datatypes. An appropriate language, which has been used in an example implementation, is NRL 1.0.

6. The property “defaultFormatExpressionLanguage,” of type String, identifies the format language to be used as a default for specifying formats in the LeafSyntaxTranslator class for all the messages in the MessageGroup. The formatExpressionLanguage must be able to describe fields as well as sub-fields, in particular the proper termination character for a field or sub-field. Appropriate languages, which have been used in an example implementation, are the SWIFT 150022 regular expression format language and XSD format attributes.

7. The property “defaultOrderingExpressionLanguage,” of type String, identifies the ordering language to be used as a default for specifying the ordering of multiple instances of Semantic Elements in which the Boolean property “mutipleInstances” is “True.” The ordering language should provide expressions that evaluate to both cardinal and ordinal positioning. For example, NRL is a language that can be used to specify ordering.
8. The property “defaultMDMIExpressionLanguage,” of type String, identifies the computational language to be used as a default for specifying the computational expression in computed Semantic Elements that are of type MDMIExpression. For example, NRL, with its declarative and action language, can be used as an MDMI Expression Language.

Associations

1. An association with one or more MessageModels, which comprise the MessageGroup.
2. An association with zero or more DataRules that are utilized by the Message models within the group.
3. An association with the MDMIDictionaryReference that identifies the central dictionary utilized by the group.

8.1.5 MDMIDomainDictionaryReference

Description

The MDMIDomainDictionaryReference class provides a reference to the central dictionary that contains the Business Elements to which the Semantic Elements in the MessageModels in the MessageGroup are mapped. This class is purely informational as the URI reference to the dictionary does not have to be machine-readable. The dictionary could reside on paper, for example. However, there must be a function or method associated with the dictionary that will provide: 1) a uniqueIdentifier for all Business Elements, and 2) a reference to a datatype that is compatible with the set of MDMIDatatype.

Properties

1. A “name” property, of type string, that provides a name for the referenced central dictionary.
2. An optional “description” property, of type String, that provides a description of the referenced central dictionary.
3. A “reference” property, of the type URI, that provides a reference to the central dictionary, such as a URL.

Associations

1. MDMIDomainDictionaryReference has a one-to-one association with MesssageGroup to indicate the central dictionary that will be used for the maps in MessageModels in the MessageGroup.
2. MDMIDomainDictionaryReference has a one-to-many relationship to the MDMIBusinessElementReference class so that a reference to the parent dictionary, to which a Business Element belongs, is easily found.

8.2 MessageSyntaxModel, Node, Bag, Choice, LeafSyntaxTranslator

8.2.1 Overview

The MessageSyntaxModel and related classes provide syntax information that will enable a process to either extract or insert a data value into or from an instance of a message. It does this by providing a description of the location and format of every Semantic Element in the message format.

The MessageSyntaxModel class is the root of the syntax tree. The syntax tree provides a map for navigating a message format. The leafs of the tree are LeafSyntaxTranslator nodes. The LeafSyntaxTranslator has location and format properties, which contain information that defines how to move a data item from/to an instance of a message and associate the data item with a Semantic Element. The MDMI specification does not require a specific language to describe a location or a format for the properties in the LeafSyntaxTranslator. Instead, language properties are included that provide a reference to the expression language that will be used to describe location and format. This flexibility was chosen given the variety of different types of message formats (for example: XML, EDIFACT, Object models, etc., and the legacy languages already out there to express location and format).

The other classes associated with the MessageSyntaxModel are used to construct the branches of the syntax tree. They are:

- Node - an abstract class that represents the branches and leaf nodes of the syntax tree.
- Bag - a branch Node that identifies a set of Nodes that are aggregated in a message format.
- Choice - a branch Node that defines rules to identify the conditions for which values in its children nodes should appear in a physical message instance.

8.2.2 Abstract Syntax

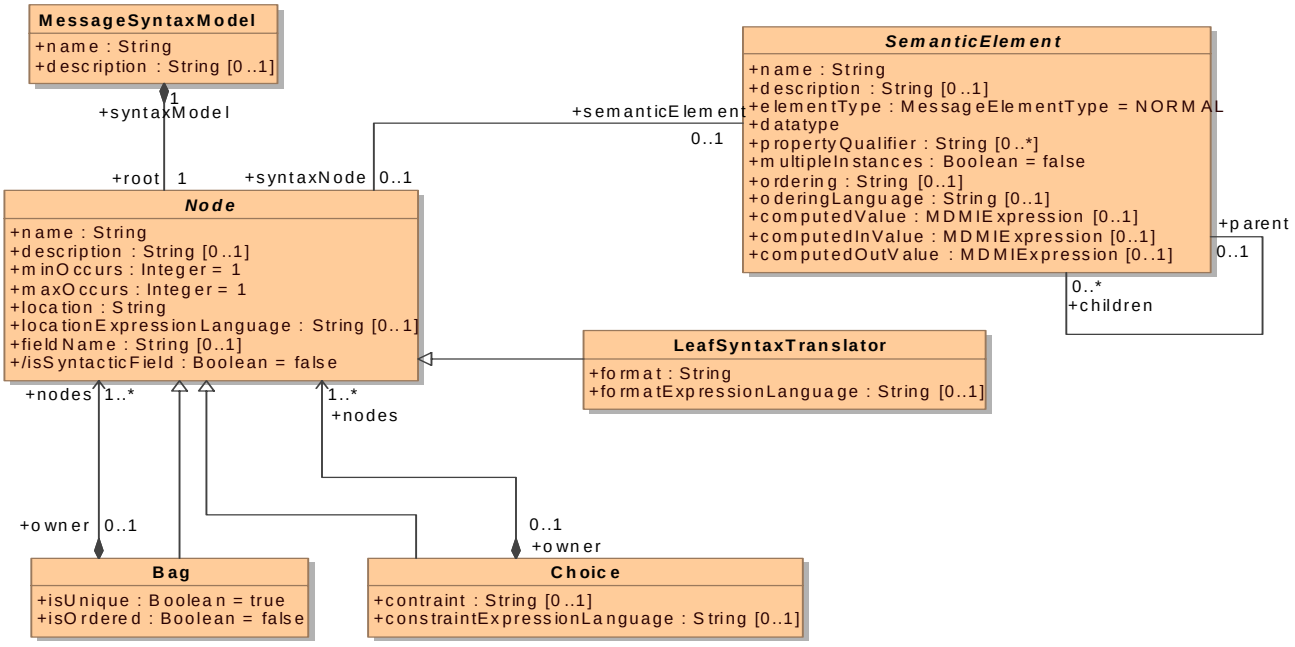

Figure 8.1 - Message Syntax Model

8.2.3 MessageSyntaxModel - Detailed Semantics

Description

The MessageSyntaxModel contains a syntax tree that describes how each Semantic Element can be either inserted into or extracted from a message based on that message's message format.

Properties

1. A “name” property, of type String, is the name of the MessageSyntaxModel. This name will often be similar to the MessageModel name, e.g., “MT103 Syntax Tree.”

2. The optional property “description” of type String provides a description of MessageGroup.

Associations

1. An associations with one-to-many Nodes as it is the parent class of the syntax tree.

2. An association with its parent MesssageModel.

3. An association with its sibling SemanticElement Set.

8.2.4 Node - Detailed Semantics

Description

The Node class is an abstract class that is inherited by all nodes in the syntax tree. It primarily contains location information so that any field or data item in a message can be located.

