Multi-Level Health Information Modelling

MLHIM

Reference Manual & User Guide

Release 2.4.3

Timothy W. Cook & Contributors

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

Table of Contents

Front Matter....4
Acknowledgements....4
Using the MLHIM Reference Manual....4
Release Information....4
Error Reporting....4
Pronunciation....5
Conformance....5
Compliance....5
Availability....5
Introduction....6
The MLHIM Eco-System....7
MLHIM Modelling....9
Approach....9
Constraint Definitions....11
CCD Identification....11
CCD Versioning....12
Pluggable complexTypes....12
Implementations....12
XML Schema....12
Python....13
Java....13
Ruby....13
C++....13
Lua....13
.Net....13
VisualBasic....13
Best Practices....15
How To Migrate a Legacy System....15
The Reference Model....16
Introduction....16
Assumed Types....16
anyType....16
boolean....16
string....16
normalizedString....16
token....17
anyURI....17
hash....17
list....17
set....17
Ordered Types:....18
dateTime....18
Time Zones....18
date....19

time....19
duration....20
Partial Date Types....21
real....21
integer....21
Technical documentation....22
Using MLHIM....25
Examination of an instance document....25
Reusable Components....29
Semantic Links....30
MLHIM in the Real World....31
Resources....33
Changes 2.4.2 to 2.4.3....34

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
- Profa. Luciana Tricai Cavalini
- 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, XML Schema concepts (esp. version 1.1) 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 As an example, a release for this version will have the filename: mlhim-ref-man-2-4-3.pdf. The official package release site is https://launchpad.net/mlhim-specs the official schema is in the release package and also at the namespace URL. For example; http://www.mlhim.org/xmlns/mlhim2/2_4_3/mlhim243.xsd

Error Reporting

Please report all errors in documentation and/or in the specifications of the information model as bug reports on the GitHub development site. https://github.com/mlhim/specs/tree/2_4_3 It is easy to do and a great way to give back.

Pronunciation

MLHIM is pronounced ***muh-leem***.

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 specification releases are available at: http://www.launchpad.net/mlhim-specs

The development site for MLHIM specifications and tools is: https://github.com/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 interoperable with *open*EHR. The MLHIM 1.x branches on launchpad were never completed due to innovation discovered through implementation in XML Schema 1.1

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.

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 run-time.

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.

The governance of CCDs is straight forward; you manage the CCDs you need to use. A web application known as the **CCD-Gen** (CCD Generator) lives at http://www.ccdgen.com [2] Anyone may sign up for an account on CCDGen.com and anyone may may download publicly available CCDs. In the CCDs created via the CCD-Gen, 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.

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

2 The CCD-Gen is in closed beta at the time of this publication. Follow the discussions on Google Plus for notice of open availability.

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, mlhim<version number without dots>.xsd, e.g. mlhim243.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) version 1.1 representation called a **Concept Constraint Definition (CCD)**. This allows MLHIM to use the XML Schema language 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. This also provides native access to other XML eco-system tools such as XPath, XSLT, Xproc, etc. with easy translation to RDF for manipulation using 'big data' tools.

CCDs represent a concept model defined by the modeller. It may or may be in agreement with other modellers and users or any standards bodies. It 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.

The **CCD-Gen** (currently in closed beta release), is a webased commercial CCD Generator tool that will be available by subscription. Follow the MLHIM Google Plus Community or sign up for the MLHIM VIP mailing list http://goo.gl/22B0U for the latest information.

Also, the 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 on GitHub; See: https://github.com/mlhim/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 generated 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 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 in any manner*, it also gets a new ct-<uuid> name as well as a corresponding global element, el-<uuid>.

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 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 have been moved from the CCD metadata section directly to an appinfo section in the PCT.

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 mlhim<version id>.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 PCTs. The MLHIM research team at LA_MLHIM has experience converting Common Data Element definitions from the US National Cancer Institute to PCTs. These PCTs 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.

Other suggestions appreciated.

The Reference Model

Introduction

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 GitHub.com at: https://github.com/mlhim/specs
The best way to get the most recent changes clone the master branch:

via HTTP:
$git clone https://github.com/mlhim/specs.git

or via SSH:
$git clone git@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.

