

Model Driven Message Interoperability (MDMI) 2.0 Submission

OMG Proposal

OMG Document Number: health/20-03-02

Standard document URL: <https://www.omg.org/spec/MDMI/2.0/>

Normative Machine Consumable Files:

<https://www.omg.org/spec/MDMI/20200301/health-20-03-06.xmi>

Informative Machine Consumable Files:

<https://www.omg.org/spec/MDMI/20200301/MDMIGenericStatementModel.rdf>

<https://www.omg.org/spec/MDMI/MDMIGenericStatementModel.rdf>

<https://www.omg.org/spec/MDMI/20200301/MDMIExampleHealthcareDomainModel.rdf>

<https://www.omg.org/spec/MDMI/MDMIExampleHealthcareDomainModel.rdf>

This OMG document is in a revised proposal in response to the MDMI 2.0 Request for Proposal/health/2017-03-04

You may view the issues for this specification from the OMG revision issues web page

<http://www.omg.org/issues/>

Copyright © 2019, MDIX, Inc.

Copyright © 2019, Model Driven Solutions, Inc.

Copyright © 2019, Object Management Group, Inc.

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.

MDMI 2.0 Submission

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™, OMGT™, Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWMT™, CWM Logo™, IIOPT™, MOFT™, 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. If 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>).

MDMI 2.0 Submission

Table of Contents

Preface	ii	
About the Object Management Group		ii
OMG Specifications		ii
Typographical Conventions		iii
Issues		iii
Section 0 Submission-Related Information	iv	
1 Scope	1	
2 Conformance	1	
3 Normative References	1	
4 Terms and Definitions	2	
5 Additional Information	3	
5.1 Acknowledgements		3
6 Overview	4	
6.1 Different Ways to Use the Current Standard		5
6.1.1 Message Transformation: Moving Data from One Message to Another		5
6.1.2 Versioning		5
6.1.3 Moving Data from an Internal Enterprise Message Format to an External Standard		5
6.1.4 Design Considerations in Applications Requiring Message Transformation		6
6.2 Basic Approach for the Use of This Standard		6
6.2.1 Developing artifacts using the UML model		6
6.3 Adding a new MDMI Business Element to a MDMI Domain SEER		7
6.3.1 No Synonyms in a MDMI Domain SEER (MDMI Uniqueness Test)		7
6.3.2 No Hypernyms or Hyponyms (MDMI Precision Test)		7
6.4 Future Benefits of the Standard		7
6.4.1 360° View of diverse IT eco-system		7
6.4.2 Support of Business Processes Automation		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 MDMI Semantic Element to a Target MDMI Semantic Element using a Unique Identifier acquired from MDMI Domain SEER		11
8 UML Semantics – Normative Definition	12	
8.1 MessageModels, MessageGroup, MDMIDomainSEERReference		12
8.1.1 Overview		12
8.1.2 Abstract Syntax		12
8.1.3 MessageModel – Detailed Semantics		13
8.1.4 MessageGroup – Detailed Semantics		13
8.1.5 MDMIDomainSEERReference		14

8.1.6	DatatypeMap – Detailed Semantics	14
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	16
8.2.5	Bag – Detailed Semantics	17
8.2.6	Choice – Detailed Semantics	18
8.2.7	LeafSyntaxTranslator	18
8.3	MDMISemanticElementSet, MDMISemanticElement, SimpleMessageComposite	18
8.3.1	Overview	18
8.3.2	Abstract Syntax	19
8.3.3	MDMISemanticElementSet	20
8.3.4	MDMISemanticElement – Detailed Semantics	20
8.3.5	Keyword – Detailed Semantics	22
8.3.6	SimpleMessageComposite – Detailed Semantics	22
8.3.7	MessageComposite -- Detailed Semantics	22
8.4	MDMIDatatype, DataRules	23
8.4.1	Overview	23
8.4.2	An example of Complex Datatype	23
8.4.3	MDMIDatatype, DataRules – Abstract Syntax	25
8.4.4	MDMIDatatype – Detailed Semantics	25
8.4.5	DataRules – Detailed Semantics	27
8.5	MDMIBusinessElementReference, Conversion Rule, To MDMISemanticElement, 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	ToMDMISemanticElement – Detailed Semantics	30
8.5.6	ToMDMIBusinessElement	30
8.5.7	MDMIBusinessElementRule	30
8.6	MDMISemanticElementRelationship	31
8.6.1	Overview	31
8.6.2	Abstract Syntax	32
8.6.3	MDMISemanticElementRelationship – Detailed Semantics	33
8.7	MDMISemanticElementBusinessRule	33
8.7.1	Overview	33
8.7.2	Abstract Syntax	34
8.7.3.	MDMISemanticElementBusinessRule – Detailed Semantics	34
8.8	Summary of Complete Metamodel	35

MDMI 2.0 Submission

8.8.1	Overview	35
8.8.2	Abstract Syntax	35
Annex A - List of Acronyms		36
ANNEX B -		37
Informative		37
Healthcare Examples		37
Developing the MDMI Healthcare Concept Model		37
Step 1: Scope of MDMI Business Elements for informative model		37
Step 2: Select the MDMI Reference Model		37
Step 3: Assigning the MDMI StatementContext property and the DataElementConcept property in the MDMIBusinessElementReference		37
Step 4: Running the MDMI Acceptance Test		38
MDMI Healthcare Concept Model		39
Example of the StatementContext and DataElementConcept properties for MDMI Business Elements		40

MDMI 2.0 Submission

Preface

About the Object Management Group

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

- CORBA services
- CORBA facilities
- 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:

MDMI 2.0 Submission

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*](mailto: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>.

Section 0 Submission-Related Information

0.1 Submission Introduction

This is a revised submission to the OMG document health19-10-01, MDMI 2.0 Specification Request for Proposal. The submission is called Model Driven Message Interoperability (MDMI) 2.0 Submission.

0.1.1 Copyright Waiver

Each of the entities listed above: (i) grants 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, and (ii) grants to each member of the OMG a nonexclusive, royalty-free, paid up, worldwide license to make up to fifty (50) copies of this document for internal review purposes only and not for distribution, and (iii) 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 any OMG specification that may be based hereon or having conformed any computer software to such specification.

0.1.2 IPR Mode

The Submitters specify they will comply with the OMG Rand Mode as specified in the OMG document ipr/12-09-02.

0.1.3 Proposed Identification

The proposed acronym for this specification is: MDMI

A normative section: <https://www.omg.org/spec/MDMI/2/health/20-02-01>

An XMI artifact, part of the normative submission:
<https://www.omg.org/spec/MDMI/2/health/20-02-03>

An OWL artifact (Generic), part of the informative submission:
<https://www.omg.org/spec/MDMI/2/health/20-02-04>

An OWL artifact (Healthcare example), part of the informative submission:
<https://www.omg.org/spec/MDMI/2/health/20-02-05>

Artifacts supporting the informative submission:
<https://www.omg.org/spec/MDMI/2/health/20-02-06>

0.2 Submission Team

The following organizations contributed to the development of this submission. Submission lead was MDIX, Inc. General questions regarding this submission shall be directed to MDIX, Inc.

0.2.1 Submitting Organizations

- MDIX, Inc.
Mr. Kenneth Lord
Klord@mdixinc.com
- Model Driven Solutions, Inc.
Mr. Cory Casanave
cory-c@modeldriven.com

MDMI 2.0 Submission

0.2.2 Acknowledgements

The authors would like to acknowledge the significant financial and guidance contributions from the

- Veterans Health Administration:
 - Ken Rubin
Director of Standards & Interoperability, Knowledge Based Systems, Clinical Informatics and Data Management Office, VHA
 - Robert Lario
Enterprise architect, Knowledge Based Systems, VHA

the KBS team of BookZurman;

- Sean Muir
 - Matthew Lord
 - Thomas Beale
 - Ioana Singureanu
 - Mathew Horridge
-
- Thematix, Inc
 - Elisa Kendall
- and
- the Mayo Clinic:
 - Davide Sottara

This work would not have been possible without their assistance.

0.3 General Requirements

The submitters declare the internationalization general requirement should be classified as Uncategorized.

0.4 Relationships to other OMG Specifications and Activities and non-OMG Activities

All specifications listed in the RFP were considered.

0.5 Mandatory requirements and non-mandatory features

The proposal addressed all mandatory requirements and non-mandatory features listed in the RFP. It should be noted that

- No content of the referent index (renamed the SEER) has been provided although some is included in the information section, Annex B.
- requirement 6.5.5 was already addressed in the MDMI 1.0 Standard.
- Requirement 6.5.6 was addressed by additional properties to the MDMIBusinessElementReference and the MDMIDomainSEER classes.

0.6 Issues to be discussed

0.6.1 The language below consolidates issues 6.7.1, 6.7.2, 6.7.3, and 6.7.5, which focus on the use of 11179, ODM/OWL.

During development of this Proposal, the ISO 11179, the OMG ODM, and the W3C OWL Standard were all investigated. A new requirement was formulated out of this work driven by the requirement that MDMI Business Elements must have a search capability to discover precise MDMI Business Elements. Also a new requirement formulated was that MDMI 2.0 must leverage existing healthcare ontologies and terminologies to 1) achieve adoption and 2) provide the basis of accurately searching and discovering the precise meaning of a MDMI Business Element.

The RFP recommended in Section 6.6.4 the submission may “include domain-specific reference content, such as transformation maps or a populated Referent Index. Note that these artifacts shall be informative and not normative”. This submission provides an informative domain-examples of the SEER in the healthcare. In addition, artifacts for the MDMI Healthcare Description Model are also provided. It was determined that the work from the Veterans Health Administration, HL7, and the Logica Consortium that the Analysis Normal Form (ANF) Model would serve as the primary Reference Model for the informative Annex B. ANF provides a model for classifying clinical statements that assist in determining precision of a MDMI Business Element and is based on the foundation of three important ontologies and

MDMI 2.0 Submission

terminologies; SNOMED, LOINC, and RxNorm. Additionally, this work effort is gaining the support within HL7 and the healthcare industry consortium of Logica.

It is important to note that there is not one ontology in healthcare that provided the scope and precision required to accurately define the meaning of MDMI Business Elements in the existing MDMI Healthcare SEER.

Much work was done investigating and using the ISO 11179 Part 3 Classification Scheme. The Proposal Team has used as informative many of 11179's principles, including the Data Element Concept, Concept, and Context classes. OWL was used as the expression language for the MDMI Healthcare Description Model because of the importance of providing a mechanism to search and discovery meaning by Clinical Decision Support applications using products that utilize the OWL technology stack, its growth as the "language of choice" in healthcare, its ease to extend and specialize the MHDM in the future, and perhaps most importantly, its ability to create a precise, computable, and unambiguous formalism for the meaning of a MDMI Business Element.

0.6.2 Other enhancements in Proposal

A general requirement is ease of use. MDMI has been used successfully on dozens of different healthcare standard and proprietary formats. A key productivity issue was the ability to automatically generate the syntactic classes using artifacts (e.g., .xml or .csv files). These files do not generally provide support for complex datatypes. Consequently, mapping data elements with primitive data types to more complex datatypes in the MDMI Business Element Reference Datatypes becomes very tedious and error-prone. For this reason, a Datatype Mapping class was added in Section 8.1.2.

1 Scope

The transmission of information across systems in multiple enterprises may rely on standardized messages. Different industry domains involve different standardized messages and different versions of standardized messages. Some examples of standardized information formats utilized in healthcare services are CDA, HL V2, FHIR, and X.12. Some examples of standardized information formats utilized in financial services are FIX, FpML, IFX, TWIST, SWIFT, Visa, and ACORD.

