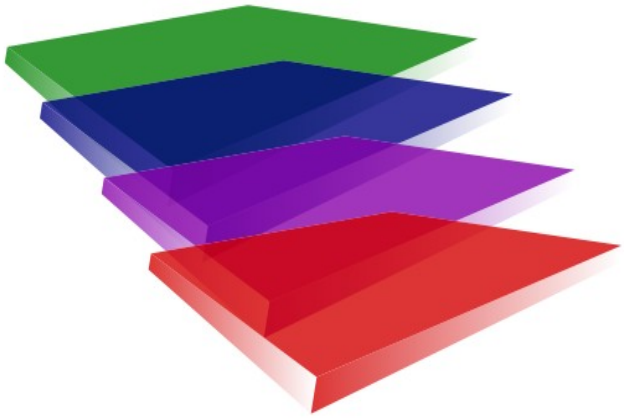
Multi-Level Health Information modeling



MLHIM

User & Reference Manual

Release 2.4.5

DRAFT!

Copyright, 2009-2014

Timothy W. Cook & Contributors

The goal of MLHIM is to be Minimalistic, Sustainable, Implementable AND Interoperable.

Distribution is permitted under the terms of the Attribution-ShareAlike 4.0 International license.

Table of Contents

Front Matter....5
Acknowledgements....5
Using the MLHIM Reference Manual....6
Scope....6
Release Information....6
Error Reporting....6
Pronunciation....7
Purpose & Scope....7
Conformance....7
Compliance....8
Availability....8
Governance....9
Introduction....10
MLHIM as a Component of an Enterprise Architecture....12
[this section is currently incomplete]....13
Data Description Levels....14
The MLHIM Eco-System....16
A Valid CCD Must:....17
A Valid CCD Must Not:....19
Modeling Concepts....20
Approach....20
Constraint Definitions....21
CCD Identification....22
CCD Versioning....22
Pluggable complexTypes (PcTs)....23
Implementations....23
Best Practices....23
The Reference Model....24
Assumed Types....24
Technical Documentation....30
2.4.5 Reference Model....30

UML....30
complexTypes....31
DataValue Group....31
DvAnyType....31
DvBooleanType....31
DvURIType....32
DvStringType....32
DvCodedStringType....33
DvIdentifierType....33
DvEncapsulatedType....34
DvParsableType....34
DvMediaType....35
DvOrderedType....35
DvOrdinalType....36
DvQuantifiedType....37
DvCountType....37
DvQuantityType....38
DvRatioType....38
DvTemporalType....39
DvIntervalType....40
interval-type....41
ReferenceRangeType....41
Common Group....42
AuditType....42
PartyType....42
AttestationType....43
ParticipationType....43
ExceptionalValueType....45
NIType....45
MSKType....45
INVType....46
DERType....46

UNCType....46
OTHType....46
NINFType....47
PINFType....47
UNKType....47
ASKRType....47
NASKType....48
QSType....48
TRCType....48
ASKUType....48
NAVType....49
NAType....49
Items Group....50
ItemType....50
ClusterType....50
DvAdapterType....50
Entry Group....51
EntryType....51
CareEntryType....51
AdminEntryType....52
DemographicEntryType....52
Constraint Group....52
CCDType....52
Example CCD....53
MLHIM Data Analysis....53
Concept Constraint Definition Generator (CCD-Gen)....54
Creating and Managing CCDs....54
Data Instances....54
HTML Forms....55
R Code....55
Publications & Social Media....57
Peer Reviewed....57

Specifications Wiki....57
Code Examples....57
SlideShare....57
YouTube....57
FaceBook....57
Google Plus....57
Twitter....58
Glossary....59

Front Matter

Acknowledgements

This work has received financial and in-kind support from the following persons and organizations:

- National Institute of Science and Technology on Medicine Assisted by Scientific Computing (INCT-MACC), coordinated by the National Laboratory of Scientific Computing (macc.lncc.br)
- Multilevel Healthcare Information Modeling Technological Development Unit, Member of the Emergent Group for Research and Innovation in Healthcare Information Technology, coordinated by Prof. Luciana Tricai Cavalini, MD, PhD (lutricav@mlhim.org)
- Timothy W. Cook, Independent Consultant
- Roger Erens, Independent Consultant

Using the MLHIM Reference Manual

This section describes typographical conventions and other information to help you get the most from this document.

The intended audience for this manual includes; software developers, systems analysts and knowledge modelers in the healthcare domain. It is assumed that the reader has knowledge of object-oriented notation, concepts and software construction practices.

In the PDF release of these specifications cross reference links are denoted by a number inside square brackets i.e. [5].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document, are to be interpreted as described in RFC 2119.

Scope

This document focuses on common data elements and their value sets for the description of data. Although use cases are drawn from the healthcare domain, this document will focus on requirements that are broadly applicable.

Release Information

The official published version is in PDF format in the English language. The ODT version is always considered a work in progress. Each version is labeled with the release number of the reference model and its implementation in XML Schema 1.1.

Previous versions of the PDF release had a version number that is the date of release followed by the language code and locality code. As an example, a release in English on January 1, 2011 has the filename: mlhim-ref-man-2011-01-01-en-US.pdf

Error Reporting

Please report all errors in documentation and/or in the specifications of the information model as bug reports at the GitHub development site. It is easy to do and a great way to give back. See: MLHIM Issues

Pronunciation

MLHIM is pronounced 'muh-leem Click Hear How It Sounds for when it is used in spoken English language such as presentations or general discussions.

Purpose & Scope

Keep everything as simple as possible; but no simpler. – Albert Einstein

The purpose of the MLHIM project is to provide a free and openly available specification and implementation of an interoperability solution for healthcare information exchange.

The MLHIM specifications are designed to provide semantic interoperability that is fully independent of any implementation specific contexts. Therefore workflow, security, user access, data persistence, etc. are all outside the scope of MLHIM.

The MLHIM site on GitHub[1] contains a growing number of demonstration projects and tools that research how this approach and limited scope enables interoperability across many contexts. MLHIM based data can be exchanged via any transport method. Including existing HL7 v.2 and v.3 exchange systems to enhance the semantic interoperability of existing HL7 implementations.

Conformance

Conformance to these specifications are represented in a Language Implementation Specification (LIS). A LIS is formal document detailing the mappings and conventions used in relation to these specifications.

A LIS is in direct conformance to these specifications when:

1. All datatypes are defined and mapped.
2. The value spaces of the healthcare datatypes used by the entity to be identical to the value spaces specified herein.
3. To the extent that the entity provides operations other than movement or translation of values, define operations on the healthcare datatypes which can be derived from, or are otherwise consistent with the characterizing operations specified herein.

1 https://github.com/mlhim

Compliance

These specifications:

- Are in indirect conformance with ISO/DIS 21090/2008.
- Are in compliance with applicable sections of ISO 18308/2008.
- Are in compliance with applicable sections of ISO/TR 20514:2005.
- Are in compliance with applicable sections of ISO 13606-1:2007.
- Are in conformance with W3C XML Schema Definition Language (XSD) 1.1

Availability

The MLHIM specifications, reference implementation and tools are available from GitHub:
https://github.com/mlhim/specs
Final versioned releases are available, packaged as .ZIP files from Launchpad:
http://launchpad.net/mlhim-specs

Governance

The MLHIM project was originally started by four individuals with a desire to build an openly available approach to true syntactic interoperability as well as *computable semantic interoperability* for global healthcare IT. The founding agreement signed by these four guarantees that the MLHIM specifications documents and the reference implementation will be published under open content and open source licenses.

The MLHIM project is a meritocracy. *Anyone may participate* in the community which uses the Google Plus social network community for discussions. It is a private community only for the sake of SPAM reduction. No requests to join will be denied. However, SPAMMING the community with off-topic content will get you banned immediately, without notice.

The complete governance guidelines and Contributor License Agreement (CLA) can be found in the repository in the document MLHIM_governance_model.pdf. They are also available under the Documents section of the MLHIM web site.

Introduction

The Multi-Level Health Information Modeling (MLHIM) specifications are partially derived from various ISO Healthcare Information Standards (See the 'Compliance Section') and the openEHR 1.0.2 specifications and the intent is that MLHIM 1.x be technologically inter-operable with openEHR.

MLHIM 2.x (this document and related artifacts) introduces further innovation through the use of XML technologies and reducing complexity without sacrificing interoperability as well as improved modeling tools and as application development platforms. These specifications can be implemented in any structured language. While a certain level of knowledge is assumed, the primary goal of these specifications is to make them 'implementable' by the widest possible number of people. The primary motivator for these specifications is the complexity involved in the recording of the temporal-spatial-ontological relationships in healthcare information while maintaining the original semantics across all applications; for all time.

We invite you to join us in this effort to maintain the specifications and build great, translatable healthcare tools and applications for global use.

International input is encouraged in order for the MLHIM specifications to allow for true inter-operability, available to everyone in all languages and most of all, implementable by mere mortals.

Actual implementation in languages other than XML Schema and related XML technologies, the packages/classes should be implemented per the standard language naming format. A Language Implementation Specification (LIS) should be created for each language. For example MLHIM-Python-LIS.odt or MLHIM-Java-LIS.odt.