Properties

1. The “name” property, of type String, provides a name for the Node. This name can be useful to label a section or element in a message format. The name property is important because it should provide an addressable reference to the node, which can be used in an expression.

2. The optional “description” property, of type String, describes the Node’s purpose.

3. The “minOccurs” property, of type Integer, has a value of 0..1. The value of “0” indicates that the Node is optional whereas the value “1” indicates that the Node is required.

4. An optional “maxOccurs” property, of type Integer, puts an upper limit on the number of instances allowed for the node.

5. A “location” property, of type String, describes the location of the Node in the physical message. The location is often in reference to, or anchored by, the URI that defines the location of the physical message instance.

6. A “locationExpressionLanguage” property, of type String, defines a reference to the expression language used in the location property. The locationExpressionLanguage must satisfy the same constraints described for the “defaultLocationExpressionLanguage” in section 8.1.5.

7. An optional “fieldname” property, of type String, provides the field name of a simple datatype that is part of a complex MDMIDatatype. The data item, whose location is indicated by the Node, has the datatype associated with the “fieldname.”

8. A derived property “isSyntaticField,” of type Boolean, indicates, if the property’s value is “True,” that this node corresponds to a data item that is part of an MDMIComplexDatatype. “isSyntaxField” will be “True” if the optional “fieldname” is present.

Node Class Generalizations

Three classes inherit from the Node abstract class: Bag, Choice, and LeafSyntaxTranslator.

Node Class Associations

1. Node has a many-to-one association with the Bag class as a Bag can have Node children.
2. Node has a many-to-one association with the Choice class as a Choice can have Node children.
3. Node has a one-to-one relationship with a SemanticElement. This is the key association that links a SemanticElement to its syntax.

8.2.5 Bag - Detailed Semantics

Description

The Bag class represents a set of syntax nodes. The associated nodes of a Bag can be a unique set or a bag, and these nodes can be ordered or unordered.

Properties

1. The “isUnique” property, of type Boolean, indicates, if its value is “True,” that the bag is a set composed of unique items. If its value is “False,” the bag of nodes can contain duplicates.
2. The “isOrdered” property, of type Boolean indicates, if its value is “True” that the nodes in the bag must be in an ordered sequence. If the value is “False,” the nodes in the bag can be unordered. This property is useful for parsing a message. The actual ordering of SemanticElements is handled 1) using the “location” property in the Node class, and 2) using the “ordering” property in the SemanticElement class.

Associations

1. The Bag class has a one-to-many association with some other classes that inherits from Node. Thus, it becomes a branch in the syntax tree. Since it must have at least one association with another class by composition, it cannot be a leaf of the syntax tree.

8.2.6 Choice - Detailed Semantics

Description

The Choice class contains the conditions that can identify the subset of its children nodes that will be present in a particular message instance. The subset is determined by a constraint expression.

Properties

1. A “constraint” property whose value is an expression that can be used to determine which of the set of nodes should be in a physical message instance.
2. An optional “constraintExpressionLanguage,” of type String that is a reference to the language used in the “constraint” property. The constraintExpressionLanguage must be able to reference any node in the syntax tree.

Associations

1. The Choice class has a one-to-many association with some other class that inherits from Node. Thus, it becomes a branch in the syntax tree. Since it must have at least one association with another class by composition, it cannot be a leaf of the syntax tree.

8.2.7 LeafSyntaxTranslator

Description

The LeafSyntaxTranslator class represents a leaf of the syntax tree. There is a LeafSyntaxTranslator corresponding to every field, sub-field, or data item in the message format. The LeafSyntaxTranslator inherits location information from the Node and has additional properties that describe the format of the data item with which it is associated.

Properties

1. The “format” property, of type String, provides the specific format of a field or subfield in the message format.
2. The “formatExpressionLanguage” property, of type String, is a reference to the expression language used in the format property. For example, SWIFT has a defined regular expression language for the format of fields in MT messages. The formatExpressionLanguage must be able to reference and fully describe the format of data item. An example would be being able to specify the proper termination character for list of fields that occur within a string. While the MDMI specification does not require a specific formatExpressionLanguage, if no formatExpressionLanguage exists for a particular message format, the MDMI specification is recommending the use of a subset of DFDL as a general solution.

8.3 SemanticElementSet, SemanticElement SimpleMessageComposite, MessageComposite, Keyword

8.3.1 Overview

The SemanticElementSet contains a set of Semantic Element classes. Each SemanticElement represents a smallest semantic unit in a message format. The SemanticElementSet and the MessageSyntaxModel, which are the two entities that comprise a Message model, can provide a complete specification of a message format. If all the Semantic Elements in a message are stored in the SemanticElementSet and instructions on how to insert or extract each of those elements are contained in the MessageSyntaxModel, then a complete model of a message format will be created. However, one of the advantages of MDMI is subsets of a message format can also be mapped. For example, given a specification such as RosettaNet and a goal of executing a payment, only the payment data-items that are to be moved into a SWIFT payment message need to be mapped.

The SemanticElementSet represents the “flattening” or the “linearization” of a message format. This flattening is important, since a primary goal of MDMI is to expedite the insertion or extraction of as little as one semantic unit of a message. For processing efficiency, it is very important that the information needed to convert one item from/to a message does not require complete information about the structure of the entire message format.

The primary constituents of the SemanticElementSet are Semantic Elements. A couple of additional classes are provided primarily for the ease of the designer, but they do not play a major role in the conversion process. These are SimpleMessageComposites and MessageComposites. These classes are conveniences for bundling SemanticElements in the design process.

A SimpleMessageComposite is an “aggregation” that only contains SemanticElements. It is important, as this first level of aggregation is a very common design mechanism.

A MessageComposite is an aggregation that contains SemanticElements, SimpleMessageComposites, and MessageComposites. It is possible therefore to create exceedingly complicated MessageComposite structures. However, these structuring mechanisms should be used with considerable caution. Such complicated structures are far away from the desired linearization or flattening of semantic units, which is a core design principle of the MDMI specification.

An important property of SemanticElements merits further discussion. This is the property “multipleInstances.” MultipleInstances indicates that instances of a particular SemanticElement can appear multiple times in a physical message instance, usually in the form of repeating fields or a list. In effect, the SemanticElement is a vector and not a singular value. As expected, the fact that SemanticElements can be an array of values increases the complexity of the model.