This information must be correctly interpreted and processed by each system involved at each step of the 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 format to another (e.g., from a HL7 V2.5.1 ADT message into a FHIR 4 Bundle, or from a FIX pre-trade message into a SWIFT settlement message). In addition, an 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. Historically, the mapping of 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 a new message format, and even changes to older messages, is very expensive. The goal of the MDMI standard is to provide a declarative, model-driven mechanism to perform message data transformation to handle the movement of data between different message formats as well as to support versioning by providing a mechanism to map information between a new and an older version of the same message. Thus, the MDMI standard can help reduce the barriers that prevent the introduction of new versions of messages and thereby greatly reduce the cost of change.

The Healthcare Domain Task Force wishes to emphasize that this specification is intended for use by both the healthcare community and other communities (e.g., the financial-services community) and has developed MDMI with these requirements in mind. The concepts, models, and mechanisms described in this specification can be applied or adapted to other application domains.

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 ; to utilize expression languages that are consistent with the constraints described in section “8.1.4 MessageGroup – Detailed”; to utilize MDMIDatatypes that are consistent with the description and constraints in section 8.4; and to utilize a central repository 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 figures 7.1 and 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 MDMI Semantic Element from a source message to a target message.

3 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

OMG2016 <https://www.omg.org/spec/MOF>

OMG2017 <https://www.omg.org/spec/UML/>

The following informative documents, through reference in this text, contain starting points or material in assisting this document in achieving its goals and objectives. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

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

This specification references ISO 11179. A complete reference for ISO 11179 can be found at <https://www.iso.org/standard/60341.html>

This specification references HL7 Analysis Normal Form (ANF), information about HL7 ANF can be found at <https://confluence.hl7.org/display/CIMI/Analysis+Normal+Form+%28ANF%29+Project>

This specification does not specifically reference, although the following were informative in the development of this specification:

The OMG ODM Standard formal/14-09-02 that can be found at <http://www.omg.org/spec/ODM/1.1>.

The ISO 704 Standard, Terminology work – Principles and methods, second edition, Reference number ISO 704:2000(E)

4 Terms and Definitions

MDMI Business Element

A MDMI Business Element are a unit representing the smallest business concept exchanged between a source format and a target format. . In healthcare, a MDMI Business Element in models such as ANF, CDA, and FHIR are for business concepts such as PatientID or MedicationAdministeredDateTime. In ISO20022, MDMI Business Elements are the attributes of Business Component (or their related Message) classes and represent a “business concept.”

CDA

HL7 Clinical Document Architecture Standard, defines construct for e-health documentation.

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 is to be applied to convert a value of a source MDMI Semantic Element into a value of a target MDMI Business Element or a target MDMI Semantic Element.

Datatype

A prescription of the form of the data that has no specific message format related business concept. (e.g., an address, a date, etc.).

FHIR (Fast Health Interoperability Resources)

A REST-based standard from HL7 for data access and representation of health information.

HL7 V2

MDMI 2.0 Submission

This HL7 messaging standard allows the exchange of clinical data between systems.

MDMI Acceptance Test

A set of tests to determine if MDMI Business Elements in a MDMI Domain SEER have a synonym, hypernym, or hyponym.

MDMI Domain Semantic Element Exchange Registry (SEER)

A repository of MDMI Business Elements that contains the set of MDMI Business Elements that represent the business concepts that have been identified in transforming messages for an identified domain. An example of the MDMI Domain SEER is the MDMI Healthcare SEER.

MDMI Healthcare Concept Model

Provided as an informative model used to define the StatementContext and the DataElementConcept properties in the MDMIBusinessElementReference Class in the MDMI Healthcare SEER.

Message Format

Definition of the syntax and information of a class of messages. It 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.

Physical Message Instance

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

MDMI 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 MDMI Semantic Elements would represent the columns of that table.

MDMI Semantic Element Set

A set of MDMI Semantic Elements, Message Composites and Simple Message Composites and MDMI Semantic Element Relationships in a message format.

MDMI Semantic

The term MDMI semantic is used as an adjective in this specification should not be interpreted as formal. It should be understood to represent a textual representation.

MDMI Semantic Map

A map that describes the relationship between a MDMI Semantic Element in a MDMI Semantic Element Set and a MDMI Business Element in a MDMI Domain SEER.

Synonym

A MDMI Business Element that is a simple equivalence to another MDMI Business Element, i.e., A=B.

TCxx

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

5 Additional Information

5.1 Acknowledgements

The following companies submitted and/or supported this specification:

MDIX, Inc.
Model Driven Solutions, Inc.

The submitters would like to acknowledge and thank those that have contributed to the submission: Veterans Health Administration: Elisa Kendall of Thematix, Inc, Ken Rubin and Robert Lario of the US Veterans Administration, Davide Sattoro of Mayo Clinic, Thomas Beale of OpenEHR, Richard Beach of Bloomberg, Matt Lord and Sean Muir of MDIX, and Mathew Horridge of Book Zurman.

6 Overview

Given the lack of mapping standards for both the healthcare and financial industries, data is usually mapped directly from one information 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, errors are easily introduced since many of these mappings are done locally and procedurally.

The result of current mapping practices leads to a lack of interoperability, high costs, and/or inaccurate information for the consumer of the mapping transaction. Virtually all organizations face this situation and they may spend a good deal of their software-development budgets on creating new interfaces and mappings or extending existing ones. Or, they may resist introducing any changes into existing message formats. Adoption of new formats becomes more difficult due to the cost of changing applications that process the older message formats.

The goal of the MDMI standard is to provide a standard framework and methodology for the mapping between information models in the healthcare, financial, and other industries, which will alleviate the mapping problem.

This standard 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 in workflow applications involving multiple services that are based on multiple information formats or versions of formats.

The MDMI standard's framework is based on two concepts:

- First, removing any syntax associated with a message format, revealing the set of core “MDMI Semantic Elements” contained in that message format. A MDMI Semantic Element is the smallest unit of a business concept defined in a message format.
- Second, specifying a MDMI Semantic Map of those MDMI Semantic Elements to an industry-accepted repository consisting of MDMI Business Elements. A MDMI Business Element represents the smallest business concept for the industry sector that is an entry in the repository. As a business concept used in transformations, the MDMI Business Element is a representation of the context in which was recorded (e.g., a patient visit or requesting a bank wire) as well as the data element concept (e.g., date or location of the visit; who the doctor was in the patient visit or the amount to be sent in a bank wire; the account receiving the wire; or the account where the money is to be sent from).

MDMI 2.0 Submission

The easiest way to recognize MDMI Semantic Elements or MDMI Business Elements is that they cannot be constructed from other MDMI Semantic Elements or MDMI Business Elements, respectively (i.e., they are represented by a class, whose primary property is a general data type).

Providing mapping between the MDMI Semantic Element to a MDMI Domain SEER creates a “hub and spoke.” A mapping then will have two steps, utilizing a map from a source to the MDMI Domain SEER and then utilizing a map from the MDMI Domain SEER to the target. Thus, the mapping process is reduced from being geometric to being linear with the number of message formats.

6.1 Different Ways to Use the Current Standard

6.1.1 Message Transformation: Moving Data from One Message to Another

The primary focus of the MDMI standard is moving information from a source in one format to a target in a different format. For example: One format may define a “patient address” field while another format may have separate fields for “patient street,” “patient city,” “patient state,” etc. One format may define a bank ID number as a BIC number while another format may define a bank ID as an ABA routing number.

The key is that the fields in each format are mapped to the same repository element or a canonical representation: the MDMI Business Element. There are two important benefits of mapping to a canonical representation:

1. This approach 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 n^2 mappings required for bilateral transformations. For example, by using a central data repository 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, and ACH messages, whereas 15 bilateral conversion maps would be needed.
2. Using MDMI one only needs to be expert in its own message formats and the well-defined semantics of the canonical representations in the MDMI Domain SEER, rather than needing to understand the business concepts and syntax of many other message groups to perform message transformations.

6.1.2 Versioning

A second costly problem in the healthcare and financial services spaces is versioning. Given the legacy of existing software, even a small change in a message format can be 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 recoded.

6.1.3 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 an “information format” and maps can be generated that transform data from that internal database to an external standard. Whenever either message format changes, these maps must be changed. With MDMI maps, the MDMI Semantic Elements in internal formats are mapped to a canonical representation and therefore can be isolated from external changes.

6.1.4 Design Considerations in Applications Requiring Message Transformation

MDMI can be used to produce artifacts that assist development teams implementing applications requiring transformations. For example, a new healthcare application built using a new message model may need to interact with existing data stores and/or existing industry messaging standards. MDMI has been used to produce gap analysis and traceability reports for design teams even before the implementation phase.

6.2 Basic Approach for the Use of This Standard

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

6.2.1 Developing artifacts using the UML model

6.2.1.1 Stage 1

The first stage artifacts utilize a Message Syntax Model to create a syntax-neutral set of MDMI Semantic Element classes. MDMI Semantic Elements are the business concepts/entities contained in a message format, for which further parsing would leave only generic data-type values.

6.2.1.2 Stage 2

The second stage provides a mapping of these MDMI Semantic Elements to a MDMI Business Element. It does this by specifying To and From Conversion Rules for source MDMI Semantic Elements to MDMI Business Elements in MDMI Domain SEER.

In most 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 may need to be defined for the mapping 1) from a source MDMI Semantic Element to a MDMI Business Element as compared to 2) from a MDMI Business Element to a MDMI Semantic Element.

Examples in healthcare are:

- Mapping the “MRN” element in the source message to the Business Element, “Patient Identifier” in the repository
- Mapping “MedicationAdministeredDateTime” in the MDMI Healthcare SEER to “MedicationEffectiveTimeLow” and setting “MedicationEffectiveTimeHigh” to a null value in the target

For example, in financial services:

- Mapping “Primary Client Identifier” element in the source message to the two elements, “Primary Client Name” and “Primary Client BIC,” in the repository
- Mapping “Primary Account Beginning Balance” and “Primary Account Ending Balance” in the repository to “Primary Account Beginning Balance” and “Primary Account Debited Amount” in the target

6.3 Adding a new MDMI Business Element to a MDMI Domain SEER

In the above Section 6.2.1.2 Stage 2, the MDMI Semantic Element may represent a concept not yet included in the MDMI Domain SEER. If this is the situation, a new MDMI Business Element may need to be added to the MDMI Domain SEER. For a MDMI Domain SEER, there are important principles for adding a MDMI Business Element. The business reason for these principles is independent organizations can and have developed MDMI Maps for their specific Message Models. It is a fundamental precept that it should not be possible to correctly map their MDMI Semantic Element to different MDMI Business Elements. The MDMI Acceptance Test is processed using the computable Statement Concept property and the DataElementConcept property of the MDMIBusinessElementReference Class to ensure this fundamental precept there is only one correct mapping between MDMI Semantic Elements and MDMI Business Elements is maintained. (See also Annex B: Informative: Healthcare Examples for more information on the process used for adding a new MDMI Business Element.)

There are two different functional tests in the MDMI Acceptance Test:

6.3.1 No Synonyms in a MDMI Domain SEER (MDMI Uniqueness Test)

If there were two MDMI Business Elements that are synonyms, it would be possible for one organization to correctly associate a MDMI Semantic Element to one MDMI Business Element ("BE_A") and another organization to correctly associate their MDMI Semantic Element to a different MDMI Business Element ("BE_Z"). An MDMI runtime engine would not be able to execute a transformation because the transformation requires alignment of the source and target MDMI Semantic Elements to the same MDMI Business Element to complete the transform. In this example, MDMI would not be able to move the data in the source format into the target format.