MLHIM intentionally does not specify full behavior within a class. Only the data and constraints are specified. Behavior may differ between various applications and should not be specified at the information model level. The goal is to provide a system that can capture and share the semantics and structure of information in the context in which it is captured.

The generic class names in the specification documents are in CamelCase type. since this is most typical of implementation usage.

Only the reference model is implemented in software where needed. In many implementations, the application can use XML Tools to validate against the CCD and reference model using a NoSQL storage. The domain knowledge models are implemented in the XML Schema language and they represent constraints on the reference model implementation that is also implemented in XML Schema.

The domain knowledge models are called ***Concept Constraint Definitions*** and the acronyms CCD and CCDs are used throughout MLHIM documents to mean these XML Schema files. This means that, since CCDs form a model that allows the creation of data instances from and according to a specific CCD, it is ensured that the data instances will be valid, in perpetuity.

However, any data instance should be able to be imported into any MLHIM based application since the root data model, for any application is the MLHIM reference model. But, the full semantics of that data will not be known unless and until the defining CCD is available to the receiving application. The CCD represents the structural syntax of a healthcare[2] concept using XML Schema constraints and contains the semantics defined by the modeler in the form of RDF or other XML based syntaxes within documentation and annotation segments of the CCD. This enables applications to parse the CCD and publish or compute using the semantics as needed on an application by application basis.

The above paragraph describes the foundation of ***computable semantic interoperability*** in MLHIM implementations. You must understand this and the implications it carries to be successful with implementing and creating the full value of MLHIM based applications. See the Constraint section for further discussion of Concept Constraint Definitions (CCDs).

2 Healthcare concepts include: (a) clinical concepts adopted in medicine for diagnostic and therapeutic decision making; (b) analogous concepts used for all healthcare professionals (e.g. nursing, dentistry, psychology, pharmacy); (c) public health concepts (e.g. epidemiology, social medicine, biostatistics); (d) demographic and (e) administrative concepts that concern healthcare

MLHIM as a Component of an Enterprise Architecture

MLHIM is a core foundational part of any enterprise architecture that strives to attain computable semantic interoperability. In this chapter we will describe how the MLHIM eco-system fits into a typical healthcare IT architecture. Keeping in mind that MLHIM is also applicable to any size application that needs to be semantically interoperable.

The Open Group Architecture Framework (TOGAF®) is a framework for enterprise architecture which provides an approach for designing, planning, implementing, and governing an enterprise information technology architecture. In this document it provides us with a formal method to communicate with implementers and decision makers on integrating MLHIM into the enterprise.

We will focus on Part V (chapter 39) and Part VI (chapter 44) of the TOGAF specification; *Enterprise Continuum* and *Integrated Information Infrastructure Reference Model.* These chapters deals with the view of the enterprise repository as part of the enterprise and the information technology industry at large as well as developing a sound foundation for boundaryless inforamtion flow.

Referring to Figure 39-1 and its description, the MLHIM reference model and its implementation fits into the category *Architecture Continuum.* In the details description and depicted graphically in Figure 39-2 are four conceptual architectures. Their relationship to MLHIM is shown in Table1. Keeping in mind that the granularity of MLHIM is much finer than these abstract architectures. However, this does serve as a good starting point.

TOGAF®	MLHIM
Foundation Architectures	The MLHIM specifications and reference implementation represent a foundation model on which to build more specific models. From a broader perspective, the XML family of technologies provides the foundation for ubiquity.
Common System Architectures	The MLHIM CCDs[] are composed of multiple, reusable (pluggable) complexType[] restrictions of the reference model. These PcTs can be reused in many CCDs and can represent a common group of components. Again from the broader perspective the use of XML Schema 1.1 as an easily transported model is of primary importance as a common architecture.
Industry Architectures	CCDs that have broad applicability in healthcare across many types of applications fit into the architecture. These CCDs may be openly licensed and shared globally. One might consider the availability of tools for XML in this category as well as tools that might be used specifically for creating MLHIM knowledge artifacts (PcTs and CCDs).
Organization-Specific Architectures	CCDs that are used in applications within one organization are in this category.

Table 1: MLHIM in the Architecture Continuum

The reader should consult the details of the TOGAF® documentation as well as the remainder of this document, in order to understand the full implications of implementing MLHIM into large organizations.

In chapter 44 of the TOGAF® specifications we being to focus more towards information interoperability of applications irregardless of their platform and location. Here we can discover the importance of ubiquitous technology in providing interoperability.

[this section is currently incomplete]

Data Description Levels

With the growing interest in *Big Data* analytics, various efforts are ongoing to improve the description of data and datasets. The various domains are attempting to use commonly developed vocabularies, such as the Dublin Core Metadata Initiative (DCMI[3]) terms, the Data Catalog Vocabulary (DCAT[4]), Schema.org[5] and others.

The popular approach is to use the vocabularies to directly annotate the data. We call this the *direct markup* approach. While this approach will work, to some limited extent. There are several problems with this method. The glaringly obvious one is that, more often than not, high quality, precise metadata is often much larger than the actual data being represented. Every instance of data and every copy of the dataset, must carry all of its own metadata. While storage size, memory size, processing speed and network bandwidth have improved immensely over the past decade. They are not infinite and every byte still counts; affecting overall performance in an inverse relationship.

In conservative testing with MLHIM, we can see that the syntactic and semantic metadata for a data instance is typically about three times the size of the data instance itself. So including metadata with the data means a small 16kb data file is now 64kb. Not too bad when you look at it on that scale. However, the growth is linear with this direct data markup approach.

Let us say you record a time-series from some device and your data is 10MB. Now, for that one instance if it is marked up individually, the size blooms to 40MB. Even with today's technology, this is a significant payload to transfer.

Estimates are[6] that every day we create 2.5 quintillion (10^{18}) bytes of data. That linear expansion that resulted in a growth of 48kb is now a growth of 7.5 quintillion (10^{18}) bytes; ***every day***.

Irregardless of any sustainability estimates. Is it even a smart thing to do?

When the data instances are marked up with semantics and they are being exchanged between systems (as medical information should be), There is no single point of reference to insure that the syntax or semantics of the information wasn't modified along the way.

The MLHIM approach to the *computable semantic interoperability* problem does not have these issues. The metadata is developed by the modeler and it is static. It can then be referred to by any required number of data instances. The multi-level modeling approach to development is what enables this level of efficiency in data management. MLHIM uses the ubiquitous XML technology stack to accomplish this.

Other multi-level modeling approaches may use a domain specific language that is not in common use and does not have tools widely available for management and analysis of the models and data.

3 http://dublincore.org/
4 http://www.w3.org/TR/vocab-dcat/
5 http://schema.org/
6 http://www.storagenewsletter.com/rubriques/market-reportsresearch/ibm-cmo-study/

As stated earlier, the growth in size of the data is only one issue with the direct markup approach. An additional concern is the specific file format used for distribution. In the direct markup approach there may be differences in semantics[7] or in the ability to even markup the data at all, using various syntaxes. In MLHIM this is solved because there are very well known and proven approaches for transforming XML to and from other syntaxes. Because we are only transforming the data and not the metadata, it cannot be corrupted or misrepresented.

We have provided open source examples of this transformation process, specifically to and from JSON without any loss of semantics or the ability to validate the data against the schema (CCD). See the MLHIM GitHub site at https://github.com/mlhim/mxic for further details.

One last comment on the issues with the direct markup approach is that is not robust enough for mission-critical data management; certainly not for your clinical healthcare data. This issue is widely recognized and is being addressed by W3C[8]. However, we know from previous experience that the W3C process is a slow one.

In a few years, the may be widespread adoption and tools for validation of RDF and or OWL. At that time it will be easy enough to migrate to MLHIM 3.x using that approach. But we need solutions today and MLHIM offers that solution now; with XML technologies.

7 http://goo.gl/oSTC1g (W3C HCLS Dataset draft specs.)
8 http://www.w3.org/2012/12/rdf-val/report

The MLHIM Eco-System

It is important here to describe all of the components of the MLHIM conceptual eco-system in order for the reader to appreciate the scope of MLHIM and the importance of the governance policies.

At the base of the MLHIM eco-system is the Reference Model (RM). Though the reference implementation is in XML Schema format, in real world applications a chosen object oriented language will likely be used for implementations. Often, tools are available to automatically generate the reference model classes from the XML Schema. This is the basis for larger MLHIM compliant applications. We will later cover implementation options for smaller applications such as mHealth (apps for smartphones and tablets, as well as purpose specific devices such as a home blood pressure monitor).

The next level of the MLHIM hierarchy is the Concept Constraint Definition (CCD). The CCD is a set of constraints against the RM that narrow the valid data options to a point where they can represent a specific healthcare concept. The CCD is essentially an XML Schema that uses the RM complex types as base types. This is conceptually equivalent to inheritance in object oriented applications, represented in XML Schema.

Since a CCD (by definition) can only narrow the constraints of the RM, then any data instance that is compliant with a CCD is also compliant in any software application that implements the RM or is designed to validate against the RM. Even if the CCD is not available, an application can know how to display and even analyze certain information. For example, if a receiving application does not have a CCD for a given data instance it will be able to discern the CCD ID and RM version from the element name and attributes of the root element. It may or may not be able to retrieve the CCD from the xsi:schemaLocation attribute. If not, it will still be able to determine, based on the reference model version, information about the data by using the names of elements nested within an element with the prefix 'ccd:'. Because these element names are unique to certain RM complexTypes. IF there is a <magnitude> element and a DVCount-units element, that fragment is a DvCountType.