8.3.2 Abstract Syntax

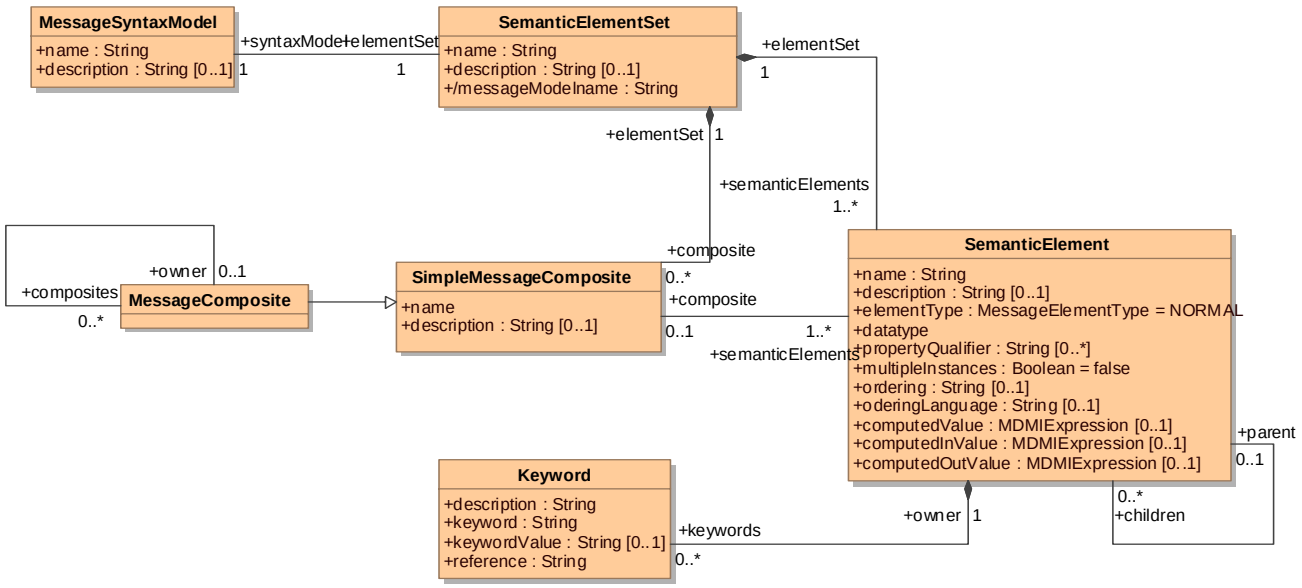

Figure 8.2 - SemanticElementSet and associated classes

8.3.3 SemanticElementSet - Detailed Semantics

Description

The SemanticElement Set contains the smallest Semantic Elements contained in a message format. The set only holds Semantic Elements. All of the message-specific syntax of selected elements from a particular message format has been removed.

Properties

1. A “name” property, of type String, contains the name of the SemanticElementSet.
2. The optional “description” property, of type String, provides a description of the SemanticElement Set.
3. The derived “MessageModelName” property, of type string, contains the name of the MessageModel to which the SemanticElementSet belongs. This derived property is included for implementation convenience.

Associations

1. The SemanticElementSet has a one-to-many association by composition to SemanticElements.

2. The SemanticElementSet has a zero-to-many association with SimpleMessageComposites. A SimpleMessageComposite is a convenient mechanism for grouping SemanticElements.

3. The SemanticElementSet has a one-to-one relationship to its parent MessageModel.

4. The SemanticElementSet has a one-to-one relationship to its sibling, the MessageSyntaxModel.

8.3.4 SemanticElement - Detailed Semantics

Description

The SemanticElement class is the core of the MDMI message map. SemanticElements represent the smallest semantic units in a message format, stripped of any complicating syntax considerations. Each SemanticElement is unique in the context of its message format, i.e., it must have an individual semantic meaning. As example, “address” cannot be a SemanticElement; “address” is a datatype that can be repeated in many message fields. “Primary Debtor Address” is a SemanticElement as it refers to a particular unique address in a message format.

Properties

1. A “name” property, of type String, contains the name of the SemanticElement.

2. The optional “description” property, of type String, contains a description of the SemanticElement.

3. An “elementType” property, of the enumerated type MessageElementType, can have three values each of which defines the type of Semantic Element.

    - NORMAL - a “NORMAL” Semantic Element is a smallest unique semantic element that appears in a message format, which is to be mapped to a central Domain Dictionary.

    - LOCAL - a “LOCAL” Semantic Element contains some technical information that is needed to correctly map NORMAL Semantic Elements, e.g., it may contain an index that is used to provide the ordering for a child Semantic Element that has multiple instances.

    - COMPUTED - a “COMPUTED” Semantic Element is to be mapped to the central dictionary but contains a value that is not directly contained in a message. Instead, a “COMPUTED” Semantic Element’s value is computed using.

4. A “datatype” property, of type MDMIDatatype, defines the simple or complex datatype of the Semantic Element.

5. A zero-to-many “propertyQualifier” property, of type String, is a list of keywords that contains reference keywords of interest that are associated with the message format, such as a “tag” associated with a SemanticElement.

6. A “multipleInstances” property, of type Boolean, which if true indicates that instances of this SemanticElement can be repeated in a physical message as a list or array.

7. An “ordering” property, of type String, contains an expression that describes how the Semantic Element instances are ordered, if the SemanticElement's multipleInstances property is “True.”

8. An optional “orderingExpressionLanguage” property, of type String, that is a reference to the expression language used for the value of the “ordering” property. The ordering language must be able to describe ordinal and cardinal positioning as well as expressions that when evaluated will provide an index. As an example, a language that can be used is NRL 1.0.

9. A “computedValue” property, of type MDMIexpression, contains an expression that computes the value for the SemanticElement. The expression can refer to the value of other SemanticElements. This property is most often used for SemanticElements of the type LOCAL.

10. A “computedInValue” property, of type MDMIexpression, contains an expression to compute a value for the SemanticElement when it is a target, based on the values of one or more BusinessElements and SemanticElements. The value when it is a source is mapped directly.

11. A “computedOutValue” property, of MDMIexpression, contains an expression to computes value for a SemanticElement, when it is a source, based on the values of one or more SemanticElements. The value when it is a target is mapped directly.

Associations

1. A one-to-many association with any children through a parent association. This allows the SemanticElementSet to include container Semantic Elements, which are identified by “parent.” Explicit container Semantic Elements allow the hierarchical structure of a message format to be maintained in the SemanticElementSet. In the case where a container SemanticElement has no message-based properties itself, that container should be of type Computed with a simple index as the computed value.

2. A zero-to-many association to the SemanticElementRelationship class. The SemanticElementRelationship provides the valid context for each SemanticElement.

3. A one-to-one relationship to a syntax Node. The Node provides the syntax information associated with the SemanticElement.

4. A many-to-(one or zero) association with a SimpleMessageComposite. SimpleMessageComposites provide a convenient mechanism for grouping SemanticElements.

5. A many-to-one association with its parent SemanticElementSet.

6. A zero-to-many association with the DataRule class, which specifies a set of rules that apply to the datatype of the SemanticElement.

7. A zero-to-many association with a keyword list, which can be used to identify the SemanticElement for searches and which can be associated with a formal ontology.

8. A zero-to-many association with a SemanticElementBusinessRule, which provides for a specific set of rules that should apply to the value of the SemanticElement.

9. A one-to-many association with the ToBusinessElement class that describes the conversion of the value of the SemanticElement to conform to the reference value of the business element referenced by the MDMIBusinessElementReference class.

10. A one-to-many association with the ToSemanticElement Semantic class that describes the conversion of the reference value of the business element referenced by the MDMIBusinessElementReference class to the value of the SemanticElement.

8.3.5 Keyword - Detailed Semantics

Description

The Keyword class contains either a keyword or a keyword/value pair. The set of Keywords can be used to profile a SemanticElement, to provide a mechanism to search for a SemanticElement, and to associate a SemanticElement with an external ontology or taxonomy.