6.3.2 No Hypernyms or Hyponyms (MDMI Precision Test)

There is a second scenario, similar to the above but different, that is to avoid correctly mapping a MDMI Semantic Element to two different MDMI Business Elements. This can occur when there are two different MDMI Business Elements that have the same meaning and one MDMI Business Element is more precise than the other. An example of this would be if there were one MDMI Business Element called PatientAddress and another called PatientHomeAddress. The MDMI Precision Test is to ensure that this scenario is not possible.

6.4 Future Benefits of the Standard

There are several future uses and extensions to the MDMI standard that should enhance the value of the standard.

6.4.1 360° View of diverse IT eco-system

Using the MDMI Domain SEER, one can query using the StatementContext property and the DataElementConcept to find the set of MDMI Business Elements. Using available MDMI Maps, it will be possible for these MDMI Business Elements to be traced to multiple sources where that business concept is used by the enterprise or even eco-system of multiple enterprises. This will be possible because within each MDMI Map there is a direct association from the MDMI Business Element to the MDMI Semantic Element to the MDMI Syntax Node that has the property of physical location of the data element in that file.

6.4.2 Support of Business Processes Automation

In the design of business processes using the OMG BPM+ Standard as well as other OMG business process standards (e.g., BPMN, CMMN, and DMN), data objects and data files are specified. These objects files can be either the source or target format. The StatementContext property and the DataElementConcept are searchable metadata to discover the appropriate MDMI Business Element and make a direct association between the item in a Data Object or Data File to the MDMI Business Element. It is then possible to create MDMI maps for these process models that can be used to as part of a transformation with other MDMI Maps (e.g., industry-standard MDMI Maps or proprietary MDMI Maps). This enables these business process models to be Platform Independent Models and the MDMI standard provides an MDA approach to create Platform Specific Models (PSMs) .

6.4.3 Handling Lossless Conversion

An important need in messaging is dealing with the loss of information when performing conversions. This problem likely will never be completely solved but improvements in lossless conversions will be a great benefit. The proposed artifacts for the MDMI standard 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 MDMI Semantic Elements; etc.).

7 Use of MDMI Artifacts Overview

The focus of the MDMI standard 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 involving thousands of data elements. The standard 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 standard, 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 Standard. The rectangles in the diagram represent these artifacts. In addition, the MDMI Business Elements in Figure 7.1 are the same MDMI Business Elements as in Figure 7.2 and that these MDMI Business Elements are defined in a MDMI Domain SEER.

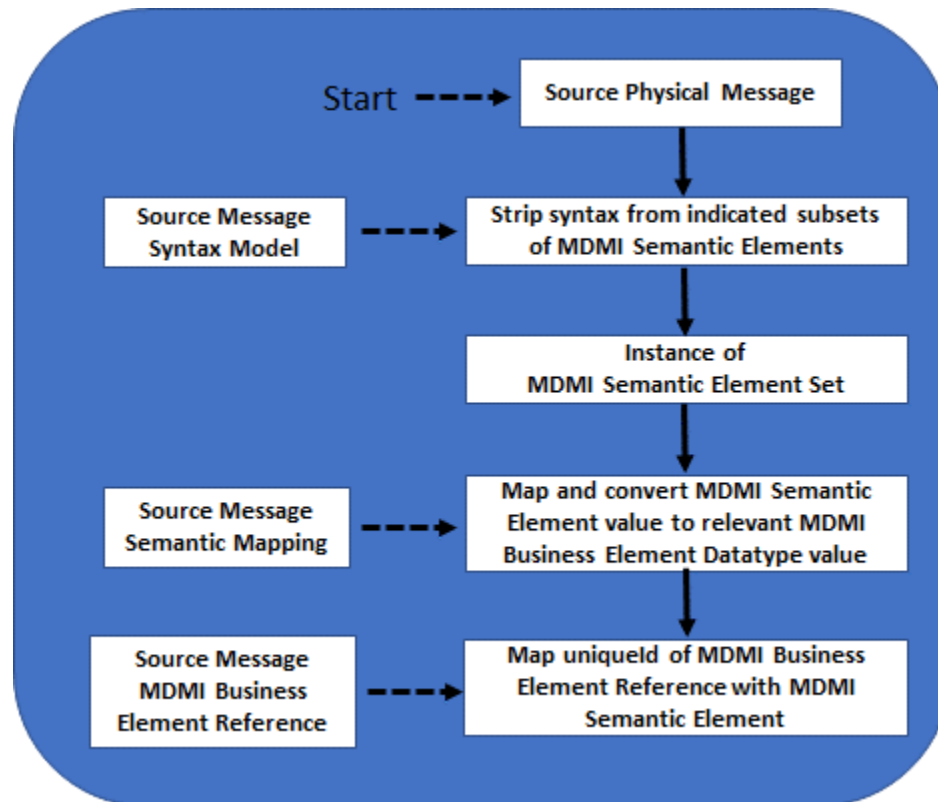


Figure 7.1 – Overview of run-time conversion methodology from Source

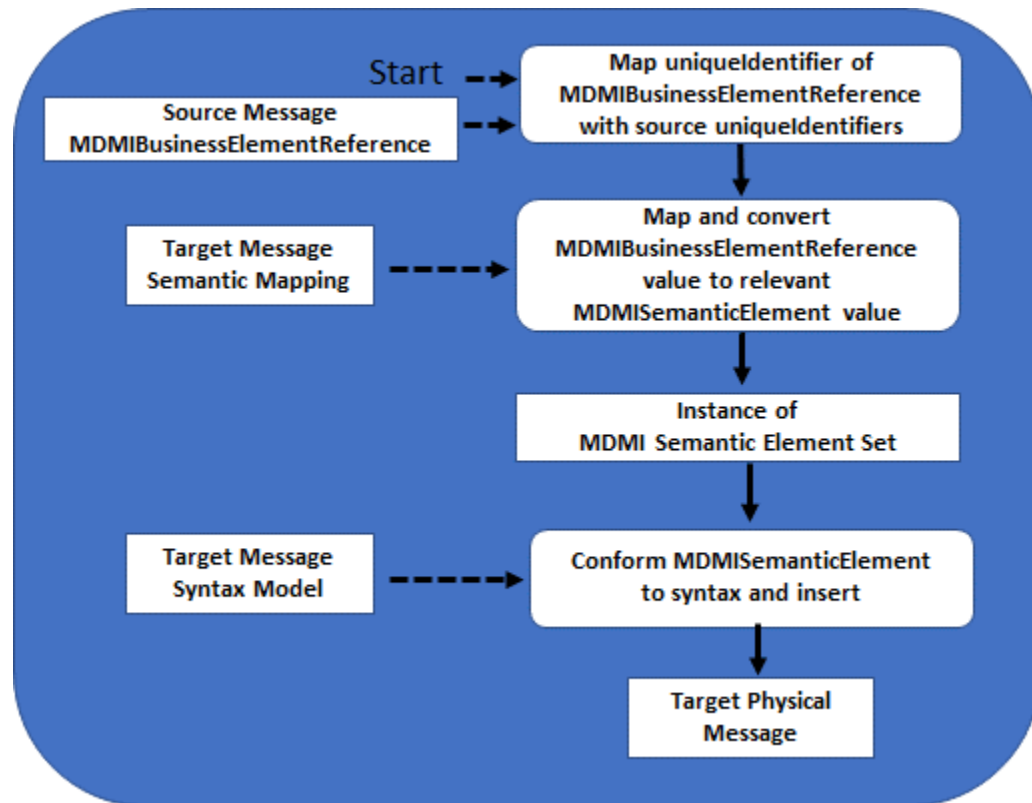


Figure 7.2 – Continuation of Overview of run-time conversion to Target

The following steps describe this conversion example.

7.1.1 Step 1 – Remove the Syntax

The first step of a conversion is to convert the source data in a physical message instance (e.g., a HL7 V2.5.1 ADT-A01, 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 Standard artifact, “Message Syntax Model.” The Message Syntax Model provides a syntactic description that contains the necessary information to extract or insert any data item from/to a physical message instance.

A data item in a message is defined as the smallest business concept in a message for which further parsing would lose meaning, leaving only generic datatype values. For example, in a HL7 V2.8.2 ADT-A01 there is a field representing Admit Date/Time. If further parsing was done, the value left would simply be a date and indistinguishable, in a business context, from any other date. Therefore, Admit Date/Time is a data item that is a smallest business concept.

Normally, the smallest business concept in a message is a field but in many overloaded message formats, a business concept can be a sub-field. In existing message formats, many “fields” have been subdivided into numerous business concepts. In healthcare, a field may represent a MedicationDate; however, depending on another field, the MoodCode, the MedicationDate may represent the date the medication was administered, or it may represent the date the medication was ordered. In a financial-services 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.

MDMI 2.0 Submission

When the data is stripped of its specific message format syntax, its value will be represented by an instance of the artifact “MDMI Semantic Element.” There will be a MDMI Semantic Element class defined for every business concept contained in a message’s message format. All these the MDMI Semantic Element classes are contained in the “MDMI Semantic Element Set” by composition.

7.1.2 Step 2 – Mapping a Source MDMI Semantic Element to a Target MDMI Semantic Element using a Unique Identifier acquired from MDMI Domain SEER

The second step for the conversion leverages a MDMI Domain SEER to define the relationship between one or more Source MDMI Semantic Elements and one or more Target MDMI Semantic Elements.

The Source and Target MDMI Semantic Elements are associated with a MDMI Business Element in a MDMI Domain SEER through a MDMI 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 attribute in the MDMI Semantic Element class, a Data Relationship Rule, or a Conversion rule. Each element in the MDMI Domain SEER must provide a unique identifier for its MDMI Business Elements. That unique identifier will be stored in the MDMI Business Element references that are associated with MDMI Semantic Elements. The appropriate Unique Identifiers will have been stored in the MDMI map for all MDMI Semantic Elements in the 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 the MDMI Semantic Elements that have matching Unique Identifiers.

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 MDMI Semantic Elements in a message. For example, it may be invalid to store an “Account Balance amount” if there is no value for an “Account ID.” Therefore, the maps for each MDMI Semantic Element can include a set of MDMI Semantic Element Relationships that describe the relationship of a MDMI Semantic Element with all other MDMI Semantic Elements in the message. A runtime application uses the MDMI Semantic Element Relationships in its target mapping to ensure 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 Model Driven Message Interoperability standard. 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, MDMIDomainSEERReference

8.1.1 Overview

This view presents the MessageModel, the MessageGroup and the MDMIDomainSEERReference. 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 messaging domain. An MDMIDomainSEERReference provides a reference to the MDMI Domain SEER Reference to which the MDMI Semantic Elements for all MessageModels in the MessageGroup will be mapped.