This is not to imply that all CCDs must be publicly available. It is possible to maintain a set of CCDs within a certain political jurisdiction or within a certain professional sector. How and where these CCDs are maintained are outside the scope of these specifications. Developers proficient in XML technologies will understand how this fits into their application environment.

This is now the point where the MLHIM eco-system is in contrast to the top-down approach used by other multi-level modeling specifications. In the real world; we know that there can never be complete consensus across the healthcare spectrum of domains, cultures and languages; concerning the details of a specific concept. Therefore the concept of a "maximal data model", though idealistically valid, is realistically unattainable. Participation in and observation of these attempts to build consensus has led to the development of the Cavalini-Cook Theory stating that: The probability of reaching consensus among biomedical experts tends to zero with the increase of; the number of concepts considered and the number of experts included in the consensus panel.

In MLHIM, participants at any level are encouraged to create domain knowledge models that fit their needs. The RM has very little semantic context in it to get in the way. This allows structures to be created as the modeler sees fit for purpose. There is no inherent idea of a specific application in the RM, such as an Electronic Health Record (EHR), Electronic Medical Record (EMR), etc, although the MLHIM specifications can also be adopted for the development of these types of applications. This provides an opening for small, purpose specific apps such as mobile or portable device software as well.

In MLHIM, there is room for thousands of CCDs to describe each healthcare concept, (e.g. blood pressure, body temperature, problem list, medication list, Glasgow Coma Scale, cost of a medical procedure, or any other healthcare phenomena) vs. a single, flat model implemented in software that must encompass all descriptions/uses/etc. This multiplicity of compatible domain knowledge models is achieved by the way CCDs are uniquely identified by a Version 4 Universal Unique Identifier (UUID)[9] prefixed with 'ccd-'. CCDs are built out of pluggable complexTypes (PcTs) so that modelers can use granular definitions to create any size application form needed. Modelers and developers can create systems that allow users to choose between a selection of CCDs to include at specific points, at run-time.

With MLHIM CCDs you can deliver your data with complete syntactic interoperability and as much semantic interoperability and information exchange as the modeler chose to include in the CCD.

The governance of CCDs is left to the modeler and/or publishing organization. There are very strict guidelines that define what constitutes a valid CCD, as seen above.

A Valid CCD Must:

- Be a valid XML Schema 1.1 schema as determined by widely available parser/validators such as Xerces[10] or Saxon[11]
- Consist of complexTypes that only use restriction of complexTypes from the associated reference model

9 http://en.wikipedia.org/wiki/Universally_unique_identifier
10 http://xerces.apache.org/xerces2-j/faq-xs.html
11 http://www.saxonica.com/documentation/schema-processing/

- Import the reference model schema from www.mlhim.org using the appropriately defined namespace. Example for release 2.4.4

  <xs:import schemaLocation='http://www.mlhim.org/xmlns/mlhim2/2_4_4/mlhim244.xsd' namespace='http://www.mlhim.org/xmlns/mlhim2/2_4_4'/>
- use Type 4 UUIDs for complexType names, with the prefix of, 'ct-'. Example:

  <xs:complexType name='ct-8c177dbd-c25e-4908-bffa-cdcb5c0e38e6' xml:lang='en-US'>

  the language attribute is optional.
- publish a global element for each complexType with the name defined using the same UUID as the complexType with the 'ct-' prefix replaced with 'el-'. Example:

  <xs:element name='el-8c177dbd-c25e-4908-bffa-cdcb5c0e38e6' substitutionGroup='mlhim244:DvAdapter-dv' type='ccd:ct-8c177dbd-c25e-4908-bffa-cdcb5c0e38e6'/>
- use the correct substitution group(s) as in the example above
- define the namespaces in accordance with the namespaces in Table 1. An example from a CCD is shown in Figure 1.
- define the minimum DCMI[12] metadata items as shown in Figure 2.

prefix	namespace
mlhim244	http://www.mlhim.org/xmlns/mlhim2/2_4_4
ccd	http://www.mlhim.org/ccd
xs	http://www.w3.org/2001/XMLSchema
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
owl	http://www.w3.org/2002/07/owl#
dc	http://purl.org/dc/elements/1.1/
rdfs	http://www.w3.org/2000/01/rdf-schema#
targetNamespace	http://www.mlhim.org/ccd

Table 2: Namespaces Table.

Example schema fragment showing namespace definitions:

12 http://dublincore.org/

```
<?xml version='1.0' encoding='UTF-8'?>
<xs:schema  xmlns:xs='http://www.w3.org/2001/XMLSchema'
```

```
<!-- METADATA Section -->
<xs:annotation>
  <xs:appinfo>
  <rdf:RDF>
  <rdf:Description rdf:about='http://www.ccdgen.com/ccdlib/ccd-20117151-e438-4d6e-849c-e23650393cac.xsd'>
    <dc:title>Test 2.4.4 All Datatypes</dc:title>
    <dc:creator>Tim Cook tim@mlhim.org</dc:creator>
    <dc:contributor>None</dc:contributor>
    <dc:subject>Test 2.4.4 All Datatypes;Generic</dc:subject>
    <dc:source>http://www.mlhim.org</dc:source>
    <dc:rights>CC-BY http://creativecommons.org/licenses/by/3.0/</dc:rights>
    <dc:relation>None</dc:relation>
    <dc:coverage>Universal</dc:coverage>
    <dc:type>MLHIM Concept Constraint Definition (CCD)</dc:type>
    <dc:identifier>ccd-20117151-e438-4d6e-849c-e23650393cac</dc:identifier>
    <dc:description>
      Test 2.4.4 All Datatypes
      This CCD was created by the CCD-Gen application.
    </dc:description>
    <dc:publisher>MLHIM LAB, UERJ</dc:publisher>
    <dc:date>2014-05-20 15:54:47.073287+00:00</dc:date>
    <dc:format>text/xml</dc:format>
    <dc:language>en-US</dc:language>
  </rdf:Description>
  </rdf:RDF>
  </xs:appinfo>
</xs:annotation>
```

Figure 2: Minimum Meta-data

A Valid CCD Must Not:

- Contain any other language processing instructions required for validating instance data. For example; Schematron rules. While Schematron can be very valuable in some processing environments it is considered implementation specific and not part of the MLHIM interoperability framework.
- Import any XML Schema document other than its parent reference model schema.

Modeling Concepts

Approach

The MLHIM specifications are arranged conceptually, into packages. These packages represent logical groupings of classes providing ease of consistent implementation. That said, the MLHIM Reference Model is implemented in a single XML Schema. The fundamental concepts, expressed in the reference model classes, are based on basic philosophical concepts of real world entities. These broad concepts can then be constrained to more specific concepts using models created by domain experts, in this case healthcare experts.

In MLHIM 1.x these constraints were known as archetypes, expressed in a domain specific language (DSL) called the archetype definition language (ADL). This language is based on a specific model called the archetype object model (AOM). MLHIM 1.x is a fork of the open source openEHR specifications.

In MLHIM 2.x and later, a given domain knowledge model is implemented in an XML Schema (XSD), called a Concept Constraint Definition (CCD). This allows MLHIM to use the XML Schema language as well as other XML technologies, as the constraint language. This provides the MLHIM development community with an approach to using multi-level modeling in healthcare using standardized, widely available tools and technologies.

The attempt to design a maximal data model for a concept is still restricted to the knowledge and context of a particular domain expert. In effect, there can never be a maximal data model for a healthcare concept. This means that from a global perspective there may be several CCDs that purport to fill the same need. There is no conflict in the MLHIM world in this case as CCDs are identified using the UUID and there are no semantics in the processing and validation model. The CCD may be further constrained at the implementation level through the use of implementation templates in the selected framework. Examples are additional XML Schemas that provide further restrictions or HTML pages used for display and data form entry. These templates are constructed in the implementation and may or may not be sharable across applications. The MLHIM specifications do not play any role in defining what these templates look like or perform like. They are only mentioned here as a way of making note that applications will require a user interface template layer to be functional.

The real advantage to using the MLHIM approach to healthcare information modeling is that it provides for a wide variety of healthcare applications to be developed based on the broad concepts defined in the reference model. By having domain experts within the healthcare field to define and create the CCDs, they can then be shared across multiple applications so that the structure and semantics of the data is not locked into one specific application, but can be exchanged among many different applications. This properly implements the separation of roles between IT and domain experts.

To demonstrate the differences between the MLHIM approach and the typical data model design approach we will use two common metaphors.

1. The first is for the data model approach to developing software. This is where a set of database definitions are created based on a requirements statement representing an information model. An application is then developed to support all of the functionality required to input, manipulate and output this data as information, all built around the data model. This approach is akin to a jigsaw puzzle (the software application) where the shape of the pieces are the syntax and the design and colors are the semantics, of the information represented in an aggregation of data components described by the model. This produces an application that, like the jigsaw puzzle, can provide for pieces (information) to be exchanged only between exact copies of the same puzzle. If you were to try to put pieces from one puzzle into a different puzzle, you might find that a piece has the same shape (syntax) but the picture on the piece (semantics) will not be the same. Even though they both belong to the same domain of jigsaw puzzles. You can see that getting a piece from one puzzle to correctly fit into another is going to require manipulation of the basic syntax (shape) and /or semantics (picture) of the piece. This can also be extended to the relationship that the puzzle has a defined limit of its four sides. It cannot, reasonably, be extended to incorporate new pieces (concepts) discovered after the initial design.