Properties

1. The optional “description” property, of type string, describes the Keyword and/or the set of Keyword associated with a SemanticElement.
2. A “keyword” property, of type String, used to describe or profile a SemanticElement.
3. An optional “keywordValue,” of type string, that is associated with the keyword creating a keyword/value pair.
4. An optional reference, of type String, identifies the origin set for the keywords, for example, a formal ontology.

Associations

1. An optional many-to-one association with the SemanticElement it is describing.

8.3.6 SimpleMessageComposite - Detailed Semantics

Description

SimpleMessageComposite represent aggregations of SemanticElements. SimpleMessageComposite is an informative artifact that can be useful when a group of SemanticElements are associated with a class in an object model. Usually the attributes of an object will be equivalent to a SemanticElement and the object itself equivalent to a SimpleMessageComposite.

Properties

1. A “name” property, of type String, names the SimpleMessageComposite.
2. An optional “description” property, of type String, describes SimpleMessageComposite.

Generalization

MessageComposite inherits from SimpleMessageComposite.

Associations

1. A zero-to-many association with a SemanticElementSet by composition.
2. A (zero or one)-to-many association with SemanticElements.

8.3.7 MessageComposite - Detailed Semantics

Description

The MessageComposite class inherits from the SimpleMessageComposite class, allowing the construction of a complex object tree. MessageComposite are an informative artifact that can be useful when there is a desire to associate SemanticElements with a complex object model.

Associations

1. A zero to many association with other MessageComposites that are the children of the MessageComposite, thus providing a mechanism to specify a tree of MessageComposites.

8.4 MDMIDatatype, DataRules

8.4.1 Overview

The MDMIDatatype references a datatype used in the model. These MDMIDatatypes are not considered part of the MDMI specification. While the specification does not deal with datatypes directly, some restrictions on MDMIDatatype definitions are necessary for syntactic modeling and to ensure that a runtime engine will do proper transformations. These restrictions include:

- that the simple datatypes be from a known standard, such as the XML simple datatypes.
- that complex datatypes are ultimately composed of simple datatypes and that every simple datatypes has an identified “fieldname.”

Associated with any value can be DataRules that describe constraints for that datatype, e.g., a zip code value must be in a table of legal zip codes. DataRules must be written in an appropriate Rule Expression Language that can access the components of a complex MDMIDatatype using “fieldnames.”

8.4.2 An Example of Complex Datatype

A Semantic Element can be composed of complex datatypes that actually span a number of fields (or sub-fields) in a message format. Each such field, by itself, does not have a specific semantic meaning in the message but is rather a syntactic artifact that when combined with other fields represent a complete datatype. For example, an address can be composed of many fields and is a complex datatype. The Syntax Model must be able to associate each component of a complex datatype with a field in the message.

An example of a modeled MDMI complex datatype is shown in Figure 8.4.

This complex datatype model is composed of classes, where the classes themselves can be complex datatypes or a class with a single valued simple datatype. Ultimately, all complex datatypes resolve to a set of simple datatypes, which correspond to fields (or subfields) in a message format. Therefore, to accommodate Semantic Elements that are complex datatypes, a “fieldname” attribute is a property of the Node abstract class, which holds the name of the simple datatype class. For computational efficiency, a derived attribute is also added that says this node instance contains a syntactic element that is part of a complex datatype.