8.1.2 Abstract Syntax

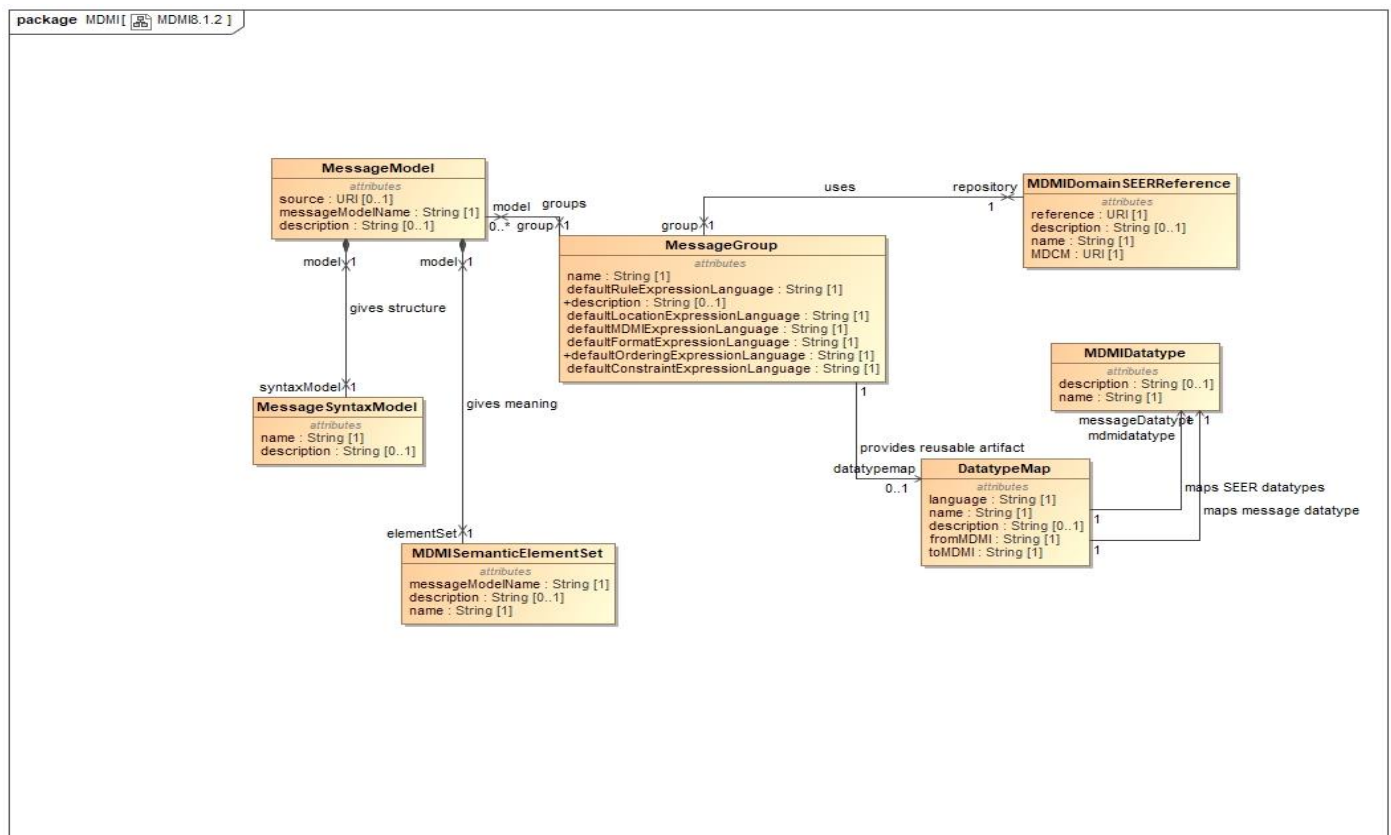


Figure 8.1 – Message Model, MessageGroup, MDMIDomainSEERReference

8.1.3 MessageModel – Detailed Semantics

MessageModel 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, JSON files, and UML documents.

MessageModel 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 a HL7 V2.8.2 ADT-A01 MessageModel could undoubtedly be “ADT-A01-2.8.2.”
2. An optional “description” property, of type String, contains a description of the message model.
3. A “source” is a property, of type URI, that 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 HL7 FHIR or ISO 20022, or it might reference a paper document.

MessageModel associations:

1. A MessageModel has a MessageSyntaxModel by composition.
2. A MessageModel has a MDMISemanticElementSet by composition.
3. A MessageModel is associated with a MessageGroup.

8.1.4 MessageGroup – Detailed Semantics

MessageGroup description: The MessageGroup class contains a set of message models that are considered in the same grouping (e.g., HL7 V2 ADT messages, CDA documents, FHIR profiles, SWIFT 15022 messages, FIX security messages, etc.). The MessageGroup class is useful for setting various defaults for closely related message formats.

The MessageGroup 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.
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 MDMISemanticElement and thus it must be able to access the fields in complex datatypes.
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, 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 MDMISemanticElements in which the Boolean property “multipleInstances” is “True.” The ordering language should provide expressions that can be evaluated to both cardinal and ordinal positioning.
8. The property “defaultMDMIExpressionLanguage,” of type String, identifies the computational language to be used as a default for specifying the computational expression in computed MDMISemanticElements that are of type MDMIExpression.

MessageGroup 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 MDMIDomainSEERReference that identifies the registry utilized by the group;
4. An association with one or more Datatype Maps.

8.1.5 MDMIDomainSEERReference

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

MDMIDomainSEERReference properties:

1. A “name” property, of type string, that provides a name for the referenced MDMI Domain SEER.
2. An optional “description” property, of type String, that provides a description of the MDMI Domain SEER.
3. A “reference” property, of type URI, that provides a reference to a MDMI Domain SEER, such as a URL.
4. The MDCM property, of type URI, references the MDCM that is used to define the StatementContext property and the DataElementConcept property in the MDMIBusinessElementReference Class. See Annex B for an example of MDCM for Healthcare.

MDMIDomainSEERReference associations:

1. MDMIDomainSEERReference has a one-to-one association with MessageGroup to indicate the MDMI Domain SEER that will be used for the maps in MessageModels in the MessageGroup.
2. MDMIDomainSEERReference has a one-to-many relationship to the MDMIBusinessElementReference class so that a reference to the MDMI Domain SEER, to which a MDMI Business Element belongs, is easily accessed.

8.1.6 DatatypeMap – Detailed Semantics

DatatypeMap description:

The DatatypeMap provides the ability to roll up simple datatypes contained in a message format into a complex Datatype that has been defined for the message format. This is an optional capability. As an example, a complex message format, because of different roles and contexts, may have hundreds of

MDMI 2.0 Submission

different persons and organizations who have an address represented by properties of street1, street2, city, state, county, postal code and country. The DatatypeMap allows mapping of these attributes to a complex datatype of Address containing these seven properties.

The DatatypeMap properties:

1. A “name” property, of type string, that provides a name for the DatatypeMap.
2. An optional “description” property, of type String, that provides a description of the DatatypeMap.
3. A “ToMDMI” property, of type string, using the language property to create a one-to-one mapping from the datatype in the message format to MDMIDatatype.
4. A “FromMDMI” property, of type string, using the language property to create a one-to-one mapping from the MDMIDatatype to messageDatatype in the message format.
5. A “mdmiDatatype” property, of type MDMIDatatype, used in the ToMDMI property and the FromMDMI property.
6. A “messageDatatype” property, of type MDMIDatatype, used in the ToMDMI property and the FromMDMI property.
7. A “language” property, of type string, is a reference to the expression language used in the “To” and “From” property of the DatatypeMap class.

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 MDMISemanticElement 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 leaves 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 MDMISemanticElement. The MDMI standard 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: XML, JSON, FHIR path, 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

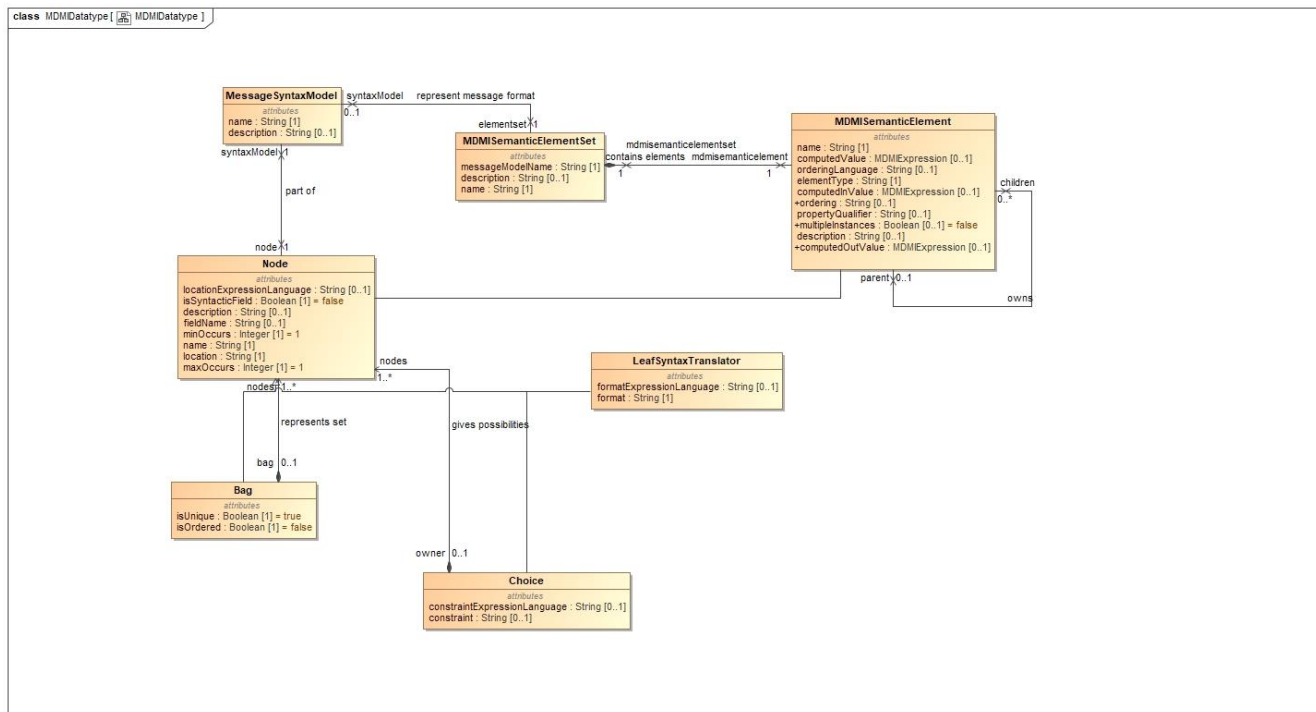


Figure 8.2 – Message Syntax Model

8.2.3 MessageSyntaxModel – Detailed Semantics

MessageSyntaxModel description:

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

MessageSyntaxModel 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.

MessageSyntaxModel associations:

1. An association with one-to-many Nodes as it is the parent class of the syntax tree.
2. An association with its parent MessageModel
3. An association with its sibling MDMISemanticElement Set

8.2.4 Node – Detailed Semantics

Node 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.

Node properties:

MDMI 2.0 Submission

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 “isSyntacticField,” of type Boolean, indicates that if the property’s value is “True,” then 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 MDMISemanticElement. This is the key association that links a MDMISemanticElement to its syntax.

8.2.5 Bag – Detailed Semantics

Bag description:

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

Bag properties:

1. The “isUnique” property, of type Boolean, indicates, if its value is “True,” then 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 MDMISemanticElements is handled 1) using the “location” property in the Node class and 2) using the “ordering” property in the MDMISemanticElement class.

Bag 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

Choice description:

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

Choice 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.

Choice 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

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.

LeafSyntaxTranslator 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 a list of fields that occur within a string. The MDMI standard does not require a specific formatExpressionLanguage.

8.3 MDMISemanticElementSet, MDMISemanticElement, SimpleMessageComposite, MessageComposite, Keyword

8.3.1 Overview

The MDMISemanticElementSet contains a set of MDMISemanticElement classes. Each MDMISemanticElement represents a smallest business concept in a message format. The MDMISemanticElementSet 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 MDMISemanticElements in a message are stored in the MDMISemanticElementSet 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 CDA and a

MDMI 2.0 Submission

goal of healthcare interoperability, only data items in the document like a HospitalSummaryDischarge that are to be moved into a FHIR composition need to be mapped.

The MDMISemanticElementSet represents the “flattening,” or “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 MDMISemanticElementSet are MDMISemanticElements. A couple of additional classes are provided primarily for the ease when creating an MDMI Map but they do not play a major role in the conversion process. These are SimpleMessageComposites and MessageComposites. These classes are conveniences for bundling MDMISemanticElements in the design process.