2. The multi-level approach used in MLHIM is akin to creating models (applications) using the popular toy blocks made by Lego® and other companies. If you compare a box of these interlocking blocks to the reference model and the instructions to creating a specific toy model (software application), where these instructions present a concept constraint (implemented as a CCD in MLHIM). You can see that the same blocks can be used to represent multiple toy models without any change to the physical shape, size or color of each block. Now we can see that when new concepts are created within healthcare, they can be represented as instructions for building a new toy model using the same fundamental building blocks that the original software applications were created upon.

Constraint Definitions

Concept Constraint Definitions (CCDs) can be created using any XML Schema or even a plain text editor. However, this is not a recommended approach. Realistic CCDs can be several hundred lines long. They also require Type4 UUIDs to be created as complexType and element names.

An open source Constraint Definition Designer (CDD) has been started but is in need of a leader and developer community. MLHIM founders are eager to support the development of this tool. It will (eventually) be used to create constraint definitions. It is open source and we hope to build a community around its development. The CDD can be used now to create a shell XSD with the correct metadata entries. Each release is available on the MLHIM GitHub site.

There is also a conceptual model of the information using the mind mapping software, Freemind. It provides domain experts a method of building up the structures to define a certain healthcare concept. There is a tool being developed to convert these Freemind definitions into a standardized format for import and processing by the CCD-Gen. It can also be found on the GitHub Site as xml2ccd

CCD Identification

The root element of a CCD and all complexType and global elements will use Type UUIDs as defined by the IETF RFC 4122 See: http://www.ietf.org/rfc/rfc4122.txt

The filename of a CCD may use any format defined by the CCD author. The CCD author must recognize that the metadata section of the CCD must contain the correct RDF:about URI with this filename. As a matter of consistency and to avoid any possible name clashes, the CCDs created by the CCD-Gen also use the CCD ID (ccd-<uuid>.xsd) To be a viable CCD for validation purposes the CCD should use the W3C assigned extension of '.xsd'. Though many tools may still process the artifact as an XML Schema without it.

The MLHIM community considers it a matter of good artifact management practice to use the CCD ID with the .xsd extension, as the filename.

CCD Versioning

Versioning of CCDs is not supported by these specifications. Though XML Schema 1.1 does have supporting concepts for versioning of schemas, this is not desirable in CCDs. The reasons for this decision focus primarily around the ability to capture temporal and ontological semantics. A key feature of MLHIM is the ability to guarantee the semantics for all future time, as intended by the original modeler. We determined that any change in the structure or semantics of a CCD, constitutes a new CCD. Since the complexTypes are re-usable (See the PcT description below), an approach that tools should use is to allow for copying a CCD and assigning a new CCD ID. When a complexType is changed within this new CCD, it also must be assigned a new name along with its global element name.

Pluggable complexTypes (PcTs)

MLHIM CCDs are made up of XML schema complexTypes composed by restriction of the Reference Model complexTypes. This is the foundation of interoperability. What is in the Reference Model is the superset of all CCDs. Pluggable complexTypes (PcTs) are a name we have given to the fact that due to their unique identification the complexTypes can be seen as re-usable components. For example, a domain expert might model a complexType that is a restriction of DvStringType with the enumerations for selecting one of the three measurement systems for temperature; Fahrenheit, Kelvin and Celsius. This PcT as well as many others can be reused in many CCDs without modification. For this reason, the semantic links for PcTs are directly expressed in an appinfo section in each PcT.

Implementations

It is the intent of the MLHIM community to maintain implementations and documentation in all major programming languages. Volunteers to manage these are welcome.

XML Schema

The reference implementation is expressed in XML Schema. Each release package contains the reference model schema as well as this and other documentation. The release and current development schemas live at the namespace on MLHIM.org

Best Practices

The concept of best practices for modeling and for implementation is an evolving set of results. To accommodate new items of interest under this heading we are using the MLHIM specs Wiki.

See the table of contents here: https://github.com/mlhim/specs/wiki/1.-Best-Practices

The Reference Model

Please note: Any discrepancies between this document and the XML Schema implementation is to be resolved by the XML Schema. The automatically generated XML Schema documentation is available in PDF and downloadable HTML forms: http://mlhim.org/documents.html The sources are maintained on GitHub at https://github.com/mlhim/specs To get the most recent development version using git create a clone of : https://github.com/mlhim/specs.git

Assumed Types

There are several types that are assumed to be supported by the underlying implementation technology. These assumed types are based on XML Schema 1.1 Part 2 Datatypes. They should be available in your implementation language or add-on libraries. The names may or may not be exactly the same.

Non-Ordered Types:

Name	Description
anyType	Also sometimes called Object. This is the base class of the implementation technology.
boolean	Two state only. Either true or false.
string	The string data type can contain characters, line feeds, carriage returns, and tab characters.
normalizedString	The normalized string data type also contains characters, but all line feeds, carriage returns, and tab characters are removed.
token	The token data type also contains characters, but the line feeds, carriage returns, tabs, leading and trailing spaces are removed, and multiple spaces are replaced with one space.
anyURI	Specifies a Unique Resource Identifier (URI).
hash	An enumeration of any type with a key:value combination. The keys must be unique.
list	An ordered or unordered list of any type.
set	An ordered or unordered but unique list of any type.

Table 3: Assumed: Non-ordered types

Name	Description
dateTime	The dateTime data type is used to specify a date and a time. The dateTime is specified in the following form "YYYY-MM-DDThh:mm:ss" where: YYYY indicates the year MM indicates the month DD indicates the day T indicates the start of the required time section hh indicates the hour mm indicates the minute ss indicates the second The following is an example of a dateTime declaration in a schema: <xs:element name="startdate" type="xs:dateTime"/> An element in your document might look like this: <startdate>2002-05-30T09:00:00</startdate> Or it might look like this: <startdate>2002-05-30T09:30:10:05</startdate> **Time Zones** To specify a time zone, you can either enter a dateTime in UTC time by adding a "Z" behind the time - like this: <startdate>2002-05-30T09:30:10Z</startdate> or you can specify an offset from the UTC time by adding a positive or negative time behind the time - like this: <startdate>2002-05-30T09:30:10-06:00</startdate> or <startdate>2002-05-30T09:30:10+06:00</startdate>
date	The date data type is used to specify a date. The date is specified in the following form "YYYY-MM-DD" where: YYYY indicates the year MM indicates the month

Name	Description
	DD indicates the day An element in an XML Document might look like this: <start>2002-09-24</start>
time	The time data type is used to specify a time. The time is specified in the following form "hh:mm:ss" where: hh indicates the hour mm indicates the minute ss indicates the second The following is an example of a time declaration in a schema: <xs:element name="start" type="xs:time"/> An element in your document might look like this: <start>09:00:00</start> Or it might look like this: <start>09:30:10:05</start> **Time Zones** To specify a time zone, you can either enter a time in UTC time by adding a "Z" behind the time - like this: <start>09:30:10Z</start> or you can specify an offset from the UTC time by adding a positive or negative time behind the time - like this: <start>09:30:10-06:00</start> or <start>09:30:10+06:00</start>
duration	The duration data type is used to specify a time interval. The time interval is specified in the following form "PnYnM-nDTnHnMnS" where: P indicates the period (required) nY indicates the number of years nM indicates the number of months nD indicates the number of days T indicates the start of a time section (required if you are going to

Name	Description
	specify hours, minutes, or seconds) nH indicates the number of hours nM indicates the number of minutes nS indicates the number of seconds The following is an example of a duration declaration in a schema: <xs:element name="period" type="xs:duration"/> An element in your document might look like this: <period>P5Y</period> The example above indicates a period of five years. Or it might look like this: <period>P5Y2M10D</period> The example above indicates a period of five years, two months, and 10 days. Or it might look like this: <period>P5Y2M10DT15H</period> The example above indicates a period of five years, two months, 10 days, and 15 hours. Or it might look like this: <period>PT15H</period> The example above indicates a period of 15 hours. **Negative Duration** To specify a negative duration, enter a minus sign before the P: <period>-P10D</period> The example above indicates a period of minus 10 days.
Partial Date Types	Support for partial dates is essential to avoid poor data quality. In order to provide for partial dates and times the following types are assumed to be available in the language or in a library. Day – provide on the day of the month, 1 – 31 Month – provide only the month of the year, 1 – 12 Year – provide on the year, CCYY MonthDay – provide only the Month and the Day (no year) YearMonth – provide only the Year and the Month (no day)

Name	Description
real	The decimal data type is used to specify a numeric value. Note: The maximum number of decimal digits you can specify is 18.
integer	The integer data type is used to specify a numeric value without a fractional component.

