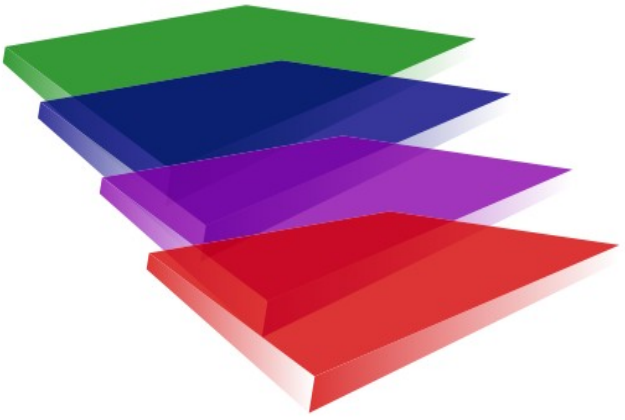
Multi-Level Health Information Modelling



MLHIM

Reference Manual

Release 2.4.2

DRAFT!

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

Table of Contents

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 Laboratory, Associated to the INCT-MACC (mlhim.lampada.uerj.br)
- 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 workers 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].

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 of the PDF release will carry 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 will have 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 Launchpad development site. It is easy to do and a great way to give back. See: https://bugs.launchpad.net/mlhim-specs

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.

Conformance

Conformance to these specification 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

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

Final versioned releases are available, packaged as .ZIP files from Launchpad:

http://launchpad.net/mlhim

Introduction

The Multi-Level Health Information Modeling (MLHIM) specifications are partially derived from ISO Healthcare Information Standards and the *open*EHR 1.0.2 specifications and the intent is that MLHIM 1.x be technologically inter-operable with *open*EHR.

MLHIM 2.x (this document and related artifacts) introduces modernization through the use of XML technologies and reducing complexity without sacrificing interoperability as well as improved modeling tools as well 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 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 be truly interoperable, available to everyone in all languages and most of all, implementable by mere mortals.

In 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. The domain knowledge models are implemented in the XML Schema language and they represent constraints on the reference model. These 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. However, any data instance should be able to be imported into any MLHIM based application since the root data model is the reference model. But, the full semantics of that data will not be known unless and until the defining CCD is available to that application.

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

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 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 small, purpose specific devices such as a home blood pressure monitor.

The next level of the multi-level 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. Basically this is inheritance 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. Even if the CCD is not available, an application can know how to display and even analyze certain information. However, the MLHIM eco-system concept does expect that every CCD for any published information is available to the receiving system(s).

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. Simply install a copy of the Healthcare Knowledge Component Repository (HKCR) (http://www.hkcr.net) and restrict access as required.

This is now the point where the MLHIM eco-system is in contrast to the top-down approach used by other multi-level modelling 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. In MLHIM, participants at any level are encouraged to create concept 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 EHR, EMR, etc. 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 blood pressure (or any other phenomena) vs. a single model that must encompass all descriptions/uses/etc. Each CCD is uniquely identified by a Version 4 Universal Unique Identifier (UUID)[1] prefixed with 'ccd_'. CCDs are pluggable so that modelers can use small CCDs to create any size concept, document, etc. needed. Modelers and developers can create systems that allow users to choose between a selection of CCDs to include at specific points, at runtime.

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

1 http://en.wikipedia.org/wiki/Universally_unique_identifier

The governance of CCDs is straight forward. A web application known as the HKCR lives at http://www.hkcr.net Anyone may install a copy of this for local use as well for performance reasons or if there is a need for some CCDs to not be published globally. Anyone may signup for an account on HKCR.net and anyone may create CCDs and submit them for publication. The only check that is made is that there is a valid XSD file. The content is completely up to the modeler and any potential users. Like other global meritocracies, those that create good and useful models will see theirs reused many times. Others, not so much.

MLHIM Modelling

Approach

The MLHIM specifications are arranged into packages. These packages represent logical groupings of classes providing ease of consistent implementation. That said, the MLHIM RM is contained in a single XML Schema, mlhim2.xsd. 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.y 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).

In MLHIM 2.x and later, we use an XML Schema (XSD) representation called a Concept Constraint Definition (CCD). This allows MLHIM to use the XML Schema language as well as XPath, as the constraint languages. This provides the MLHIM development community with an approach to using multi-level modelling in healthcare using standardized tools and technologies.

CCDs may contain other CCDs in a structure called a Slot. This provides a basis for selection and reuse, at runtime, of commonly occurring concepts within a larger context. A CCD is 'ideally', a maximal data model for a concept. However, as has been learned from other approaches, the concept of a maximal data model for a concept is still restricted to the knowledge and context of that particular domain expert. 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 named using the UUID. The CCD may be further constrained at the implementation level through the use of implementation templates in the selected framework. These templates shall be 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 health care information modelling is that it provides for a wide variety of healthcare applications to be developed based on the broad concepts defined in the reference model. Then by having domain experts within the healthcare field define and create the CCDs. They can then be shared across multiple applications so that the structure 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 people 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 six 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. 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. Relasitic CCDs can be several hundred lines and require Type4 UUIDs to be created as complexType and element names.