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

A MessageComposite is an aggregation that contains MDMISemanticElements, SimpleMessageComposites and MessageComposites. It is therefore possible 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 business concepts, which is a core design principle of the MDMI standard.

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

8.3.2 Abstract Syntax

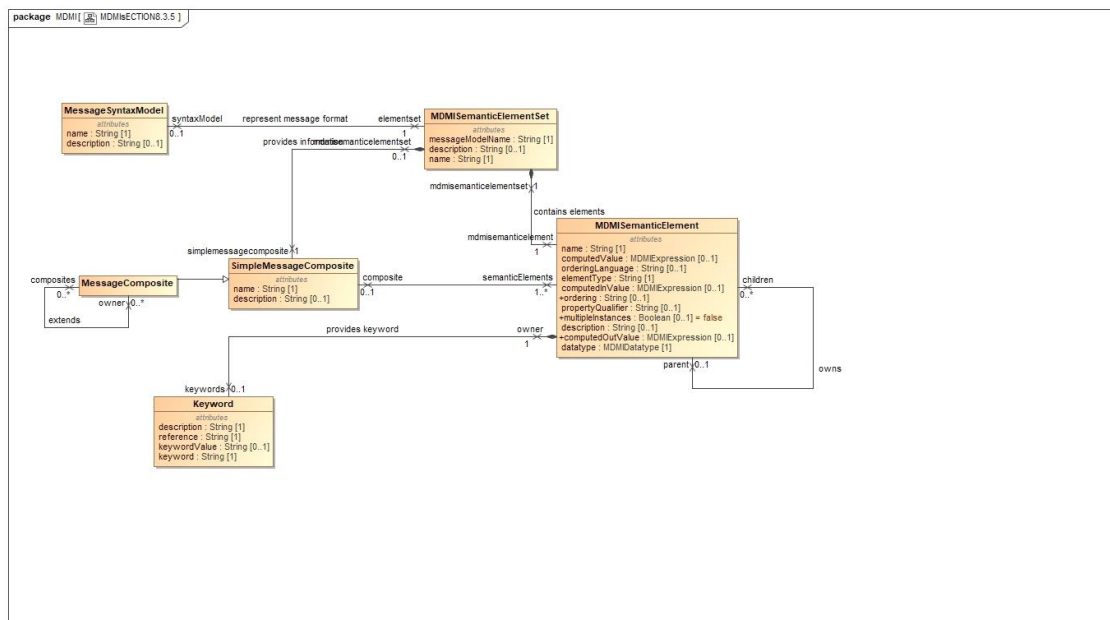


Figure 8.3 – MDMISemanticElementSet and associated classes

8.3.3 MDMISemanticElementSet - Detailed Semantics

MDMISemanticElementSet description:

The MDMISemanticElementSet contains the smallest MDMISemanticElements contained in a message format. The set only holds MDMISemanticElements. All the message-specific syntax of selected elements from a message format has been removed.

MDMISemanticElementSet properties:

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

MDMISemanticElementSet associations:

1. The MDMISemanticElementSet has a one-to-many association by composition to MDMISemanticElements.
2. The MDMISemanticElementSet has a zero-to-many association with SimpleMessageComposites. A SimpleMessageComposite is a convenient mechanism for grouping MDMISemanticElements.
3. The MDMISemanticElementSet has a one-to-one relationship to its parent MessageModel.
4. The MDMISemanticElementSet has a one-to-one relationship to its sibling, the MessageSyntaxModel.

8.3.4 MDMISemanticElement – Detailed Semantics

MDMISemanticElement description:

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

The MDMISemanticElement properties:

1. A “name” property, of type String, contains the name of the MDMISemanticElement.
2. The optional “description” property, of type String, contains a description of the MDMISemanticElement.
3. An “elementType” property, of the enumerated type MessageElementType, can have three values, each of which defines the type of MDMI Semantic Element.
 - NORMAL – A “NORMAL” MDMI Semantic Element is equivalent to the current definition of a MDMISemanticElement contained in a message format, which is to be mapped to a central repository.
 - LOCAL – A “LOCAL MDMI Semantic Element contains some technical information that is needed to correctly map NORMAL MDMI Semantic Elements, e.g., it may contain an index that is used to provide the ordering for a child MDMI Semantic Element that has multiple instances.

MDMI 2.0 Submission

- **COMPUTED** – A “COMPUTED” MDMI Semantic Element is to be mapped to the central repository but contains a value that is not directly contained in a message. Instead, a “COMPUTED” MDMI Semantic Element’s value is computed using a MDMIExpression.
4. A “datatype” property, of type MDMIDatatype, defines the simple or complex datatype of the MDMI 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 MDMISemanticElement.
 6. A “multipleInstances” property, of type Boolean, which if true indicates that instances of this MDMISemanticElement 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 MDMI Semantic Element instances are ordered, if the MDMISemanticElement’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.
 9. A “computedValue” property, of type MDMIexpression, contains an expression that computes the value for the MDMISemanticElement. The expression can refer to the value of other MDMISemanticElements. This property is most often used for MDMISemanticElements of the type LOCAL.
 10. A “computedInValue” property, of type MDMIexpression, contains an expression to compute a value for the MDMISemanticElement when it is a target, based on the values of one or more MDMI Business Elements and MDMISemanticElements. The value when it is a source is directly mapped.
 11. A “computedOutValue” property, of MDMIexpression, contains an expression to computes value for a MDMISemanticElement, when it is a source, based on the values of one or more MDMISemanticElements. The value when it is a target is directly mapped.

The MDMISemanticElement associations:

1. A one-to-many association with any children through a parent association. This allows the MDMISemanticElementSet to include container Semantic Elements, which are identified by “parent.” Explicit container MDMI Semantic Elements allow the hierarchical structure of a message format to be maintained in the MDMISemanticElementSet. In the case where a container MDMISemanticElement 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 MDMISemanticElementRelationship class. The MDMISemanticElementRelationship provide the valid context for each MDMISemanticElement.
3. A one-to-one relationship to a syntax Node. The Node provides the syntax information associated with the MDMISemanticElement.
4. A many-to-one (or -zero) association with a SimpleMessageComposite. SimpleMessageComposites provide a convenient mechanism for grouping MDMISemanticElements.
5. A many-to-one association with its parent MDMISemanticElementSet.
6. A zero-to-many association with the DataRule class, which specifies a set of rules that apply to the datatype of the MDMISemanticElement.
7. A zero-to-many association with a keyword list, which can be used to identify the MDMISemanticElement for searches and which can be associated with a formal ontology.
8. A zero-to-many association with a MDMISemanticElementBusinessRule, which provides for a specific set of rules that should apply to the value of the MDMISemanticElement.

9. A one-to-many association with the ToMDMIBusinessElement class that describes the conversion of the value of the MDMISemanticElement to conform to the reference value of the MDMI Business Element referenced by the MDMIBusinessElementReference class.
10. A one-to-many association with the ToMDMISemanticElement class that describes the conversion of the reference value of the MDMI Business Element referenced by the MDMIBusinessElementReference class to the value of the MDMISemanticElement.

8.3.5 Keyword – Detailed Semantics

Keyword description:

The keyword class contains either a keyword or a keyword/value pair. The set of Keywords can be used to profile a MDMI Semantic Element, to provide a mechanism to search for a MDMI Semantic Element, and to associate a MDMI Semantic Element with an external ontology or taxonomy. See also Section 8.x.x for an alternative approach mechanism for searching.

Keyword properties:

1. The optional “description” property, of type string, describes the Keyword and/or the set of Keyword associated with a MDMI Semantic Element.
2. A “keyword” property, of type String, used to describe or profile a MDMI Semantic Element.
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.

Keyword associations:

1. An optional many-to-one association with the MDMI Semantic Element it is describing.

8.3.6 SimpleMessageComposite – Detailed Semantics

SimpleMessageComposite description:

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

SimpleMessageComposite properties:

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

SimpleMessageComposite generalization:

MessageComposite inherits from SimpleMessageComposite.

SimpleMessageComposite associations:

1. A zero-to-many association with a MDMISemanticElementSet by composition.
2. A (zero or one)-to-many association with MDMI Semantic Elements.

8.3.7 MessageComposite -- Detailed Semantics

MessageComposite description:

MDMI 2.0 Submission

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 MDMI Semantic Elements with a complex object model.

MessageComposite associations:

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 reference a datatype used in the model. These MDMIDatatypes are not considered part of the MDMI standard. The specification does not deal with datatypes directly but some restrictions on MDMIDatatype definitions are necessary for syntactic modeling and to ensure that a runtime engine will do proper transformations. These restrictions include: 1) that the simple datatypes be from a known standard, such as the XML simple datatypes; and 2) that complex datatypes are ultimately composed of simple datatypes and that every simple datatype 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 the simple datatype fieldname.

8.4.2 An example of Complex Datatype

A Semantic Element can be composed of complex datatypes that span a number of fields (or sub-fields) in a message format. Each such field, by itself, does not have a specific meaning in the message but is rather a syntactic artifact that – when combined with other fields – represent a complete datatype. For example, an address is 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 8.4.2. 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 MDMI 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.