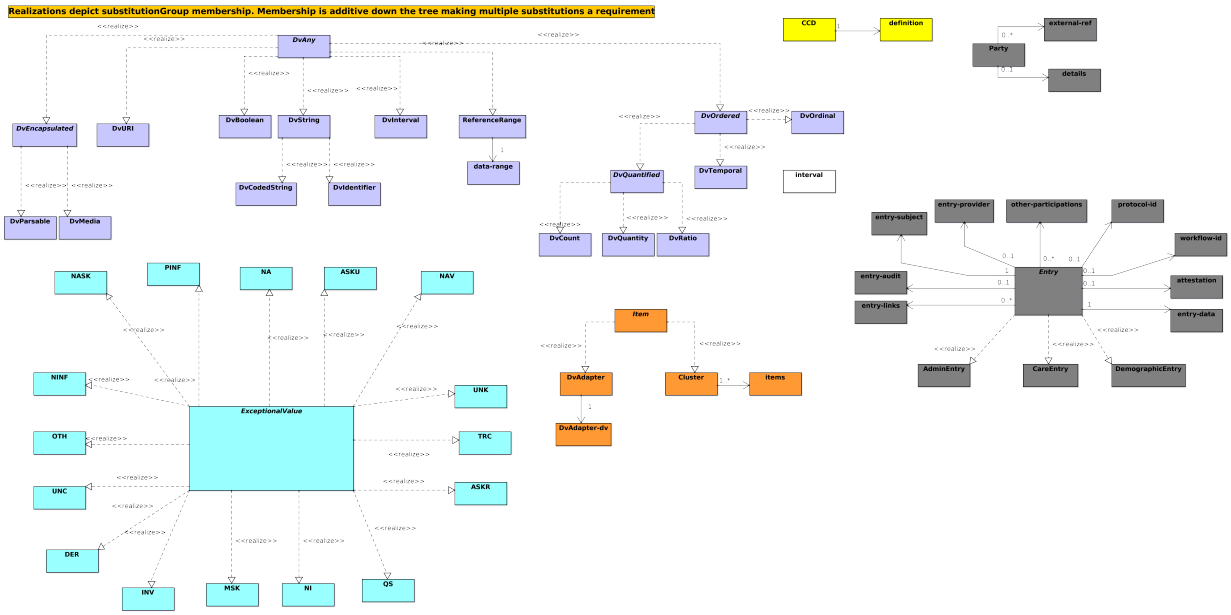
Table 4: Assumed: Ordered Types

2.4.5 Reference Model

The complete documentation in a graphical, clickable format is available on the MLHIM web-site via this Reference implementation link.

UML

Drawing 1 depicts the global elements of the reference implementation. This full size image as well as the ArgoUML project[13] is available in the UML directory of the specifications.



Drawing 1: Global Elements

13 Requires the included xsd.xmi profile.

complexTypes

The reference implementation complexType descriptions. The prefix 'xs:' is used to indicate W3C XML Schema datatypes.

DataValue Group

DvAnyType

Derived from: n/a

Abstract: True

Description: Serves as a common ancestor of all data-types in MLHIM models.

Name	Type	minOccurs	maxOccurs	Description
data-name	xs:string	1	1	Descriptive name of this fragment.
mlhim244:Excep-tionalValue	mlhim244:E xceptionalVal-ueType	0	1	Any of the restrictions of the ExceptionalValueType.
valid-time-begin	xs:dataTime	0	1	If present this must be a valid datetime including timezone. It is used to indicate the beginning time that information is consid-ered valid or when the information was collected.
valid-time-end	xs:dataTime	0	1	If present this must be a valid datetime including timezone. It is used to indicate the ending time that information is considered valid.

DvBooleanType

Derived from: DvAnyType **by extension**

Abstract: False

Description: Items which represent boolean decisions, such as true/false or yes/no answers. Use for such data, it is important to devise the meanings (usually questions in subjective data) carefully, so that the only allowed results are in fact true or false. The possible choices for True or False are enumerations in the CCD. The reference model defines valid-true and valid-false in a choice so only one or the other may be present in the instance data.

Potential Misuse: The DvBooleanType should not be used as a replacement for naively modelled enumerated types such as male/female etc. Such values should be coded, and in any case the enumeration often has more than two values. The elements, valid-true and valid-false are contained in an xs:choice and only one or the other is instantiated in the instance data with its value coming from the enumerations defined in a CCD.

Name	Type	minOccurs	maxOccurs	Description
valid-true	xs:string	0	1	A string that represents a boolean True in the implementation. These are generally constrained by a set of enumerations in the PcT.
valid-false	xs:string	0	1	A string that represents a boolean False in the implementation. These are generally constrained by a set of enumerations in the PcT.

DvURIType

Derived from: DvAnyType **by extension**

Abstract: False

Description: Used to specify a Universal Resource Identifier. Set the pattern to accommodate your needs in a CCD.

Name	Type	minOccurs	maxOccurs	Description
DvURI-dv	xs:anyURI	1	1	anyURI as a pointer.
relation	xs:string	1	1	Normally constrained by on ontology such as the OBO RO http://purl.obolibrary.org/obo/ro.owl

DvStringType

Derived from: DvAnyType **by extension**

Abstract: False

Description: The string data type can contain characters, line feeds, carriage returns, and tab characters.

Name	Type	minOccurs	maxOccurs	Description
DvString-dv	xs:string	0	1	The string value of the item.
language	xs:language	0	1	Optional indicator of the localized language in which this data-type is written. The ·value space· of language is the set of all strings that are valid language identifiers as defined [RFC 3066] . Only required when the language used here is different from the enclosing Entry.

DvCodedStringType

Derived from: DvStringType **by extension**

Abstract: False

Description: A text item whose DvString-dv element must be the long name or description from a controlled terminology. The key (i.e. the 'code') of which is the terminology-code attribute. In some cases, DvString-dv and terminology-code may have the same content.

Name	Type	minOccurs	maxOccurs	Description
terminology-abbrev	xs:string	0	1	Version Source Abbreviation (VSAB) from NLM Metathesarus; or similar.
terminology-name	xs:string	0	1	Full Source Name from NLM Metathesarus; or similar
terminology-version	xs:string	0	1	The proper release/version ID from the releasing authority.
terminology-code	xs:string	1	1	The uniquely identifiable code string from the terminology.

DvIdentifierType

Derived from: DvStringType **by extension**

Abstract: False

Description: Type for representing identifiers of real-world entities. Typical identifiers include: drivers license number, social security number, veterans affairs number, prescription id, order id, system id and so on. The actual identifier is in the DvString-dv element.

Name	Type	minOccurs	maxOccurs	Description
id-name	xs:string	0	1	The identifier common name, such as “Driver's License” or “SSN”.
issuer	xs:string	0	1	Authority which issues the kind of id used in the id field of this object.
assignor	xs:string	0	1	Organisation that assigned the id to the item being identified.

DvEncapsulatedType

Derived from: DvAnyType **by extension**

Abstract: True

Description: Abstract class defining the common meta-data of all types of encapsulated data.

Name	Type	minOccurs	maxOccurs	Description
size	xs:int	1	1	Original size in bytes of unencoded encapsulated data. I.e. encodings such as base64, hexadecimal, etc do not change the value of this attribute.
encoding	xs:string	0	1	Name of character encoding scheme in which this value is encoded. Coded from the IANA character set table: http://www.iana.org/assignments/character-sets Unicode is the default assumption in MLHIM, with UTF-8 being the assumed encoding. This attribute allows for variations from these assumptions.
language	xs:language	0	1	Optional indicator of the localised language in which the value is written. Coded IAW IETF RFC 5646 http://tools.ietf.org/html/rfc5646 language tag information should be used from the IANA registry http://www.iana.org/assignments/language-subtag-registry Only used when the text object is in a different language from the enclosing CCD.

DvParsableType

Derived from: DvEncapsulatedType **by extension**

Abstract: False

Description: Encapsulated data expressed as a parsable String. The internal model of the data item is not described in the MLHIM model, but in this case, the form of the data is assumed to be plain-text, rather than compressed or other types of large binary data. If the content is to be binary data then use a DvMediaType.

Name	Type	minOccurs	maxOccurs	Description
DvParsable-dv	xs:string	0	1	The string, which may validly be empty in some syntaxes.
formalism	xs:string	0	1	Name of the formalism, e.g. 'GLIF 1.0', 'ADL 1.4', etc.

DvMediaType

Derived from: DvEncapsulatedType **by extension**

Abstract: False

Description: A specialization of DvEncapsulatedType for audiovisual and bio-signal types. Includes further metadata relating to media types which are not applicable to other subtypes of DvEncapsulatedType.

Name	Type	minOccurs	maxOccurs	Description
mime-type	xs:string	0	1	MIME type of the original media-content w/o any compression. See IANA registered types: http://www.iana.org/assignments/media-types/index.html
compression-type	xs:string	0	1	Compression/archiving mime-type. Void means no compression/archiving. For a list of common mime-types for compression/archiving see: http://en.wikipedia.org/wiki/List_of_archive_formats
hash-result	xs:string	0	1	Hash function result of the 'media-content'. There must be a corresponding hash function type listed for this to have any meaning. See: http://en.wikipedia.org/wiki/List_of_hash_functions#Cryptographic_hash_functions
hash-function	xs:string	0	1	Hash function used to compute hash-result. See: http://en.wikipedia.org/wiki/List_of_hash_functions#Cryptographic_hash_functions
alt-txt	xs:string	0	1	Text to display in lieu of multimedia display or execution.
uri	xs:string	0	1	URI reference to electronic information stored outside the record as a file, database entry etc, if supplied as a reference.
media-content	xs:base64Binary	0	1	The content; if stored locally.