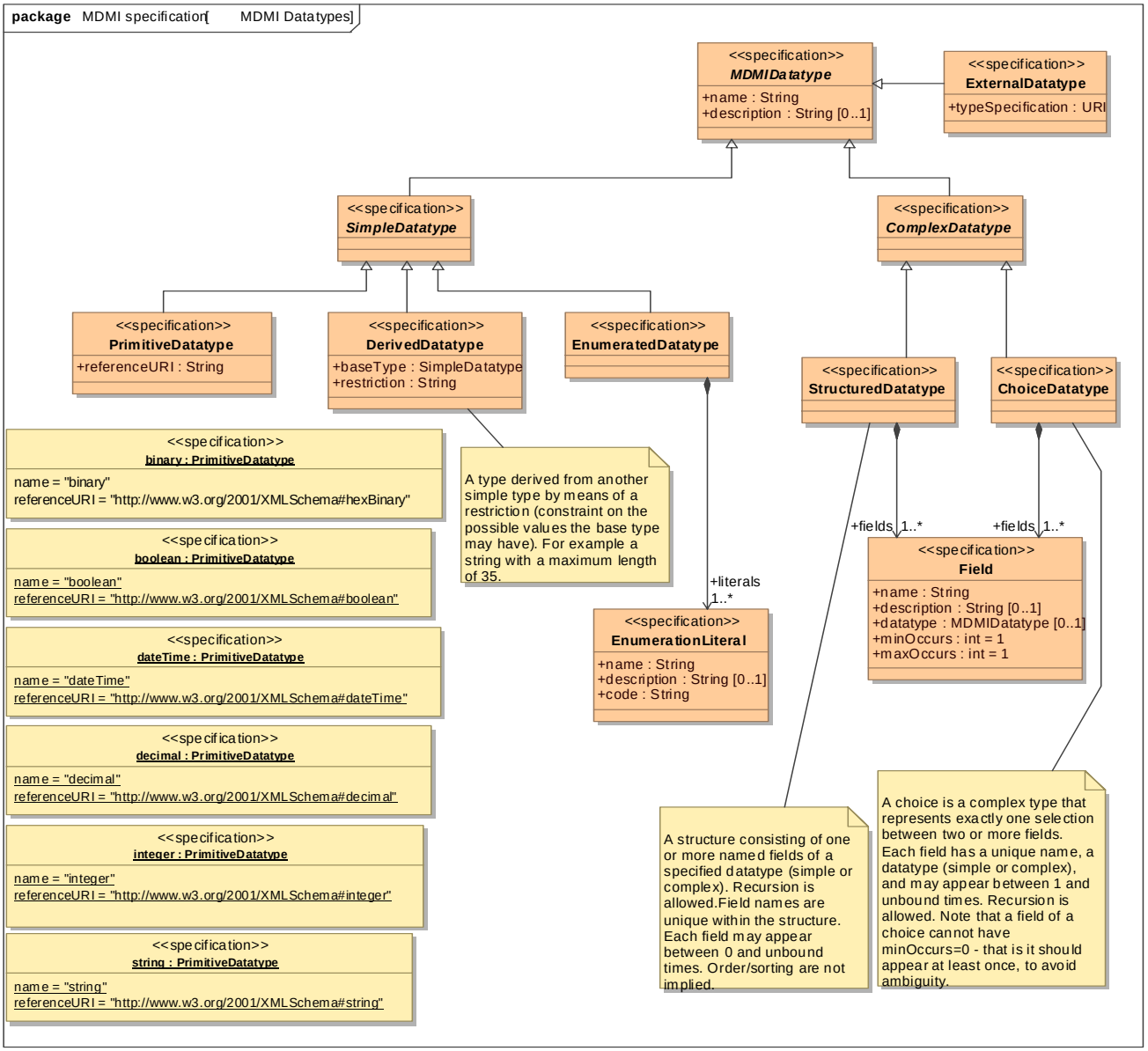

Figure 8.3 - Complex Datatype

8.4.3 MDMIDatatype, DataRules - Abstract Syntax

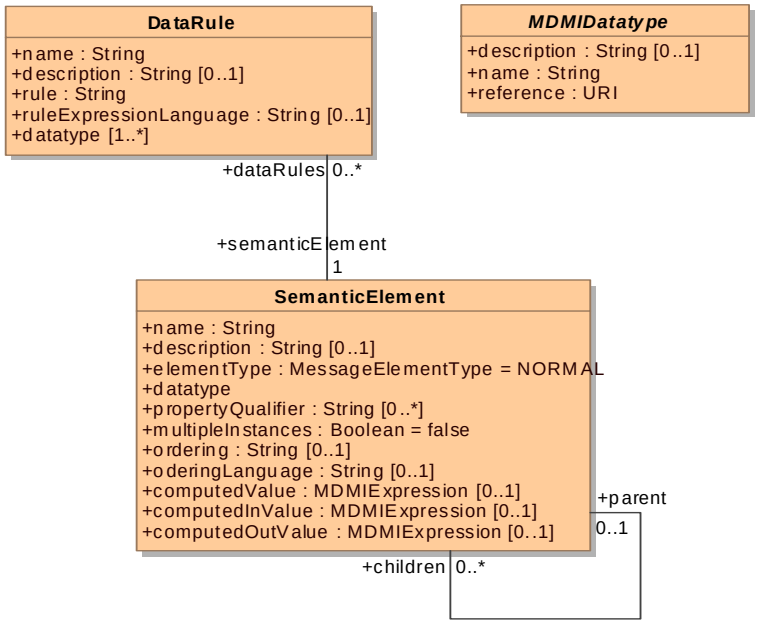

Figure 8.4 - MDMIDatatypes, DataRule

8.4.4 MDMIDatatype - Detailed Semantics

Description

The MDMIDatatype class contains a reference to a conformant datatype, i.e., one that can be processed by the DataRule language. This class is used as a property type.

Properties

1. A “name” property, of type string, names of the MDMIDatatype.
2. An optional “description” property, of type string, describes the MDMIDatatype.
3. A “reference” property, of type URI, contains a reference to the MDMIDatatype definition.

8.4.5 DataRules - Detailed Semantics

Description

The DataRule class contains a rule that is a constraint on the MDMIDatatype that are used in the MessageGroup, to ensure that values extracted or inserted are valid.

Properties

1. A “name” property of type String whose value is the name of the DataRule.
2. An optional “description” property, of type String, contains a description of the DataRule.

3. A “rule” property, of type String, contains an expression for a rule or constraint associated with an MDMIDatatype either for the entire MessageGroup or for the particular use of an MDMIDatatype in a SemanticElement class.

4. A “ruleExpressionLanguage,” of type String, references the language in which the “rule” property is expressed. The standard does not require any particular rule language, but the language has to allow access to fields represented by simple datatype classes within a complex datatype.

5. A “datatype” property, of type MDMIDatatype and multiplicity of one-to-many, explicitly identifies the MDMIDatatypes that are referenced in a DataRule’s “rule.” The “datatype” references the complete structure of an MDMIDatatype, so that its structure and simple datatype fields are known. The “datatype” property is used to assist in the parsing and runtime processing of complex data.

Associations

1. Zero-to-many DataRules can be associated with a MessageGroup.

2. Zero-to-many DataRules can be associated with a SemanticElement class.

8.5 MDMIBusinessElementReference, Conversion Rule, ToSemanticElement, To BusinessElement, MDMIBusinessElementRule

8.5.1 Overview

The classes in this view describe the mapping between a SemanticElement and an MDMIBusinessElementReference. An MDMIBusinessElementReference class references a Business Element in a dictionary. No assumption is made about the format of the business element in the central dictionary. Because the format of the dictionary is not known and can even be a reference to documentation, an MDMIBusinessElementRules class is included in the specification so that rules and constraints concerning the business element can be specified.

Given the BusinessElementReference, a conversion between it and a SemanticElement can be made. This conversion may not be symmetric so a mapping must be defined for each direction - SemanticElement to MDMIBusinessElement and MDMIBusinessElement to SemanticElement. (Mappings for both directions must be defined, one way mappings are not allowed in the specification.) These mappings are specified in a ToSemanticElement class and a ToBusinessElement class. Both of these classes inherit from a ConversionRule abstract class that defines how conversion rules are to be specified.

A key feature of the conversion is the restrictions that are implied in the ConversionRules ruleExpressionLanguage. These restrictions define the allowed semantic distance for which mapping can be done. In effect, they define the domain of “near-synonyms” that are allowed in a mapping. For example, a set of allowed conversion rules may include, simple arithmetic expressions, aggregation of a set of elements, the removal or inclusion of qualifiers, etc. If a SemanticElement cannot be mapped, it implies that is not in the dictionary and should be added to the dictionary.

8.5.2 Abstract Syntax

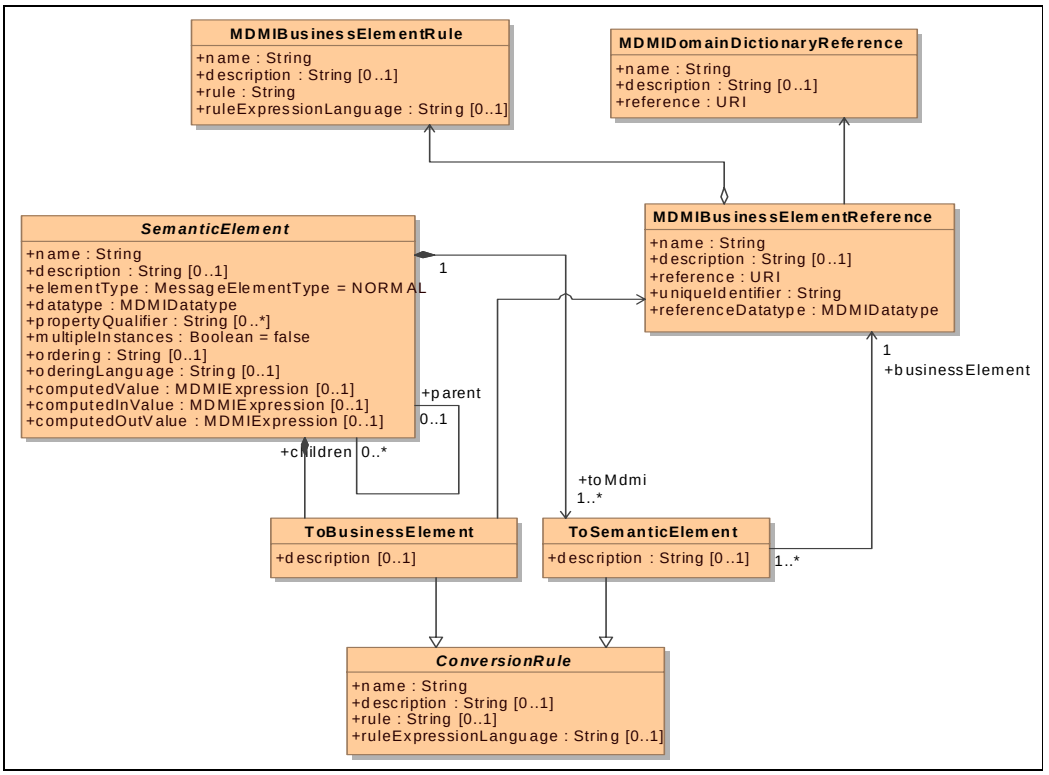

Figure 8.5 - MDMIBusinessReference and ConversionRule

8.5.3 MDMIBusinessElementReference - Detailed Semantics

Description

The MDMIBusinessElementReference is a class that references a business element in a dictionary. No assumption is made about the format of the business element in the central dictionary. Therefore, the reference can only be informational. However a function must be available that, given the reference, will return a persistent uniqueIdentifier and a reference MDMIDatatype.