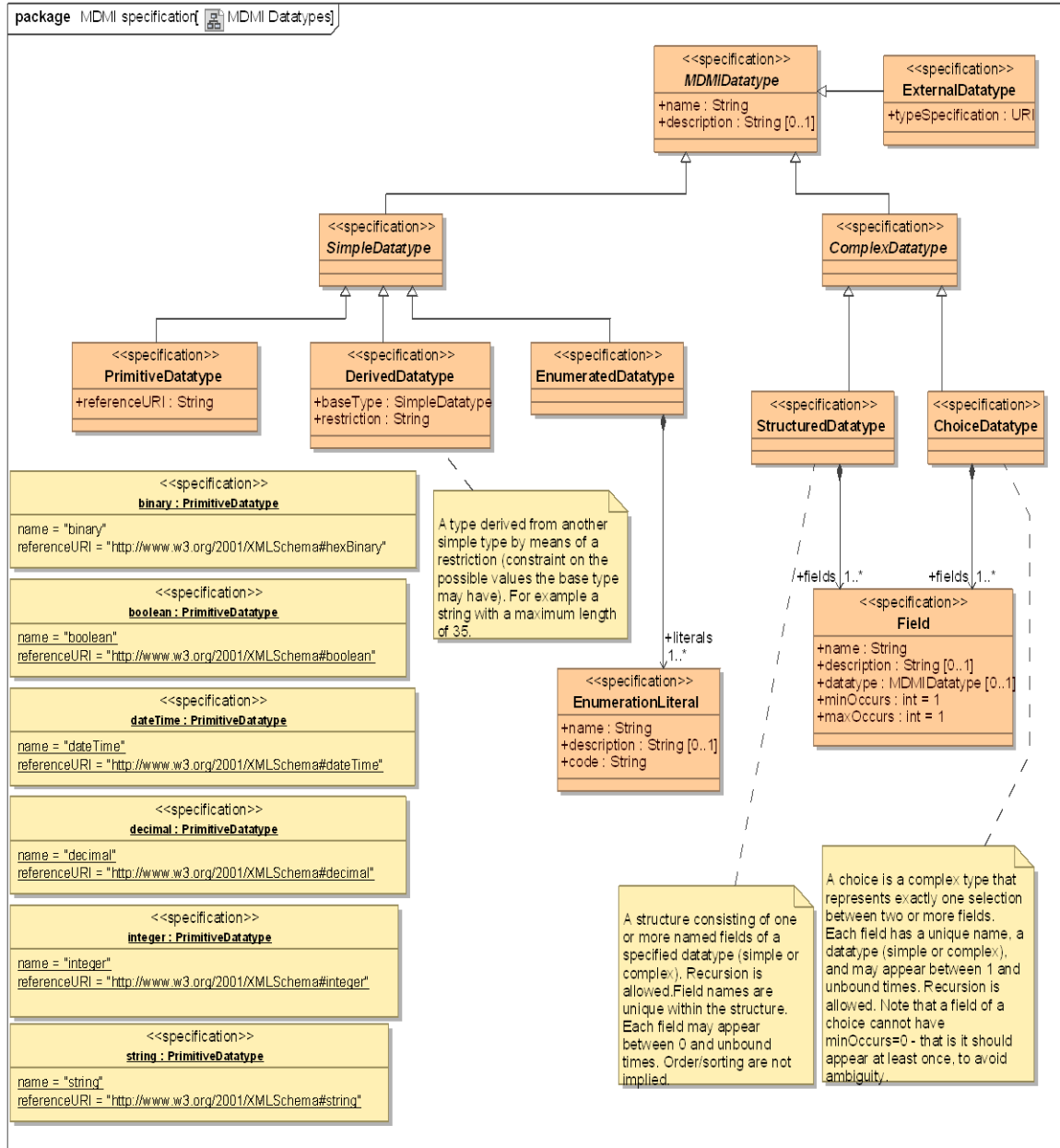


Figure 8.4.2 – Complex Datatype

8.4.3 MDMIDatatype, DataRules – Abstract Syntax

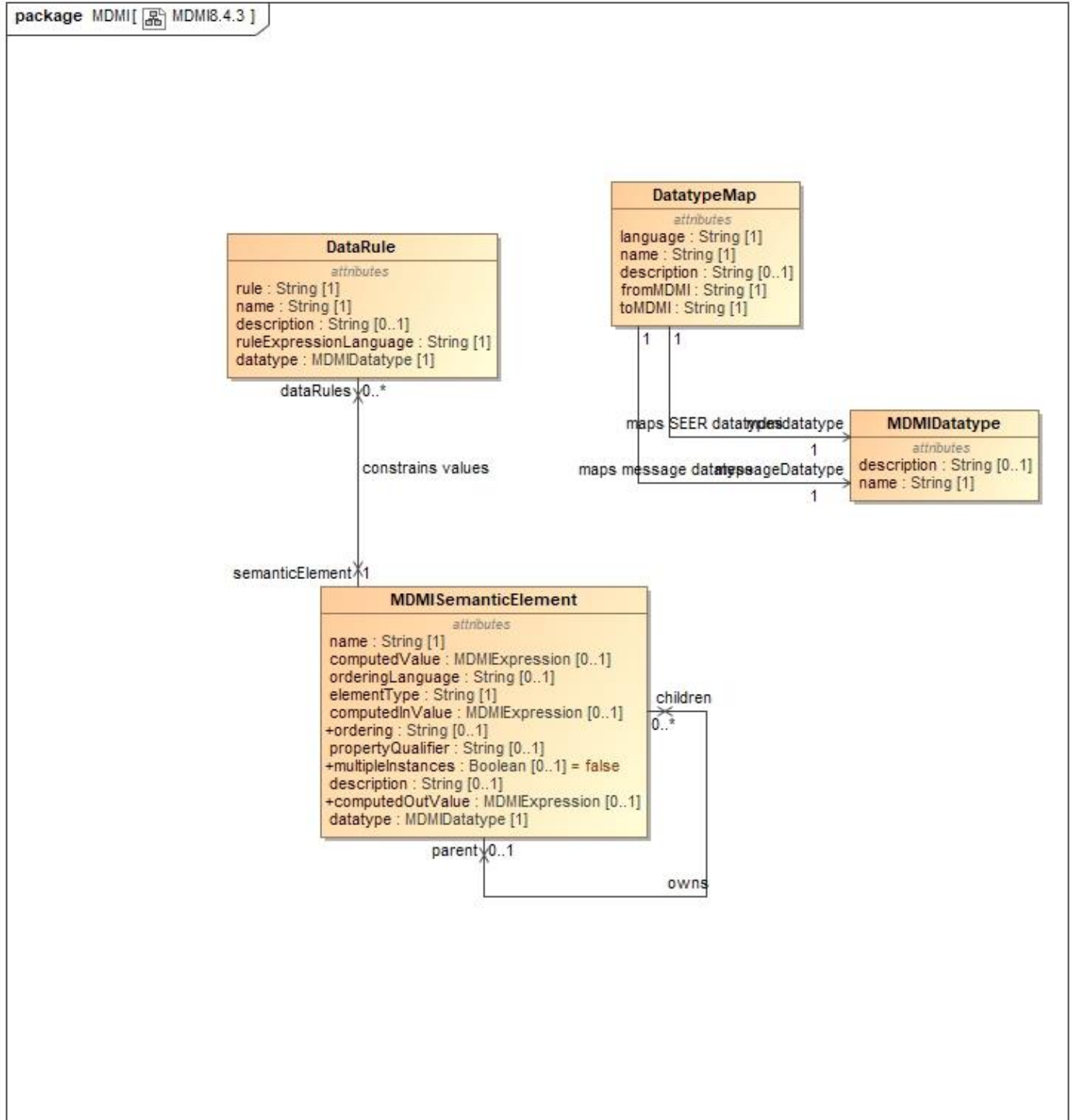


Figure 8.4.2 – MDMIDatatypes, DataRule

8.4.4 MDMIDatatype – Detailed Semantics

MDMIDatatype 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.

MDMIDatatype 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

DataRule 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 known.

DataRules 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 MDMI Semantic Element class.
4. A “ruleExpressionLanguage,” of type String, references the language in which the “rule” property is expressed. The standard does not require any rule language but the language must 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 parsing and runtime processing.

DataRules associations:

1. Zero-to-many DataRules can be associated with a MessageGroup.
2. Zero-to-many DataRules can be associated with a MDMI Semantic Element class.

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

8.5.1 Overview

The classes in this view describe the mapping between a MDMI Semantic Element and an MDMIBusinessElementReference. An MDMIBusinessElementReference class references a MDMI Business Element in a MDMI Domain SEER. No assumption is made about the format of the MDMI Business Element in the MDMI Domain SEER. Because the format of the registry 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 MDMI Business Element can be specified.

Given the MDMIBusinessElementReference, a conversion between it and a MDMI Semantic Element can be made. This conversion may not be symmetric so a mapping must be defined for each direction: MDMI Semantic Element to MDMI Business Element and MDMI Business Element to MDMI Semantic Element. (Mappings for both directions must be defined; one-way mappings are not allowed in the standard.) These mappings are specified in a ToMDMISemanticElement class and a ToMDMIBusinessElement class. Both 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 differences between MDMISemanticElements 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 MDMISemanticElement cannot be mapped it implies that is not in the MDMI Domain SEER and should be added to it. The MDMI 2.0 Standard supports this ability. However, since MDMI Maps are

generally developed independently, a MDMI Domain SEER has added the requirement to eliminate hyponyms and hypernyms in a MDMI Domain SEER. This should reduce the use of the MDMConversionRule Class.

8.5.2 Abstract Syntax

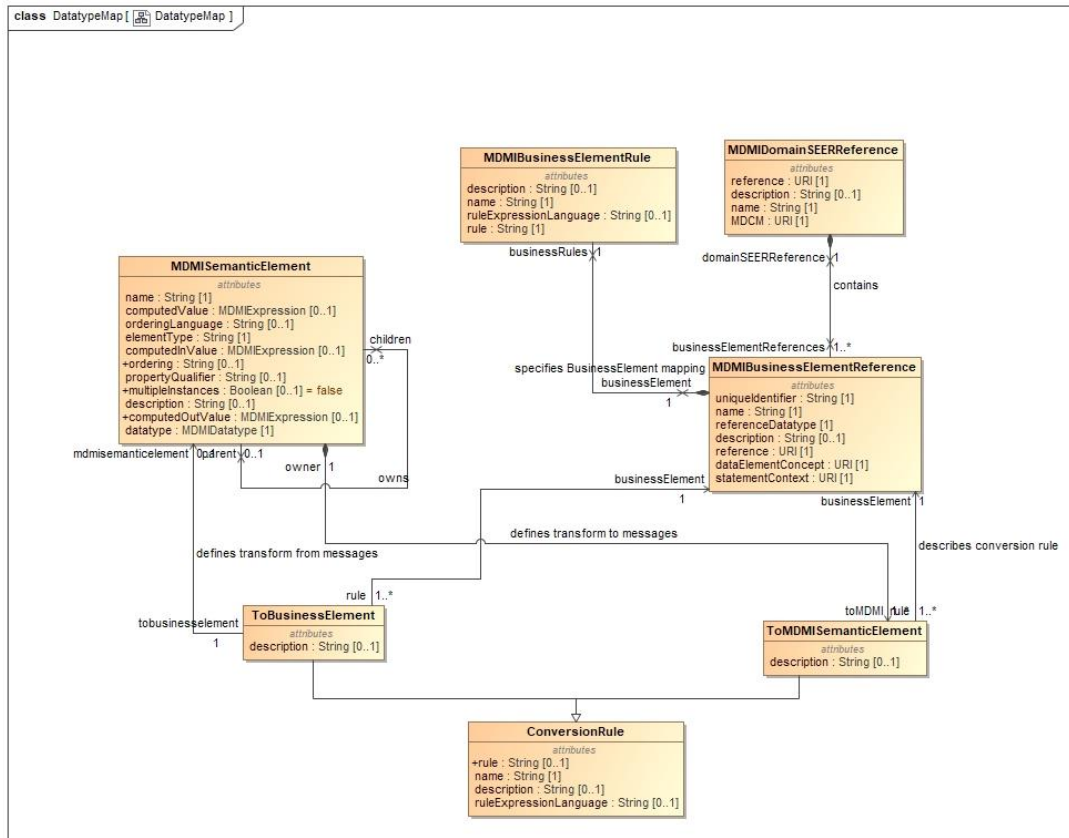


Figure 8.5 – MDMIBusinessElementReference and ConversionRule

8.5.3 MDMIBusinessElementReference – Detailed Semantics

MDMIBusinessElementReference description:

The MDMIBusinessElementReference is a class that references a MDMI Business Element in a MDMI Domain SEER. No assumption is made about the format of the MDMI Business Element in the MDMI Domain SEER. Therefore, the reference can only be informational. However, a function must be available that, given the reference, will return a uniqueIdentifier and a reference MDMIDatatype.

MDMIBusinessElementReference 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 MDMIBusinessElementReference in a SEER. (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 MDMI Business Element

MDMI 2.0 Submission

in the SEER. There must be a function associated with the SEER that provides this identifier. Runtime transformation engines recognize the matching source and target mappings for a MDMI Semantic Element because they will each have the same “uniqueIdentifier.”

5. The “referenceDatatype” property, of type MDMIDatatype, provides a reference datatype for each MDMI Business Element in the SEER. There must be a function associated with the SEER that will deliver the “referenceDatatype.” Maps to/from this reference datatype to the “datatype” in the MDMISemanticElement should be provided as a ConversionRule.
6. The StatementContext property shall be a URI referencing an OWL property in the MDMI MDCM. The StatementContext property the context for the MDMI Business Element in the message format. The statement context must be precise and unambiguous. It shall have no subtypes as subtypes could introduce variation in the meaning. StatementContext may have “qualifiers.” Qualifiers are properties intended to narrow the meaning of a context to make it precise and unambiguous. As a healthcare example, a Medication Activity is a context but it is not precise or unambiguous. It is not known whether the Medication Activity was an administration of a medication or an order for a medication. Where the statement context has qualifiers, the restricted range of that qualifier shall have no subtypes. (See Annex B: Informative:)
7. The DataElementConcept property shall be a URI referencing an OWL property in the MDMI MDCM that identifies an atomic datum, such as a name or measurement value. It is expected that data property concepts will reference datatype properties but MDMI does not make this restriction to allow for representation flexibility and reified values. The referenced property must be precisely and unambiguously defined. It must have a domain that is the same as or a supertype of the StatementContext. It must have no sub-properties. It may have super-properties and such super-properties may be useful in the process of locating MDMI Business Elements. It is frequently useful to define a data property concept as a “path” or “chain” though other properties, such as traversing from the statement context to its author to the author’s legal name. To ensure uniqueness, properties referenced in a chain may not have sub-properties. (See also Annex B.)