A Constraint Definition Designer (CDD) is being developed. It is a tool to 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 in the Tools section of HKCR.net. See: http://www.hkcr.net/tools

There is also a conceptual model of the information using the mind mapping software, XMind. It provides domain experts a copy/paste method of building up the structures to define a certain concept.

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 correct RDF:about URI must include this filename. As a matter of consistency and to avoid any possible name clashes, the CCDs on HKCR.net 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 research team 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 modeller. 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), an approach that tools should use is to allow for copying a CCD and assigning a new CCD ID. When a complexType is changed, it also gets a new ct-name.

Pluggable complexTypes

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 complexType that is a DvString 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 have been moved from the CCD meta-data section directly to an appinfo section in the PCT.

We are currently investigating whether PCTs should be published on HKCR.net as standalone (though non-operational) artifacts. Comments are encouraged.

Implementations

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

XML Schema

The reference implementation is expressed in XML Schema. Each release package contains mlhim2.xsd as well as this and other documentation. The release and current development schemas live at the namespace on MLHIM.org See: http://www.mlhim.org/xmls/mlhim2

Python

Looking for volunteers.

See: https://launchpad.net/oshippy

Java

Looking for volunteers.

See: https://launchpad.net/oshipjava

Ruby

Looking for volunteers.

See: https://launchpad.net/oshiprb

C++

Looking for volunteers.

See: https://launchpad.net/oshipcpp

Lua

Looking for volunteers.

See: https://launchpad.net/oshiplua

.Net

Looking for volunteers.

See: https://launchpad.net/oship

VisualBasic

Looking for volunteers.

See: https://launchpad.net/oship

We are open to suggestions for others.

Best Practices

The following is an unordered list drawn from implementers questions and experience. Primarily in creating CCDs but including some application development level suggestions as well.

How To Migrate a Legacy System

The first requirement is to have a good data dictionary of the existing system. Then convert each of these definitions into separate CCDs. The MLHIM research team at LA_MLHIM has experience converting Common Data Element definitions from the US National Cancer Institute to discrete CCDs. These CCDs can now be used to build larger concept CCDs for use in applications while remaining compliant with the structure and semantics of the legacy systems based on the NCI CDE. We used a simple Python script to extract the information from the NCI Standard Templates as .xls files. This script and example .xls files are available from the CDD repository at: https://code.launchpad.net/cdd

Other suggestions appreciated.

The Reference Model

Please note: Any discrepancies between this document and the XML Schema implementation is to be resolved by the XMLSchema. The automatically generated XML Schema documentation is available in PDF and downloadable HTML forms: http://www.mlhim.org/documentation/specs/schema-docs

The sources are maintained on Launchpad.net at: www.launchpad.net/mlhim-specs
The best way to get the most recent changes are to create a Bazaar branch using:
bzr branch lp:mlhim-specs

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 are described here.

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 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.

Ordered Types:

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
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 "PnYnMnDTnHnMnS" 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 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

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)

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.

Technical documentation

The MLHIM Reference Model is graphically depicted in the Xmind Template available from GitHub at https://github.com/mlhim/tools/blob/master/xmind_templates/MLHIM_Model-2.4.2.xmt

The reference implementation documentation, in XML Schema, is available from https://docs.google.com/file/d/0B9KiX8eH4fiKSk51Y1AxU0RGb0E/edit?usp=sharing

A sample Concept Constraint Definition (CCD) documentation is available at https://docs.google.com/file/d/0B9KiX8eH4fiKZ1QxT0kzT0ViRmM/edit?usp=sharing

An example instance document for this CCD is available at https://github.com/mlhim/tech-docs/blob/master/BMPinstance1.xml

Here are two example XQueries for that CCD as well:

1) https://github.com/mlhim/tech-docs/blob/master/BMP_Values.xql
2) https://github.com/mlhim/tech-docs/blob/master/BMP_ValuesAvg.xql

Healthcare Knowledge Component Repository

Managing CCDs

An open source content management system, the Healthcare Knowledge Component Repository (HKCR) is being deployed to provide an easy path for the development and distribution of CCDs on a global basis. See the HKCR documentation for more information.

OSHIP

The Open Source Health Information Platform (OSHIP) is a generic acronym for all implementations of the the MLHIM reference model in all programming languages. The basic concept is to supply a common information model for all healthcare applications, irregardless of the implementation language. For most implementation languages it is suggested that they use available tools and create bindings to the XML Schema provided as the reference implementation.

OSHIPpy

The MLHIM Reference Model implementation in Python. The reference model has been generated using generateDS. It is expected that implementers using Plone, Grok, Django, Web2py, etc. will create apps based on the reference model.

OSHIPjava

The MLHIM Reference Model implementation in Java. (pending implementation; taking volunteers)

OSHIPrb

The MLHIM Reference Model implementation in Ruby. (pending implementation; taking volunteers)

OSHIPcpp

The MLHIM Reference Model implementation in C++. (pending implementation; taking volunteers)

OSHIPlua

The MLHIM Reference Model implementation in Lua. (pending implementation; taking volunteers)