DvOrderedType

Derived from: DvAnyType **by extension**

Abstract: True

Description: Abstract class defining the concept of ordered values, which includes ordinals as well as true quantities. The implementations require the functions '<', '>' and is_strictly_comparable_to ('==').

Name	Type	minOccurs	maxOccurs	Description
reference-ranges	ReferenceRangeType	0	unbounded	Optional list of ReferenceRanges for this value in its particular measurement context. Implemented as restrictions on the ReferenceRangeType.
normal-status	xs:string	0	1	Optional normal status indicator of value with respect to normal range for this value. Often included by lab, even if the normal range itself is not included. Coded by ordinals in series HHH, HH, H, (nothing), L, LL, LLL, etc.

DvOrdinalType

Derived from: DvOrderedType **by extension**

Abstract: False

Description: Models rankings and scores, e.g. pain, Apgar values, etc, where there is

a) implied ordering,

b) no implication that the distance between each value is constant, and

c) the total number of values is finite.

Note that although the term 'ordinal' in mathematics means natural numbers only, here any decimal is allowed, since negative and zero values are often used by medical professionals for values around a neutral point. Also, decimal values are sometimes used such as 0.5 or .25

Examples of sets of ordinal values:

- -3, -2, -1, 0, 1, 2, 3 -- reflex response values
- 0, 1, 2 -- Apgar values

Used for recording any clinical datum which is customarily recorded using symbolic values.

Example: the results on a urinalysis strip, e.g. {neg, trace, +, ++, +++} are used for leukocytes, protein, nitrites etc; for non-haemolysed blood {neg, trace, moderate}; for haemolysed blood {neg, trace, small, moderate, large}

Name	Type	minOccurs	maxOccurs	Description
DvOrdinal-dv	xs:decimal	1	1	Value in ordered enumeration of values. The base integer is zero with any number of integer values used to order the symbols. Example 1: 0 = Trace, 1 = +, 2 = ++, 3 = +++, etc. Example 2: 0 = Mild, 1 = Moderate, 2 = Severe
symbol	xs:string	1	1	Coded textual representation of this value in the enumeration, which may be strings made from "+" symbols, or other enumerations of terms such as "mild", "moderate", "severe", or even the same number series as the values, e.g. "1", "2", "3".

DvQuantifiedType

Derived from: DvOrderedType **by extension**

Abstract: True

Description: Abstract type defining the concept of true quantified values, i.e. values which are not only ordered, but which have a precise magnitude.

Name	Type	minOccurs	maxOccurs	Description
magnitude	xs:decimal	0	1	Numeric value of the quantity in canonical (i.e. single value) form.
magnitude-status	xs:string	0	1	Optional status of magnitude with values: "=" : magnitude is a point value "<" : value is < magnitude ">" : value is > magnitude <= : value is <= magnitude ">=" : value is >= magnitude "~" : value is the approximate magnitude
error	xs:int	0	1	Error margin of measurement, indicating error in the recording method or instrument (+/- %). Implemented in subtypes. A logical value of 0 indicates 100% accuracy, i.e. no error.
accuracy	xs:decimal	0	1	Accuracy of the value in the magnitude attribute. 0% to +/- 100% A value of 0 means that the accuracy is unknown.

DvCountType

Derived from: DvQuantifiedType **by extension**

Abstract: False

Description: Countable quantities. Used for countable types such as pregnancies and steps (taken by a physiotherapy patient), number of cigarettes smoked in a day, etc. Misuse:Not used for amounts of physical entities (which all have standardized units). Note that PcTs derived from DvCountType should make magnitude, error and accuracy attributes minOccurs = '1'. The magnitude element is restricted to integers via an xs:assert.

Name	Type	minOccurs	maxOccurs	Description
DvCount-units	DvStringType	1	1	The name or type of the countable quantity. Examples: cigarettes, drinks, pregnancies, episodes, etc. implemented as a DvStringType restriction.

DvQuantityType

Derived from: DvQuantifiedType **by extension**

Abstract: False

Description: Quantified type representing “scientific” quantities, i.e. quantities expressed as a magnitude and units. Can also be used for time durations, where it is more convenient to treat these as simply a number of individual seconds, minutes, hours, days, months, years, etc. when no temporal calculation is to be performed. Note that PcTs derived from DvQuantityType should make magnitude, error and accuracy attributes minOccurs = '1'.

Name	Type	minOccurs	maxOccurs	Description
DvQuanity-units	DvStringType	1	1	Stringified units, expressed in unit syntax, e.g. "kg/m2", “mm[Hg]", "ms-1", "km/h". A DvCodedStringType should be used when possible. UOM codes can be found: http://www.obofoundry.org Also available in other terminologies such as SNOMEDCT; implemented as a DvStringType restriction.

DvRatioType

Derived from: DvQuantifiedType **by extension**

Abstract: False

Description: Models a ratio of values, i.e. where the numerator and denominator are both pure numbers. Should not be used to represent things like blood pressure which are often written using a '/' character, giving the misleading impression that the item is a ratio, when in fact it is a structured value. Similarly, visual acuity, often written as (e.g.) "6/24" in clinical notes is not a ratio but an ordinal (which includes non-numeric symbols like CF = count fingers etc). Should not be used for formulations.

Name	Type	minOccurs	maxOccurs	Description
ratio-type	xs:string	1	1	Indicates semantic type of ratio must be set as fixed to one of the below strings in PcTs. • **ratio** = a relationship between two numbers. • **proportion** = a relationship between two numbers where there is a bi-univocal relationship between the numerator and the denominator (the numerator is contained in the denominator) • **rate** = a relationship between two numbers where there is not a bi-univocal relationship between the numerator and the denominator (the numerator is not contained in the denominator)
numerator	xs:decimal	1	1	numerator of ratio
denominator	xs:decimal	1	1	denominator of ratio
numerator-units	DvStringType	1	1	Used to convey the meaning of the numerator. Typically countable units such as; cigarettes, drinks, exercise periods, etc. May or may not come from a terminology such as OBO Foundry Units ontology; implemented as a DvStringType restriction.
denominator-units	DvStringType	1	1	Used to convey the meaning of the denominator. Typically units such as; days, years, months, etc. May or may not come from a standard terminology; implemented as a DvStringType restriction.
ratio-units	DvStringType	0	1	Used to convey the meaning of the magnitude (ratio units). May or may not come from a standard terminology. In many cases there is not a meaningful term for the magnitude. Implemented as a DvStringType restriction.

DvTemporalType

Derived from: DvOrderedType **by extension**

Abstract: False

Description: Type defining the concept of date and time types. Must be constrained in PcTs to be one or more of the below elements. This gives the modeler the ability to optionally allow full or partial dates at run time. Setting maxOccurs and minOccurs to zero causes the element to be prohibited.

Name	Type	minOccurs	maxOccurs	Description
dvtemporal-date	xs:date	0	1	See the W3C documentation.
dvtemporal-time	xs:time	0	1	See the W3C documentation.
dvtemporal-date-time	xs:dateTime	0	1	See the W3C documentation.
dvtemporal-day	xs:gDay	0	1	See the W3C documentation.
dvtemporal-month	xs:gMonth	0	1	See the W3C documentation.
dvtemporal-year	xs:gYear	0	1	See the W3C documentation.
dvtemporal-year-month	xs:gYear-Month	0	1	See the W3C documentation.
dvtemporal-month-day	xs:gMonth-Day	0	1	See the W3C documentation.
dvtemporal-duration	xs:duration	0	1	See the W3C documentation.
dvtemporal-ymduration	xs:yearMonthDuration	0	1	See the W3C documentation.
dvtemporal-dtduration	xs:dayTimeDuration	0	1	See the W3C documentation.

DvIntervalType

Derived from: DvAnyType **by extension**

Abstract: False

Description: Generic type defining an interval (i.e. range) of a comparable type. An interval is a contiguous subrange of a comparable base type. Used to define intervals of dates, times, quantities, etc. Whose datatypes are the same and are ordered.

Name	Type	minOccurs	maxOccurs	Description
lower	mlhim245:interval-type	0	1	Defines the lower value of the interval.
upper	mlhim245:interval-type	0	1	Defines the upper value of the interval.
lower-included	xs:boolean	1	1	Is the lower value of the interval inclusive?.
upper-included	xs:boolean	1	1	Is the upper value of the interval inclusive?.
lower-bounded	xs:boolean	1	1	Is the lower value of the interval bounded?.
upper-bounded	xs:boolean	1	1	Is the upper value of the interval bounded?.

interval-type

Derived from: n/a

Abstract: False

Description: Defines the data type of the DvIntervalType upper and lower elements. In a CCD restriction, the xs:choice is constrained to one of the reference model elements with minOccurs = 1 and a fixed attribute defining the value. If the value is unbounded, then the element in the CCD will not have the fixed attribute. Instead it will have nillable="true" and an xs:assert to validate the instance has an empty element. E.g. <xs:assert test='boolean(invl-int/node()) = false()'/>

The instances must also declare the value as nil, e.g. <invl-int xsi:nil='true'/>

Name	Type	minOccurs	maxOccurs	Description
invl-int	xs:int	0	1	Defines the upper or lower interval datatype.
invl-decimal	xs:decimal	0	1	Defines the upper or lower interval datatype.
invl-float	xs:float	0	1	Defines the upper or lower interval datatype.
invl-date	xs:date	0	1	Defines the upper or lower interval datatype.
invl-time	xs:time	0	1	Defines the upper or lower interval datatype.
invl-datetime	xs:dateTime	0	1	Defines the upper or lower interval datatype.
invl-duration	xs:duration	0	1	Defines the upper or lower interval datatype.