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

This has been commented on as a great training aid for modellers however as of 2.4.3 this is no longer maintained. If you wish to maintain this artifact please let us know in the Google Plus community.

The technical documentation of the reference model and of the example CCD are quite large and therefore can be found in separate files.

The reference implementation (XML Schema 1.1) of the reference model documentation:
HTML - Reference Model/index.html

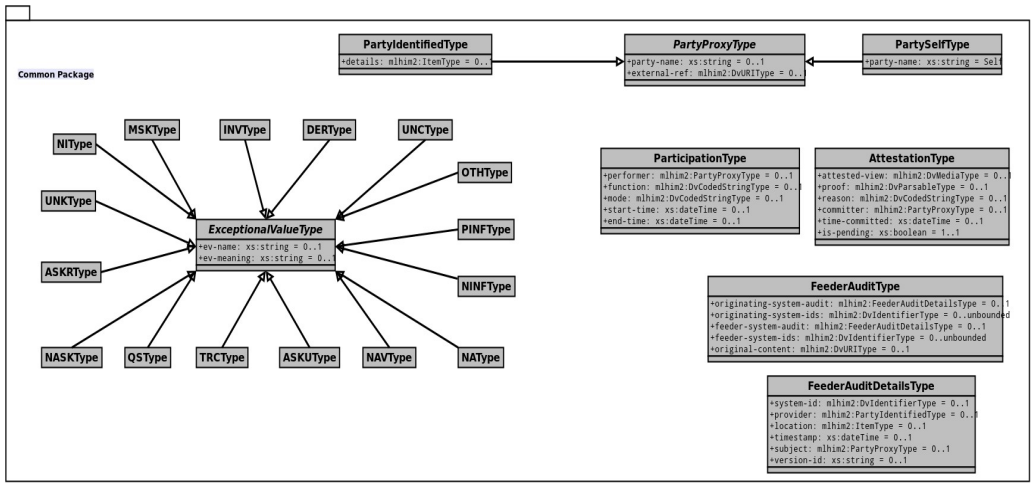
A sample Concept Constraint Definition (CCD) documentation:
HTML - Example CCD/index.html

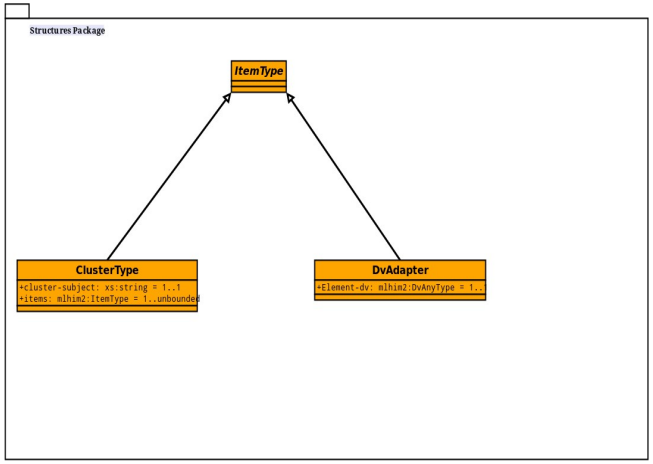
The example CCD is ccd-f2b7c24b-39c0-4ff4-b528-e7a509429572.xsd