MDMIBusinessElementReference associations:

1. MDMIBusinessElementReference has a one-to-many association with the ToMDMISemantic class.
2. MDMIBusinessElementReference has a one-to-many association with the ToMDMIBusinessElement 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 MDMIDomainSEERReference class.

8.5.4 ConversionRule – Detailed Semantics

ConversionRule description:

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

ConversionRule 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. 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.

ConversionRule generalizations:

The abstract `ConversionRule` class is inherited by two classes, the “`ToMDMIBusinessElement`” and the “`ToMDMISemanticElement`.”

8.5.5 ToMDMISemanticElement – Detailed Semantics

ToMDMISemanticElement description:

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

ToMDMISemanticElement properties:

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

ToMDMISemanticElement associations:

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

8.5.6 ToMDMIBusinessElement

ToMDMIBusinessElement description:

The `ToMDMIBusinessElement` associates an `MDMIBusinessElementReference` with a `MDMISemanticElement`, describing the directed conversion rule for converting the value of the `MDMISemanticElement` to the reference value of the referenced `MDMI Business Element`. A `MDMISemanticElement` may be related to more than one `MDMIBusinessElementReference` but will have a separate `ToMDMIBusinessElement` class with individual rules for each relationship.

ToMDMIBusinessElement properties:

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

ToMDMIBusinessElement associations:

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

8.5.7 MDMIBusinessElementRule

MDMIBusinessElementRule description:

Given that the `MDMI` standard allows mapping to any appropriate `MDMI MDCM` then some business rules may have to be specified within a map to make sure that the mapping is correct. Instances of the `MDMIBusinessElementRule` maintain these rules.

MDMIBusinessElementRule 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 `MDMI Semantic`

MDMI 2.0 Submission

Element instance and it should allow external function calls. If this property is not specified the default ruleExpressionLanguage will be used.

MDMIBusinessElementRule associations:

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


8.6 MDMISemanticElementRelationship

8.6.1 Overview

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

An important relationship is the parent-child relationship. For example, the MDMI Semantic Element of MedicationAdministrationActivity is the parent of many MDMI Semantic Element siblings, such as MDMI Business Elements that describe the who, what, where, why, and when data concepts within a MedicationAdministrationActivity. This pattern is found throughout all industries.

8.6.2 Abstract Syntax

class MDMISemanticElementRelationship [ MDMISemanticElementRelationship]

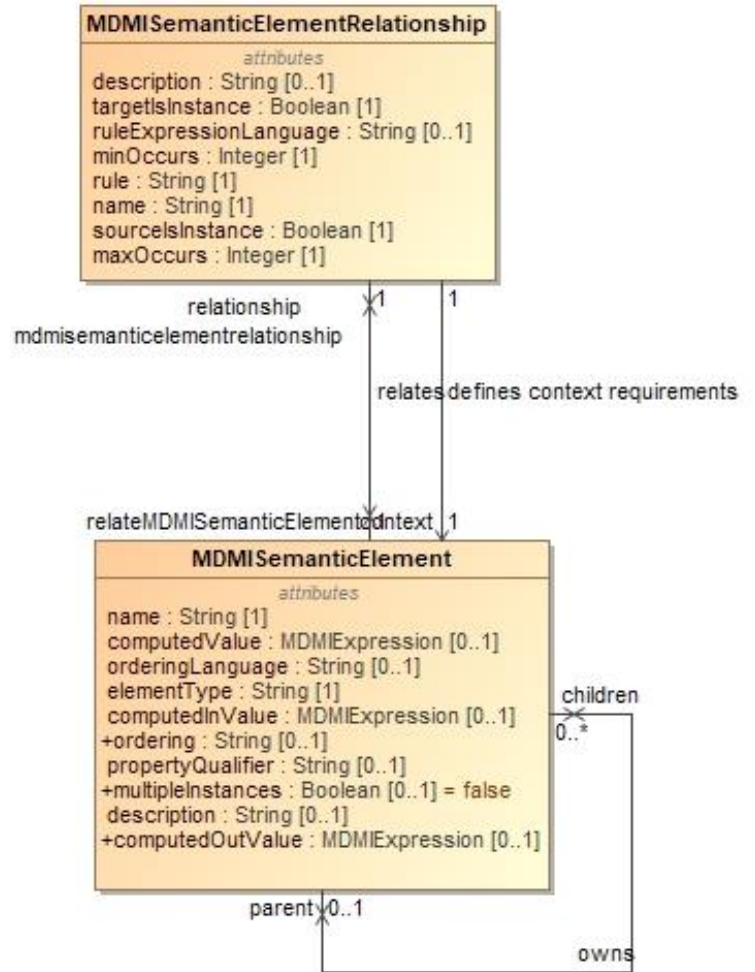


Figure 8.6 – MDMISemanticElementRelationship

8.6.3 MDMISemanticElementRelationship – Detailed Semantics

MDMISemanticElementRelationship description:

The MDMISemanticElementRelationship class is a key artifact in the MDMI standard. It provides all the context and dependency relationships for each MDMISemanticElement.

MDMISemanticElementRelationship makes it possible to extract and insert MDMISemanticElement values in a valid manner.

MDMISemanticElementRelationship 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 MDMI Semantic Element and other MDMI Semantic Elements in the MDMISemanticElementSet.
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 MDMI Semantic Element 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, that 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 MDMI Semantic Element. (The association with the “source” MDMI Semantic Element is labeled “relatedMDMISemanticElement.” The relatedMDMISemanticElement owns the relationship by composition. This source is the MDMISemanticElement whose context is being modeled. When the sourceIsInstance is false, the defined relationship is for the source MDMISemanticElement 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 MDMISemanticElement. (The association with the set of one-to-many “targets” is labeled “context.” Thus, a MDMISemanticElementRelationship describes a relationship between a source and the other MDMISemanticElements, which are then targets.) When the targetIsInstance is false, the defined relationship is for the MDMISemanticElement class as a whole.

MDMISemanticElementRelationship associations:

1. The MDMISemanticElementRelationship has a zero- or many-to-one association with its source MDMISemanticElement.
2. The MDMISemanticElementRelationship has a one to-one association with a target MDMISemanticElement.

8.7 MDMISemanticElementBusinessRule

8.7.1 Overview

The MDMISemanticElementBusinessRule class contains a rule that is to be applied to a specific MDMISemanticElement in the context of the MessageModel that contains the MDMI Semantic Element.

8.7.2 Abstract Syntax

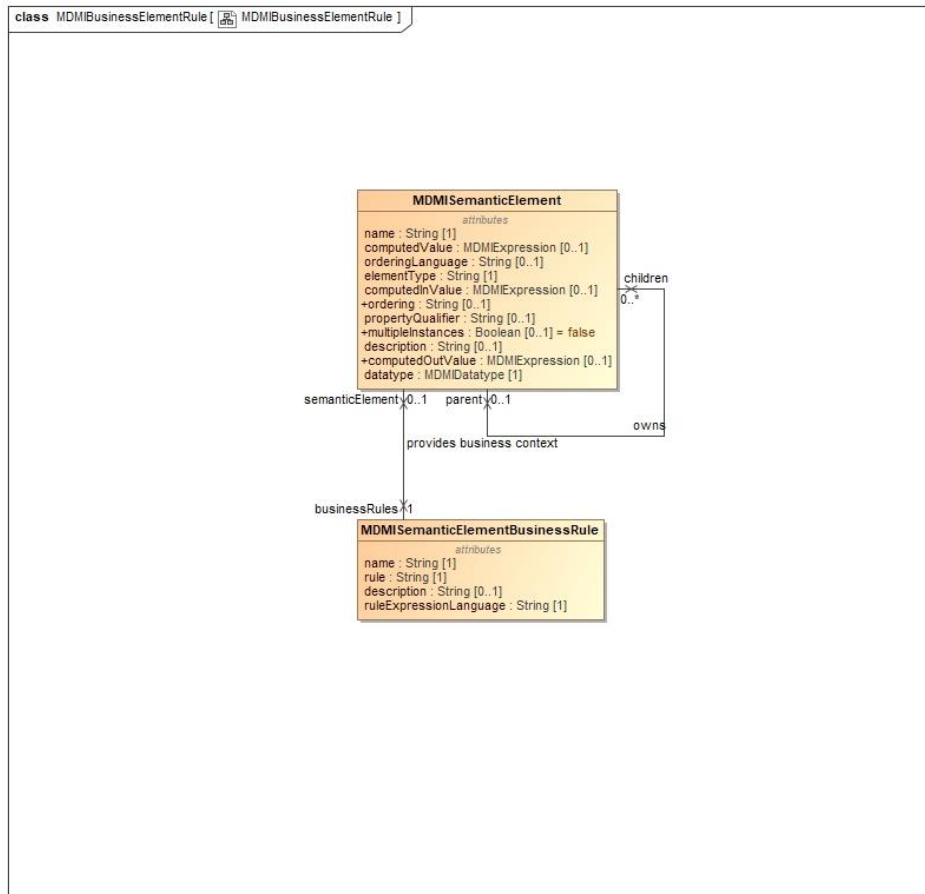


Figure 8.7 – MDMISemanticElementBusiness Rule

8.7.3. MDMISemanticElementBusinessRule – Detailed Semantics

MDMISemanticElementBusinessRule description:

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

MDMISemanticElementBusinessRule 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.

MDMISemanticElementBusinessRule associations:

Annex A - List of Acronyms

Abbreviation	Notes
ANF	Analysis Normal Form
CDA	Clinical Document Architecture http://www.hl7.org/
HL7	Health Level 7 http://www.hl7.org/
HL7 V2	Health Level 7 messaging standard http://www.hl7.org/
FHIR®	Fast Healthcare Interoperability Resources https://www.hl7.org/fhir/
FIX	Financial Information eXchange http://www.fixtrading.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
Swift	Society for Worldwide Interbank Financial Telecommunication supplies secure messaging services. http://www.swift.com
Twist	Transaction Workflow Innovation Standards Team www.twiststandards.org

ANNEX B - Informative Healthcare Examples

The following describes how the MDMI Healthcare Concept Model was developed and used to assign StatementContext and DataElementConcept properties for MDMI Business Elements, a diagram of the informative model, and examples developing the StatementContext and DataElementConcept properties. The MDMI Healthcare Concept Model is the domain-specific MDMI Domain Concept Model.

Developing the MDMI Healthcare Concept Model

The process used to develop the MDMI Healthcare Concept Model was:

1. Define the scope of the MDMI Business Element set in the MDMI Healthcare SEER.
2. Select the MDMI Reference Ontology(-ies) and terminologies (“Reference Models”) for the MDMI Healthcare Concept Model .
3. Develop an initial MDMI Healthcare Concept Model.
4. Refactor the model based on input from healthcare industry subject matter experts and experience in creating StatementContext and DataElementConcept properties for MDMI Healthcare Business Elements.

Step 1: Scope of MDMI Business Elements for informative model

The present domain of interest for the MDMI Healthcare Domain is any business concepts exchanged among the industry-standard message standards of FHIR R4, CCD 2.1, and HL7V2.x. For this example, the scope has been further limited to MDMI Business Elements are those MDMI Business Elements that have the StatementContext property of Clinical Statement.