ReferenceRangeType

Derived from: DvAnyType **by extension**

Abstract: False

Description: Defines a named range to be associated with any ORDERED datum. Each such range is particular to the patient and context, e.g. sex, age, and any other factor which affects ranges. May be used to represent normal, therapeutic, dangerous, critical etc ranges.

Name	Type	minOccurs	maxOccurs	Description
referencerange-definition	xs:string	1	1	Term whose value indicates the meaning of this range, e.g. “normal”, “critical”, “therapeutic” etc.
data-range	DvIntervalType	1	1	The data range for this meaning, as a restriction on a DvIntervalType.
is-normal	xs:boolean	1	1	True if this reference range only contains values that are considered to be normal.

Common Group

AuditType

Derived from: n/a

Abstract: False

Description: AuditType provides a mechanism to identifiy the who/where/when tracking of instances as they move from system to system.

Name	Type	minOccurs	maxOccurs	Description
system-id	DvIdentifierType	1	1	Identifier of the system which handled the information item.'Systems' can also be defined as an individual application or a data repository in which the data was manipulated.
system-user	PartyType	0	1	User(s) who created, committed, forwarded or otherwise handled the item.
location	ItemType	0	1	Location information of the particular site/facility within an organization which handled the item.
timestamp	xs:dateTime	1	1	Timestamp of handling the item. For an originating system, this will be time of creation,for an intermediate feeder system, this will be a time of accession or other time of handling, where available.

PartyType

Derived from: n/a

Abstract: False

Description: Description of a party, including an optional external link to data for this party in a demographic or other identity management system. An additional details element provides for the inclusion of information related to this party directly. If the party information is to be anonymous then do not include the details element. The string 'Self' may be entered as the party-name if an external_ref is include.

Name	Type	minOccurs	maxOccurs	Description
party-name	xs:string	0	1	Optional human-readable name (in String form).
external-ref	DvURIType	0	1	Optional reference to more detailed demographic or identification information for this party, in an external system.
details	ItemType	0	1	Structural details about the party.

AttestationType

Derived from: n/a

Abstract: False

Description: Record an attestation by a party of item(s) of record content. The type of attestation is recorded by the reason attribute, which my be coded.

Name	Type	minOccurs	maxOccurs	Description
attested-view	DvMediaType	0	1	Optional visual representation of content attested e.g. screen image.
proof	DvParsableType	0	1	Proof of attestation such as an GPG signature.
reason	DvStringType	0	1	Reason of this attestation. Coded from a standardized vocabulary.
committer	PartyType	0	1	Identity of person who committed the item.
time-commited	xs:dateTime	0	1	Datetime of committal of the item.
is-pending	xs:boolean	0	1	True if this attestation is outstanding; 'false' means it has been completed.

ParticipationType

Derived from: n/a

Abstract: False

Description: Model of a participation of a Party (any Actor or Role) in an activity. Used to represent any participation of a Party in some activity, which is not explicitly in the model, e.g. assisting nurse. Can be used to record past or future participations. Should not be used in place of more permanent relationships between demographic entities.

Name	Type	minOccurs	maxOccurs	Description
performer	PartyType	0	1	
function	DvStringType	0	1	
mode	DvStringType	0	1	
start-time	xs:dateTime	0	1	
end-time	xs:dateTime	0	1	

ExceptionalValueType

Derived from: n/a

Abstract: True

Description: Subtypes are used to indicate why a value is missing (Null) or is outside a measurable range. The element ev-name is fixed in restricted types to a descriptive string. The subtypes defined in the reference model are considered sufficiently generic to be useful in many instances. CCDs may contain additional ExceptionalValueType restrictions.

Name	Type	minOccurs	maxOccurs	Description
ev-name	xs:string	1	1	The fixed name of the exceptional value.

NIType

Derived from: ExceptionalValueType **by restriction**

Abstract: False

Description: *No Information* : The value is exceptional (missing, omitted, incomplete, improper). No information as to the reason for being an exceptional value is provided. This is the most general exceptional value. It is also the default exceptional value.

MSKType

Derived from: ExceptionalValueType **by restriction**

Abstract: False

Description: *Masked* : There is information on this item available but it has not been provided by the sender due to security, privacy or other reasons. There may be an alternate mechanism for gaining access to this information.

Warning:
Using this exceptional value does provide information that may be a breach of confidentiality, even though no detail data is provided. Its primary purpose is for those circumstances where it is necessary to inform the receiver that the information does exist without providing any detail.

INVType

Derived from: ExceptionalValueType **by restriction**

Abstract: False

Description: *Invalid* : The value as represented in the instance is not a member of the set of permitted data values in the constrained value domain of a variable.

DERType

Derived from: ExceptionalValueType **by restriction**

Abstract: False

Description: *Derived* : An actual value may exist, but it must be derived from the provided information; usually an expression is provided directly.

UNCType

Derived from: ExceptionalValueType **by restriction**

Abstract: False

Description: *Unencoded* : No attempt has been made to encode the information correctly but the raw source information is represented, usually in free text.

OTHType

Derived from: ExceptionalValueType **by restriction**

Abstract: False

Description: *Other*: The actual value is not a member of the permitted data values in the variable. (e.g., when the value of the variable is not by the coding system)

NINFType

Derived from: ExceptionalValueType **by restriction**

Abstract: False

Description: *Negative Infinity* : Negative infinity of numbers

PINFType

Derived from: ExceptionalValueType **by restriction**

Abstract: False

Description: *Positive Infinity* : Positive infinity of numbers

UNKType

Derived from: ExceptionalValueType **by restriction**

Abstract: False

Description: *Unknown* : A proper value is applicable, but not known.

ASKRType

Derived from: ExceptionalValueType **by restriction**

Abstract: False

Description: *Asked and Refused* : Information was sought but refused to be provided (e.g., patient was asked but refused to answer)

NASKType

Derived from: ExceptionalValueType **by restriction**

Abstract: False

Description: *Not Asked* : This information has not been sought (e.g., patient was not asked)

QSType

Derived from: ExceptionalValueType **by restriction**

Abstract: False

Description: *Sufficient Quantity* : The specific quantity is not known, but is known to non-zero and it is not specified because it makes up the bulk of the material; Add 10mg of ingredient X, 50mg of ingredient Y and sufficient quantity of water to 100mL.

TRCType

Derived from: ExceptionalValueType **by restriction**

Abstract: False

Description: *Trace* : The content is greater or less than zero but too small to be quantified.

ASKUType

Derived from: ExceptionalValueType **by restriction**

Abstract: False

Description: *Asked but Unknown* : Information was sought but not found (e.g., patient was asked but did not know)

NAVType

Derived from: ExceptionalValueType **by restriction**

Abstract: False

Description: *Not Available* : This information is not available and the specific reason is not known.

NAType

Derived from: ExceptionalValueType **by restriction**

Abstract: False

Description: *Not Applicable* : No proper value is applicable in this context e.g.,the number of cigarettes smoked per day by a non-smoker subject.

Items Group

ItemType

Derived from: n/a

Abstract: True

Description: The abstract parent of ClusterType and DvAdapterType structural representation types.

ClusterType

Derived from: ItemType **by extension**

Abstract: False

Description: The grouping variant of Item, which may contain further instances of Item, in an ordered list. This provides the root Item for arbitrarily complex structures.

Name	Type	minOccurs	maxOccurs	Description
cluster-subject	xs:string	1	1	Descriptive name of this branch.
items	ItemType	0	unbounded	List of Item types.

DvAdapterType

Derived from: ItemType **by extension**

Abstract: False

Description: The leaf variant of Item, to which any DvAnyType subtype instance is attached for use in a Cluster.

Name	Type	minOccurs	maxOccurs	Description
DvAdapter-dv	DvAnyType	0	unbounded	Data value type of this leaf.

NOTE: *The purpose for maxOccurs being unbounded is for validation of multiple instances of a DvAnyType subtype. This seems odd, but it is how it works. It is **NOT** for allowing multiple DvAnyType restrictions in one DvAdapterType.*

Entry Group

EntryType

Derived from: n/a

Abstract: True

Description: The abstract parent of all Entry subtypes. An Entry is the root of a logical set of data items. Each subtype has an identical information structure. The subtyping is used to allow persistence to separate the types of Entries; primarily import in healthcare for the de-identification of clinical information.