Properties

1. The “name” property, of type String, names the MDMIBusinessElementReference.
2. The optional “description” property, of type String, describes the MDMIBusinessElementReference.
3. The “reference” property, of type URI, identifies the location of the BusinessElement in a central dictionary. (URIs are very general addresses, i.e., the URI could even point to a line in a page in a document therefore the “reference” property is informational.)

4. The “uniqueIdentifier,” of type String, provides a unique identifier for all MDMIBusinessElementReference instances that reference the same business element in the central dictionary. There must be a function associated with the central dictionary that provides this identifier. Runtime transformation engines recognize the matching source and target mappings for a Semantic Element because they will each have the same “uniqueIdentifier.”
5. The “referenceDatatype” property, of type MDMIDatatype, provides a reference datatype for each business element in the central dictionary. There must be a function associated with the central dictionary that will deliver the “referenceDatatype.” Maps to/from this reference datatype to the “datatype” in the SemanticElement should be provided as a ConversionRule.

Associations

1. MDMIBusinessElementReference has a one-to-many association with the ToSemantic class.
2. MDMIBusinessElementReference has a one-to-many association with the ToBusinessElement class.
3. MDMIBusinessElementReference has a (zero or one)-to-many association with the MDMIBusinessElementRule class.
4. MDMIBusinessElementReference has a many-to-one relationship with the MDMIDomainDictionaryReference class.

8.5.4 ConversionRule - Detailed Semantics

Description

ConversionRule is an abstract class that defines a rule used to convert values.

Properties

1. A “name” property, of type String, names the ConversionRule.
2. An optional “description” property, of type String, describes the ConversionRule.
3. A “rule” property, of type String, holds an expression for converting one value to another.
4. A “ruleExpressionLanguage” property, of type String, is a reference to the expression language used to define the rule. The scope of the language allowed in conversions should be limited so that only very straightforward transformations are possible. This is because these ConversionRules can be used to define the semantic distance between business elements in a central dictionary by identifying “near synonyms.” It is important that the “near synonyms” do not turn out to be far synonyms.

Generalizations

The abstract ConversionRule class is inherited by two classes, the “ToBusinessElement” and the “ToSemanticElement.”

8.5.5 ToSemanticElement - Detailed Semantics

Description

The ToSemanticElement associates an MDMIBusinessElementReference to a SemanticElement, describing the directed conversion rule for converting the reference value of a Business Element to the value in a SemanticElement. MDMIBusinessElementReferences may be related to more than one SemanticElement but will have a separate ToSemanticElement class with individual rules for each relationship.

Properties

1. The optional “description” property, of type String, describes the ToSemanticElement.

Associations

1. A many-to-one association with an MDMIBusinessElementReference.
2. A many-to-one association with a SemanticElement.

8.5.6 ToBusinessElement

Description

The ToBusinessElement associates an MDMIBusinessElementReference with a SemanticElement, describing the directed conversion rule for converting the value of the SemanticElement to the reference value of the referenced business element. A SemanticElement may be related to more than one MDMIBusinessElementReference but will have a separate ToBusinessElement class with individual rules for each relationship.

Properties

1. The optional “description” property, of type String, describes the ToBusinessElement.

Associations

1. A many-to-one association with an MDMIBusinessElementReference.
2. A many-to-one association with a SemanticElement.

8.5.7 MDMIBusinessElementRule

Description

Given that the MDMI specification does not provide a specification for the hub dictionary and allows mapping to any appropriate dictionary, such as the ISO 20022 Domain Dictionary, then some business rules may have to be specified within a map to make sure that the mapping is correct.

Properties

1. A “name” property, of type String, contains a name of the rule.
2. An optional “description” property, of type String, provides a description of the rule.
3. A “rule” property, of type String, is an expression defining the rule that applies to an associated MDMIBusinessElementReference.
4. An optional “ruleExpressionLanguage,” of type String, provides a reference to the language used in the “rule” property. This language must be able to describe the context in which the rule applies. The language should be able to reference the value of any Semantic Element instance and it should allow external function calls. If this property is not specified, the default ruleExpressionLanguage will be used.

Associations

1. The MDMIBusinessElementRule has a many-to-one association with an MDMIBusinessElementReference.

8.6 SemanticElementRelationship

8.6.1 Overview

The SemanticElementRelationship classes define all the allowed contexts for SemanticElement in a message format. For example, a SemanticElement that is “ClientAccountBalance” may not be valid in a message instance unless there is also a value in the SemanticElement “ClientAccountID.” The SemanticElementRelationship class would define this relationship. On the other hand, “ClientAccountID” may exist without a value for “ClientAccountBalance,” in which case there will be no SemanticElementRelationship associating “ClientAccountID” with “ClientAccountBalance.”

8.6.2 Abstract Syntax

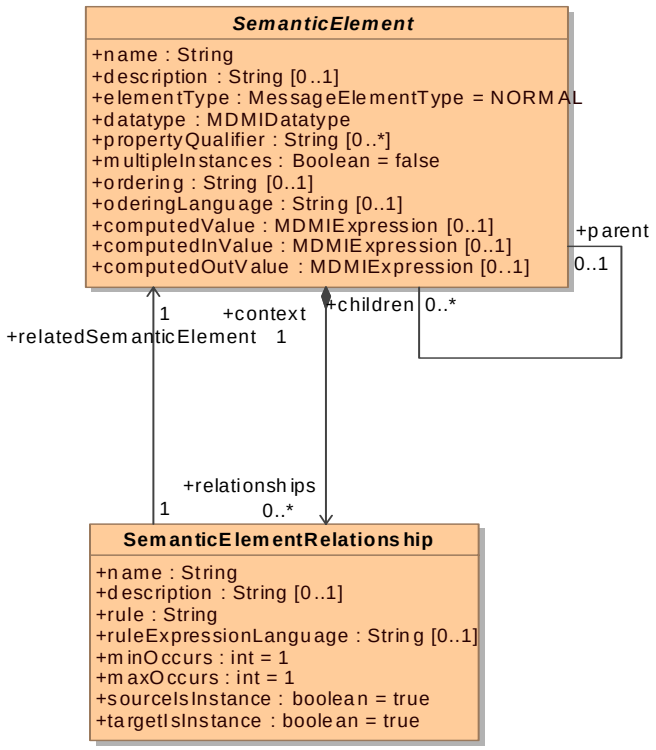

Figure 8.6 - SemanticElementRelationship

8.6.3 SemanticElementRelationship - Detailed Semantics

Description

The SemanticElementRelationship class is a key artifact in the MDMI specification. It provides all the context and dependency relationships for each SemanticElement. SemanticElementRelationship makes it possible to extract and insert SemanticElement values in a valid manner.