Step 2: Select the MDMI Reference Model

There was no single ontology or terminology identified that could be used as a MDMI Reference Model.. Multiple industry ontologies and terminologies are required. The informative section has chosen the HL7 ANF standard as the primary Reference Model for the following reasons:

- It has a well-defined definition for a Clinical Statement that is fundamental for specializing the StatementContext property.
- It provides a broad range of properties and classes for specializing the DataElementConcept property.
- It provides a broad foundation that can and has references to industry standard ontologies and terminologies. The ANF standard and related technologies has the mission to harmonize the ontologies and terminologies used in clinical statements. This includes SNOMED-CT from the International Health Terminology Standards Development Organization (IHTSDO) containing more than 300,000 concepts and 1.6 million relationships between the concepts; LOINC from the Regenstrief Institute with more than 71,000 terms; and RxNorm from the U.S. National Library of Medicine.
- Other healthcare ontologies and terminologies were used to complement ANF as required. The primary sources were terminologies produced by the HL7.

Step 3: Assigning the MDMI StatementContext property and the DataElementConcept property in the MDMIBusinessElementReference

This was achieved manually by members of the proposal team. The basic processes were to:

1. Select the existing MDMI Business Elements in the MDMI Healthcare SEER based on the scoping criteria.

2. Using the MDMI Healthcare Concept Model, assign a StatementContext property to the MDMIBusinessElementReference.
3. Using the MDMI Healthcare Concept Model , assign a DataElementConcept property to the MDMIBusinessElementReference.

Step 4: Running the MDMI Acceptance Test

It is anticipated that the MDMI Acceptance Test will be used to add new MDMI Business Elements to the MDMI Healthcare SEER. There are two components of the MDMI Acceptance Test: the Uniqueness Test and the Precision Test.

Step 4.1 – The Uniqueness Test

The Uniqueness Test ensures there are no MDMI Business Elements are synonyms in a MDMI Domain SEER. If two different MDMI Business Elements have the same StatementContext property and DataElementConcept property, these MDMI Business Elements are synonyms. If the MDMI Business Elements were determined to be synonyms, only one of the MDMI Business Elements was included in the MDMI Healthcare SEER.

Acceptance Criteria for Uniqueness Test: If the new MDMI Business Element does not have any other MDMI Business Elements with the same StatementContext property and DataElementConcept property, the MDMI Business Element has passed the Uniqueness Test.

If the new MDMI Business Element did not pass the Uniqueness Test, the following actions was taken.

- Those MDMI Business Elements that were determined to be synonyms were discarded.
- If it determined the duplicate MDMI Business Element were not synonyms but have duplicate StatementContext and DataElementConcept properties, then:
 - if it were determined that the StatementContext and DataElementConcept properties were not specified correctly, they were re-factored and the more accurate representations were assigned for these properties.
 - if it were determined that the StatementContext and DataElementConcept properties had been specified correctly, the MDMI Healthcare Concept Model was reviewed to determine if it needed to be refactored. If this were the case, the MDCM was refactored and more accurate StatementContext and DataElementConcept properties are assigned for the MDMI Business Element based on the refactored the MDMI HEALTHCARE CONCEPT MODEL. Also reviewed were the existing MDMI Business Elements to determine if the refactored MDMI HEALTHCARE CONCEPT MODEL may have impacted any other MDMI Business Element. If this were the case, new StatementContext properties and DataElementConcept properties were assigned.
 - The Uniqueness Test was repeated for entire scope of MDMI Business Elements repeated until all MDMI Business Elements passed.

Step 4.2 – The Precision Test

The Precision Test is to ensure that the MDMI Business Elements are not a hyponym or hyponym of another MDMI Business Element. The practical meaning of this is that different values in the MDMIBusinessElementReference in the StatementContext property and the DataElementConcept cannot be used to accurately describe the same MDMI Business Element. As an example, having MDMI Business Elements of PatientPhone, PatientHomePhone, and PatientCellPhone could lead to loss of information in a transformation.

Acceptance Criteria for Precision Test: The StatementContext property and the DataElementConcept properties are compared for all MDMI Business Elements. If the condition was found that there was a set of MDMI Business Elements that had a hypernym/hyponym relationship, the StatementContext

MDMI 2.0 Submission

property and DataElementConcept property were refactored until the MDMI Business Element passed the Precision Test.

The Precision Test was repeated for entire scope of MDMI Business Elements until all MDMI Business Elements passed.

Step 4.3 – Final Test

A final test is to repeat for all the MDMI Business Elements to ensure they passed the Uniqueness Test and the Precision Test.

MDMI Healthcare Concept Model

The MDMI Healthcare Concept Model is depicted below. This model was developed based on the above process, although the Acceptance Test was executed on a very small subset of MDMI Business Elements. The model can be found at <https://www.omg.org/spec/MDMI/2/health/20-02-04>

The model has been segmented into two different diagrams below. The first diagram is a generic model intended to be applicable across different industry domains.

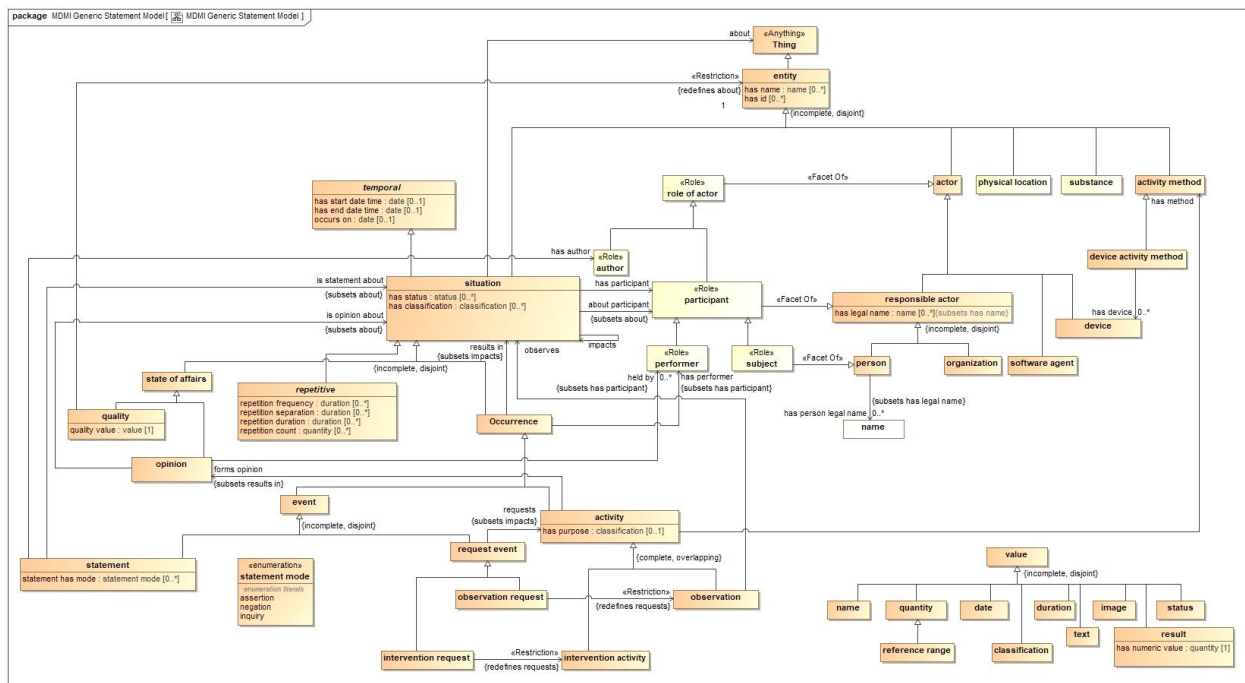


Figure B-1.
Generic component of the MDMI Healthcare Concept Model

The second diagram contains healthcare specific concepts that are required to precisely and accurately describe information exchange in the healthcare domain. In this diagram, those classes in a pale shading are also represented in the generic model, Figure B-1.

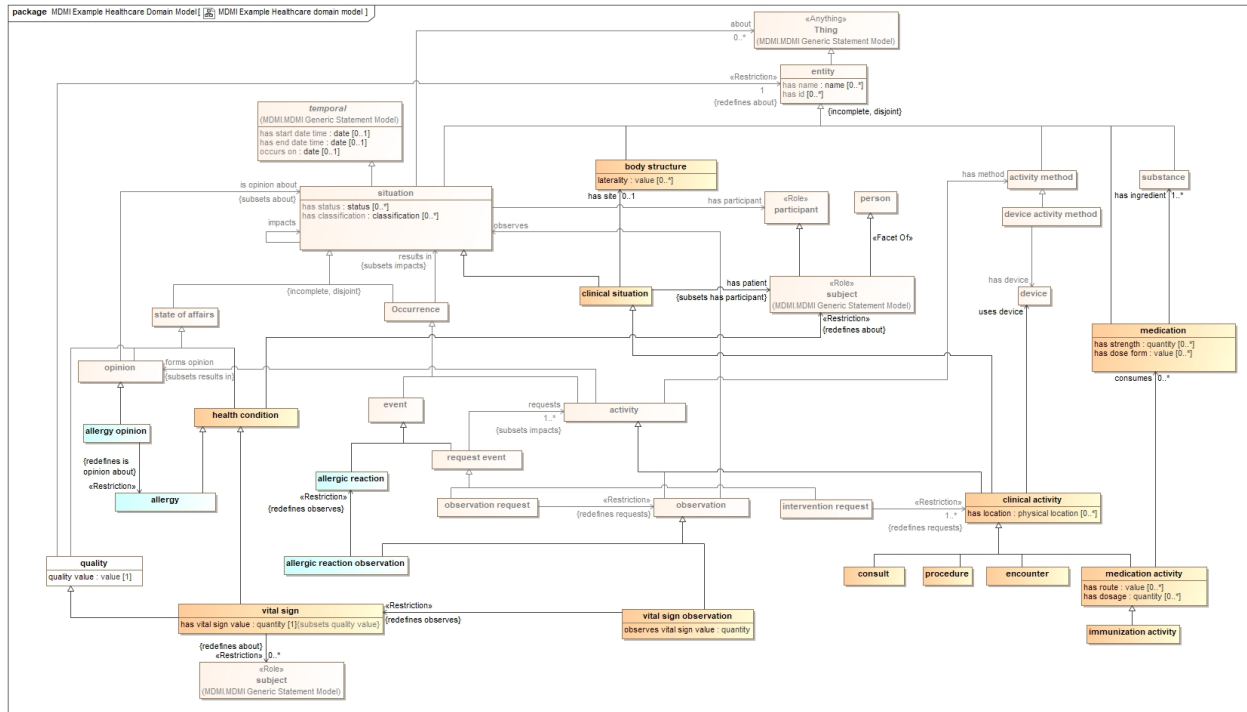


Figure B-2.
Healthcare specific component of the MDMI Healthcare Concept Model

Example of the StatementContext and DataElementConcept properties for MDMI Business Elements

A healthcare professional takes the systolic blood pressure for the patient. Although there are many more MDMI Business Elements necessary to completely and precisely describe this Situation, the MDMI Business Elements used for this example is VitalSignObservationActionFocus.

The StatementContext property for the MDMI Business Element.

Graph description: The StatementContext property for the MDMI Business Element of VitalSignObservationActionFocus: “ is a Statement about a Situation that is an Occurrence of an Event which was the Activity of an Observation about a Health Condition that is Vital Sign.”

StatementContext property: VitalSignObservation

The DataElementConcept property for the MDMI Business Element. (The actual value is Systolic Blood Pressure)

Graph description: DataElementConcept property for the MDMI Business Element of VitalSignObservationActionFocus: has a Value that is ActionFocus.

DataElementConcept property: ActionFocus