Name	Type	minOccurs	maxOccurs	Description
entry-links	DvURIType	0	unbounded	Optional link(s) to other locatable structures or external entities.
entry-audit	AuditType	0	unbounded	Audit trail from the system of original commit of information forming the content of this node through each system of handling the data.
language	xs:language	1	1	Mandatory indicator of the localised language in which this Entry is written. The value space of language is the set of all strings that are valid language identifiers as defined [RFC 3066]
encoding	xs:string	1	1	Name of character set encoding in which text values in this Entry are encoded. Default should be utf-8.
entry-subject	PartyType	0	1	Id of human subject of this Entry, e.g.: • subject of record (patient, etc.) • organ donor • foetus • a family member • another clinically relevant person.
entry-provider	PartyType	0	1	Optional identification of the source of the information in this Entry, which might be: • the patient• a patient agent, e.g. parent, guardian • the clinician • a device or software
other-participations	Participation-Type	0	unbounded	List of other participations at Entry level.
protocol-id	DvIdentifier-Type	0	1	Optional external identifier of protocol used to create this Entry. This could be a clinical guideline, an operations protocol,etc.
current-state	xs:string	0	1	The current state according to the state machine / workflow engine identified in workflow-id as a string.
workflow-id	DvURIType	0	1	Identifier of externally held workflow engine (state machine) data for this workflow execution.
attestation	Attestation-Type	0	1	Attestation recorded.
entry-data	ItemType	1	1	The data structure of the Entry.

CareEntryType

Derived from: EntryType **by extension**

Abstract: False

Description: Entry subtype for all entries related to care of an a subject of record.

AdminEntryType

Derived from: EntryType **by extension**

Abstract: False

Description: Entry subtype for administrative information, i.e. information about setting up the clinical process, but not itself clinically relevant. Archetypes will define contained information. Used for administrative details of admission, episode, ward location, discharge, appointment (if not stored in a practice management or appointments system). Not used for any clinically significant information.

DemographicEntryType

Derived from: EntryType **by extension**

Abstract: False

Description: Entry subtype for demographic information, i.e. name structures, roles, locations, etc. Modelled as a separate type from AdminEntryType in order to facilitate the separation of clinical and non-clinical information to support de-identification of clinical and administrative data.

Constraint Group

CCDType

Derived from: n/a

Abstract: False

Description: This is the root node of a Concept Constraint Definition.

Name	Type	minOccurs	maxOccurs	Description
definition	EntryType	1	1	Structural definition element for this CCD.

Example CCD

Please check the website documents section as well as the CCD Library on the CCD-Gen.

The CCD-Gen requires free registration in order to view the CCD Library.

MLHIM Data Analysis

The value in data and especially in semantically computable data is in how you can use it after it is captured. Each implementation will have different analysis requirements from static reports to dynamic dashboards and decision support. MLHIM data is designed to accommodate all of these use cases.

In many cases XQuery[14] is a natural choice for analyzing data. However, there is excellent support in all major programming languages for XML data and implementers may choose to develop in their favorite programming language.

The popular R[15] programming language is specifically designed for data analysis and supports XML data via an XML package[16] using the commonly used libxml2 library[17] used by most of the other languages. Libxml2 includes complete XPath, Xpointer and Xinclude implementations. There is also a companion library called libxslt[18] based on libxml2 that is used to provide support for XML transformations (XSLT[19])

The CCD-Gen54 provides example R code automatically generated for each CCD as an R project.

14 http://en.wikibooks.org/wiki/XQuery
15 http://cran.r-project.org/
16 http://cran.r-project.org/web/packages/XML/index.html
17 http://xmlsoft.org/
18 http://xmlsoft.org/XSLT/
19 http://www.w3.org/TR/xslt

Concept Constraint Definition Generator (CCD-Gen)

Creating and Managing CCDs

The CCD-Gen is an online tool used to create CCDs via a web driven, declarative environment. The CCD-Gen allows the building of complete CCDs by selecting the desired Pluggable complex-Type (PcT) definitions from existing PcTs or ones that you design yourself. You assemble the pieces into the concept definition you need. By re-using PcTs you improve the data exchange and analysis capability. Many of the PcTs have been designed based on existing data models. Resources such as the Common Data Element definitions from the US National Cancer Institute, the United States Health Information Knowledgebase (USHIK) from the Agency for Healthcare Research and Quality (AHRQ) and HL7 FHIR® models have been translated to PcTs. More are planned and you can contribute to the open content effort.

The CCD is an XML Schema conformant with the W3C XML Schema 1.1 standards. It is valid against one MLHIM Reference Model release. However, many of the PcTs maybe used in CCDs across various reference model releases. A CCD provides an approach to improve the process of allowing semantic interoperability between various applications. This process is enabled due the ability to exchange the syntax and semantics for data models designed by domain experts.

Data Instances

The CCD-Gen creates one sample instance along with every CCD. These samples are valid instances of the CCD schema. The one exception is when an *assert* is used in a PcT definition. The instance generator cannot (yet) create guaranteed valid data content for any arbitrary assert statement. If your sample is not valid, this is likely the case. For regular expressions (regexes) used to describe a string pattern in a DvString-dv element, the CCD-Gen can handle most patterns. Strings like postal codes, phone numbers, etc. should be randomly created okay. Other PcTs have no facility for xs:assert matching.

HTML Forms

There are hundreds of possibilities for uses of data when defining a model. Any given application component may write to multiple CCD instances at any one time. There is no *apriori* solution for this.

The CCD-Gen takes the most generic approach and creates a HTML form that depicts the nested nature of data defined by this CCD. This automatically generated form is more appropriately useful as a communications tool between the knowledge modeler (domain expert) and the software developer. It allows the developer to visually see the structure of the data.

This form also includes helpful information for the developer to relate the visual content to the CCD and data instance.

There is a tooltip on every structure name (Cluster, DvString, etc.) that provides the UUID for that structure. A HTTP link is also provided here that connects the developer to the specific documentation on the MLHIM website concerning that portion of the Reference Model.

The textual semantics for each component is also presented. A tooltip on these names/titles provided by the modeler displays the documentation as well as all links/semantic references that the modeler included in the PcT. This provides the developer with the same context that the modeler used when defining the PcT.

R Code

The CCD-Gen, in conjunction with each CCD, creates source code for use with the R statistical programming environment[20]. Specifically the code is created in a project suitable for use in R Studio[21], a package development and management tool for R.

The code is package complete and compliant for distribution via the *The Comprehensive R Archive Network*[22] (CRAN mirrors[23]). You can use *R Studio* or the *R tools* for package creation to generate the desired package format, either source or compiled binary for a specific platform (Linux, Mac, Windows, etc.). The code written by the CCD-Gen also contains package documentation (R help files) that can be created using the tools mentioned above.

The R code is created on a PcT level and will return a dataframe from all matching instances of a PcT. For example you may search across a complete repository of XML instances and receive a dataframe containing all occurrences of a specific PcT in a CCD. Users should note however that the generated R code will only return the first instance in a given file.

For more practical use, the code modules can be edited and combined into an application specific library and used to build any desired data structure based on the content of your repository. For example you may want to provide an additional loop or apply function to gather cases where multiple instances appear within a given DvAdapter node.

20 http://www.r-project.org/
21 https://www.rstudio.com/
22 http://cran.r-project.org/ (currently almost 6000 packages)
23 http://cran.r-project.org/mirrors.html

Customizing these basic routines allows the analyst to apply any of the thousands of algorithms available as R packages55 to their MLHIM based data repository.

Users and developers should note that the metadata.R file is not included in the NAMESPACES that are exported. However, it is required for every other R file since there are functions for XML namespace resolution located there. This file must be specifically 'sourced'. You may optionally include it in your NAMESPACE file.

Publications & Social Media

Publications, tutorials and presentations by the developers of MLHIM and other interested parties that relate to MLHIM.

Peer Reviewed

See the MLHIM website for the current listing.

Specifications Wiki

Read various pieces of content not in the documentation.

Code Examples

Release 2.4.5 examples of CCDs, instances and an XML Catalog.

SlideShare

http://www.slideshare.net/twcook

YouTube

http://www.youtube.com/user/MLHIMdotORG

FaceBook

https://www.facebook.com/mlhim2

Google Plus

MLHIM Community (primary support)

MLHIM page

Twitter

Follow MLHIM

Glossary

MLHIM:

Multi-Level Healthcare Information modeling. An open source/open content project with the goal of solving the healthcare information, semantic interoperability problem.

It uses a multiple level information modeling approach in combination with widely available tools and language infrastructure to allow the exchange of the syntactic and semantic information along with data.

PcT:

An acronym for Pluggable complexType. The name comes from the fact that it is a complete XML Schema complexType and that it can be reused or 'plugged in' to any CCD. This is due to the use of UUIDs for the complexType name attribute. Since complexType names must begin with an alphabetic character all MLHIM PcT names begin with the prefix 'ct-' followed by the UUID. This also facilitates the association with public element names in instances since they reuse the UUID but are prefixed with 'el-' in place of the 'ct-'.

CCD:

A MLHIM CCD is a Concept Constraint Definition. This is a data model that is created for a concept and it is created by expressing constraints on a generic reference model. In the MLHIM reference implementation eco-system these constraints are created through the use of the XML Schema restriction attribute on complexTypes. CCDs are immutable, once published. They are uniquely identified by the prefix 'ccd-' followed by a Type 4 UUID. They are valid against one version of the MLHIM Reference Model. This design gives them temporal durability and prevents the requirement to ever migrate instance data. The results of this approach is that all data, for all time in all contexts can be maintained with complete syntactic integrity and complete semantics.