Properties

1. A “name” property, of type String, assigns a name to the rule.
2. An optional “description” property, of type String, provides a description of the rule.

3. A “rule” property, of type String, defines a relationship between a source SemanticElement and other SemanticElements in the SemanticElementSet.

4. A “ruleExpressionLanguage” property, of type String, that contains a reference to the expression language used in the “rule” property. This rule language must be able to access the values of any SemanticElement and to do that it must be able to access the fields in complex datatypes.

5. “minOccurs” property, of type integer, indicates how many instances of the target at a minimum must be involved in the relationship.

6. A “maxOccurs” property of type integer, which says how many instances, at most can be involved in the relationship.

7. A “sourceIsInstance” property of type Boolean. When the sourceIsInstance is true, the defined relationship is for each Instance of the source SemanticElement. The association with the “source” Semantic Element is labeled “relatedSemanticElement.” The relatedSemanticElement owns the relationship by composition. This source is the SemanticElement whose context is being modeled. When the sourceIsInstance is false, the defined relationship is for the source SemanticElement class as a whole.

8. A “targetIsInstance” property of type Boolean. When the targetIsInstance is true, the defined relationship is for each Instance of the target SemanticElement. The association with the set of one-to-many “targets” is labeled “context.” Thus, a SemanticElementRelationship describes a relationship between a source and the other SemanticElements, which are then targets. When the targetIsInstance is false, the defined relationship is for the SemanticElement class as a whole.

Associations

1. The SemanticElementRelationship has a (zero or many)-to-one association with its source SemanticElement.

2. The SemanticElementRelationship has one to-one association with a target SemanticElement.

8.7 SemanticElementBusinessRule

8.7.1 Overview

The SemanticElementBusinessRule class contains a rule that is to be applied to a specific SemanticElement in the context of the MessageModel that contains the SemanticElement.

8.7.2 Abstract Syntax

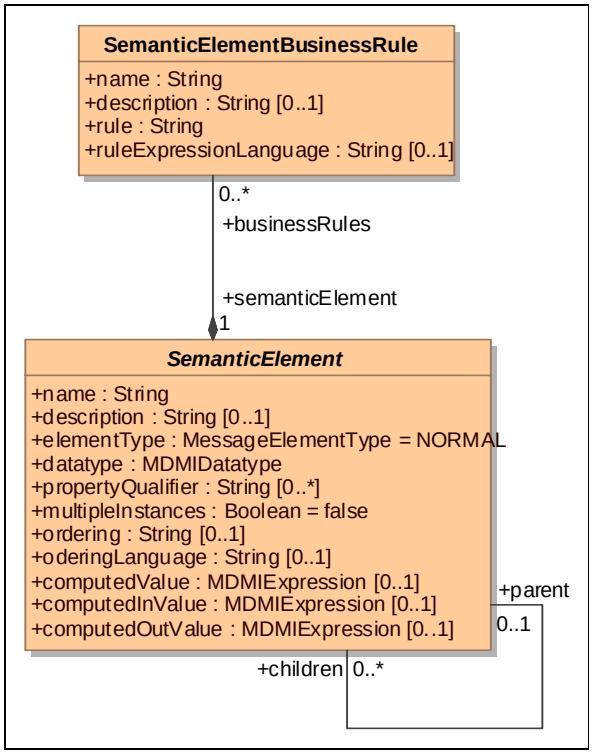

Figure 8.7 - SemanticElementBusiness Rule

8.7.3 SemanticElementBusinessRule - Detailed Semantics

Description

The SemanticElementBusinessRule holds a rule that is to be applied to a SemanticElement to make sure that the SemanticElement is valid. SemanticElementBusinessRule usually does not refer to other SematicElements in a message. They are meant to provide rules that reflect an external context, e.g., a “Primary AcountID” SemanticElement must be from an EU bank, etc.

Properties

1. A “name” property, of type String, assigns a name to the rule.
2. An optional “description” property, of type String, provides a description of the rule.
3. A “rule” property, of type String, is an expression defining a business rule or constraint.
4. A “ruleExpressionLanguage” property, of type String, is a reference to the expression language used in the “rule” property.

Associations

1. A (zero or many)-to-one association with the SemanticElement to which the MDMIBusinessElementRule applies.

8.8 Summary of Complete Metamodel

8.8.1 Overview

The complete metamodel is shown in Figure 8.8.

8.8.2 Abstract Syntax

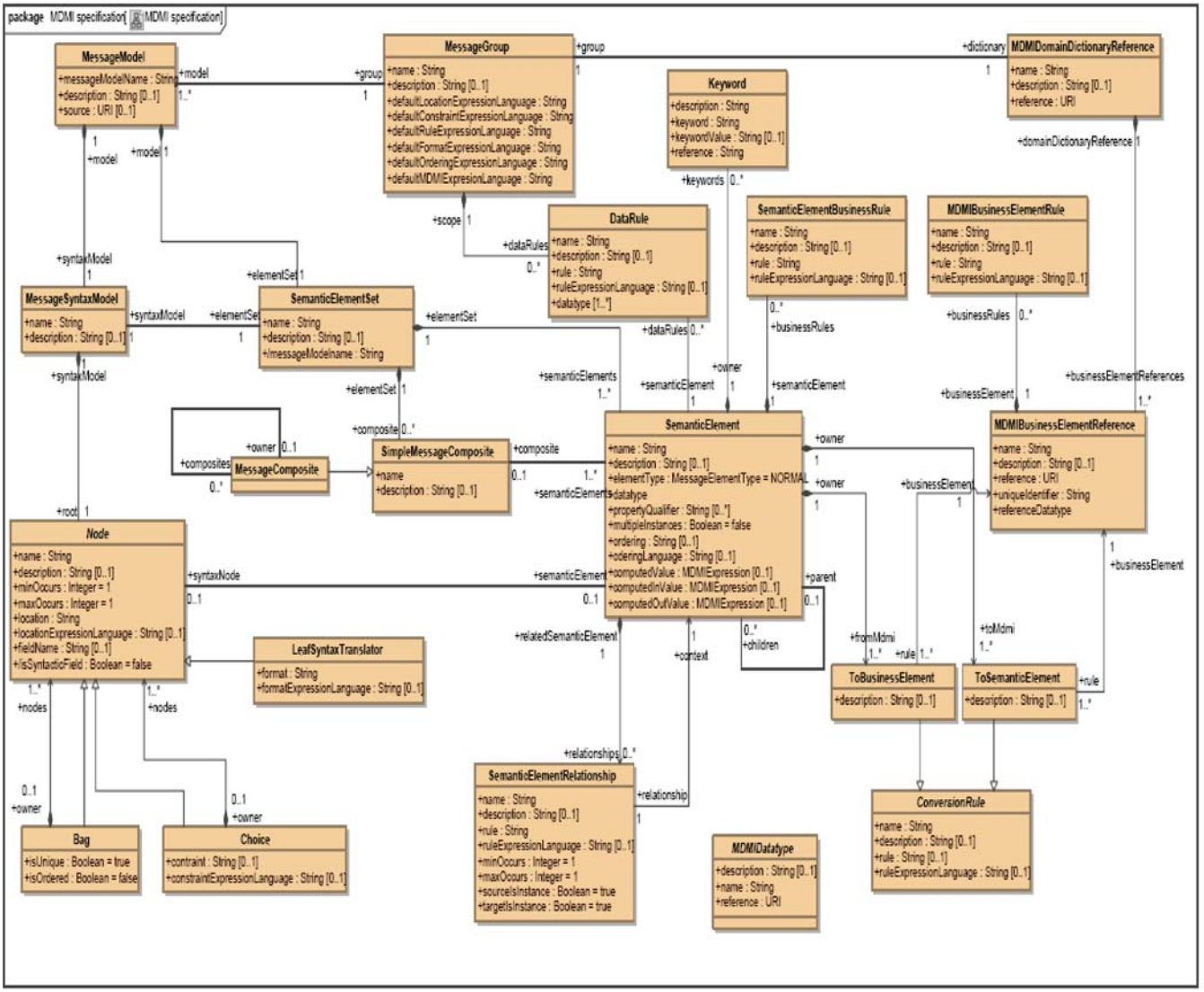

Figure 8.8 - Summary: Complete Metamodel

Annex A - List of Acronyms

Abbreviation	Notes
FIX	Financial Information eXchange http://www.fixprotocol.org
FpML	Financial products Markup Language is the industry-standard protocol for complex financial products. http://www.fpml.org
IFX	Interactive Financial eXchange www.ifxforum.org
MDDL	Market Data Definition Language www.mddl.org
NRL and NRL 1.0	Natural Rule Language - Open source constraint and action language based on OCL. The user guide can be found at http://nrl.sourceforge.net/userguide/userguide.htm
Swift	Society for Worldwide Interbank Financial Telecommunication supplies secure messaging services. http://www.swift.com
Twist	Transaction Workflow Innovation Standards Team www.twiststandards.org

INDEX

A
Acknowledgements 3
Additional Information 3
Artifacts 9

B
Bag 16, 18
Bag - Detailed Semantics 18
Bag associations 18
Bag properties 18
Bilateral Mapping 7
Business Element 2

C
Choice 16, 18
Choice - Detailed Semantics 18
Choice associations 18
Choice description 18
Choice properties 18
Complete Metamodel 34
Complex Datatype 24, 25
Compliance 1
Composition 2
Conformance 1
Conversion Rule 2
ConversionRule 29
ConversionRule - Detailed Semantics 29
ConversionRule description 29
ConversionRule generalizations 29
ConversionRule properties 29

D
DataRule 26
DataRule description 26
DataRules 24
DataRules - Detailed Semantics 26
DataRules associations 27
DataRules properties 26
Datatype 2
Definitions 2
Domain Dictionary 8

F
Federated Dictionary 2
Financial Information eXchange 35
Financial products Markup Language 35

I
Interactive Financial eXchange 35
ISO 20022 5
issues/problems iv

K
Keyword 23
Keyword - Detailed Semantics 23
Keyword associations 23
Keyword description 23
Keyword properties 23

L
LeafSyntaxTranslator 16, 19
LeafSyntaxTranslator description 19
LeafSyntaxTranslator properties 19
Lossless conversion 8

M
Market Data Definition Language 35
MDMI Expression Language 15
MDMIBusinessElementReference 27, 28
MDMIBusinessElementReference - Detailed Semantics 28
MDMIBusinessElementReference associations 29
MDMIBusinessElementReference description 28
MDMIBusinessElementReference properties 28
MDMIBusinessElementRule 30
MDMIBusinessElementRule associations 30
MDMIBusinessElementRule description 30
MDMIDatatype 24, 26
MDMIDatatype - Detailed Semantics 26
MDMIDatatype description 26
MDMIDatatype properties 26
MDMIDictionaryReference 13
MDMIDomainDictionaryReference 15
MDMIDomainDictionaryReference associations 15
MDMIDomainDictionaryReference description 15
MDMIDomainDictionaryReference properties 15
Message Format 2
Message Syntax Model 10
MessageComposite 19, 23
MessageComposite - Detailed Semantics 23
MessageComposite associations 24
MessageComposite description 23
MessageGroup 13, 14
MessageGroup - Detailed Semantics 14
MessageGroup associations 15
MessageGroup description 14
MessageGroup properties 14
MessageModel 13
MessageModel - Detailed Semantics 13
MessageModel associations 14
MessageModel description 13
MessageModel properties 13
MessageSyntaxModel 15, 16
MessageSyntaxModel - Detailed Semantics 16
MessageSyntaxModel associations 17
MessageSyntaxModel description 16
MessageSyntaxModel properties 17
Meta-Object Facility (MOF) 13
Moving data 6

MT103 MessageModel 13
MTxx 2
MXxx 2

N
Natural Rule Language (NRL) 35
Near Synonym 2, 8
Node 16, 17
Node - Detailed Semantics 17
Node class associations 18
Node class generalizations 17
Node description 17
Node properties 17
NRL 1.0 14

O
Object Management Group, Inc. (OMG) iii
OMG specifications iii

P
Physical Message Instance 3
Primary Account IDs 10

R
References 2
Remove the Syntax 10
Rule language 14

S
Scope 1
Semantic Element 3
Semantic Element Set 3
Semantic Map 3
SemanticElement 21
SemanticElement - Detailed Semantics 21
SemanticElement associations 22
SemanticElement description 21
SemanticElement Set 20
SemanticElement Set description 20
SemanticElementBusinessRule 32, 33
SemanticElementBusinessRule - Detailed Semantics 33
SemanticElementBusinessRule associations 33
SemanticElementBusinessRule description 33
SemanticElementBusinessRule properties 33
SemanticElementRelationship 31
SemanticElementRelationship - Detailed Semantics 31
SemanticElementRelationship associations 32
SemanticElementRelationship description 31
SemanticElementRelationship properties 31
SemanticElementSet 19
SemanticElementSet - Detailed Semantics 20
SemanticElementSet associations 20
SemanticElementSet properties 20
SimpleMessageComposite 19, 23
SimpleMessageComposite - Detailed Semantics 23
SimpleMessageComposite associations 23
SimpleMessageComposite description 23
SimpleMessageComposite generalization 23
SimpleMessageComposite properties 23
Society for Worldwide Interbank Financial Telecommunication 35
Source Semantic Element 11
Stage 1 7
Stage 2 7
Step 10
Straight Though Processing (STP) 1
SWIFT 150022 regular expression format language 14
Synonym 3

T
Target Semantic Element 11
TCxx 3
Terms and definitions 2
The SemanticElement properties 21
ToBusinessElement 30
ToBusinessElement associations 30
ToBusinessElement description 30
ToBusinessElement properties 30
ToSemanticElement 29
ToSemanticElement - Detailed Semantics 29
ToSemanticElement associations 30
ToSemanticElement description 29
ToSemanticElement properties 30
Transaction Workflow Innovation Standards Team 35
Typographical conventions iv

V
Versioning 6

X
XPath 2.0 14
XSD format attributes 14