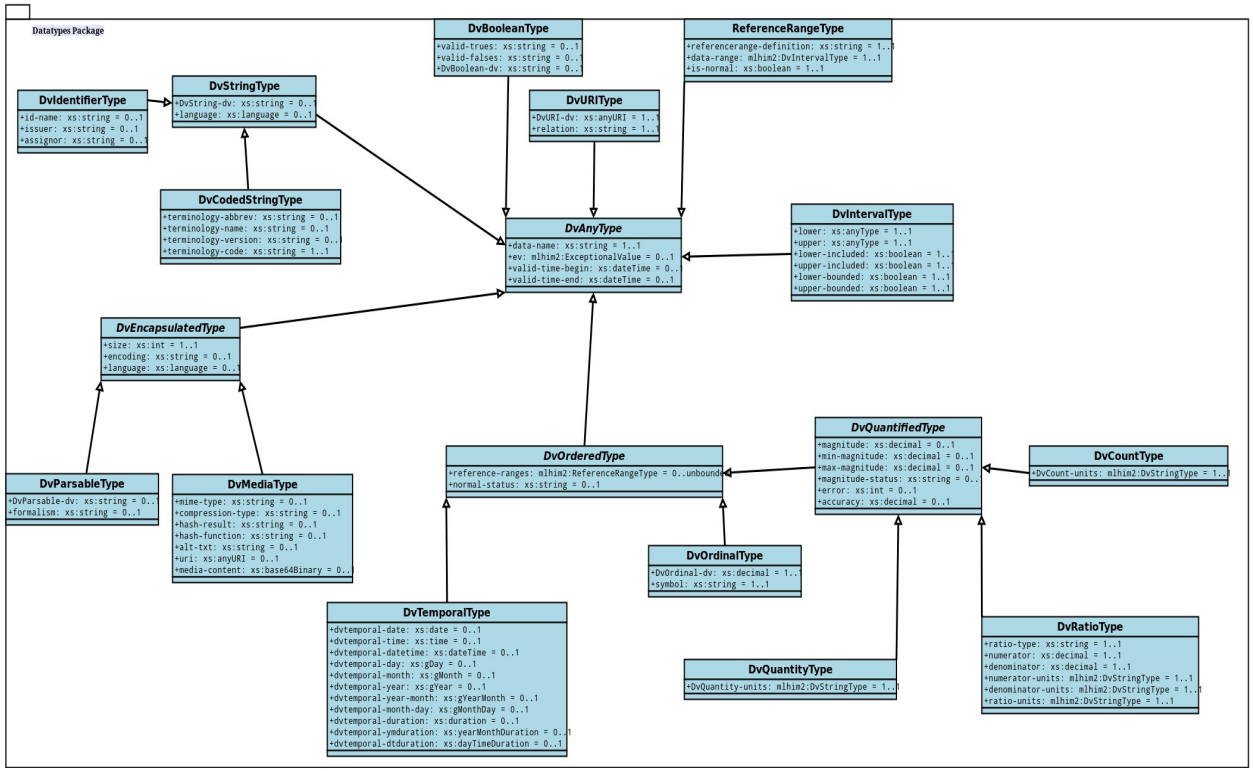
Please note that this CCD is completely non-sensical. It is an example that includes all of the datatypes and also includes the reference model links so that it is easier to understand the concept of the restrictions.

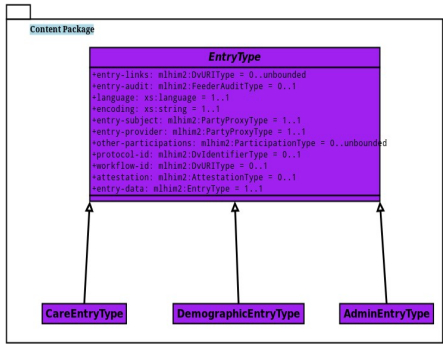
An example instance of this CCD is available as example-instance.xml.

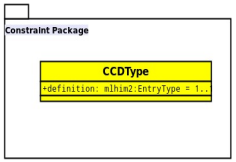
The Reference Model diagram on the next page is available in the documentation folder as a PNG so that the reader may zoom in to see the details.


Common Package
PartyIdentifiedType
+details: mlhim2:ItemType = 0..1
PartyProxyType
+party-name: xs:string = 0..1
+external-ref: mlhim2:DvURIType = 0..1
PartySelfType
+party-name: xs:string = Self
NIType
MSKType
INVType
DERType
UNCType
OTHType
UNKType
PINFType
ASKRType
NINFType
NASKType
QSType
TRCType
ASKUType
NAVType
NAType
ExceptionalValueType
+ev-name: xs:string = 0..1
+ev-meaning: xs:string = 0..1
ParticipationType
+performer: mlhim2:PartyProxyType = 0..1
+function: mlhim2:DvCodedStringType = 0..1
+mode: mlhim2:DvCodedStringType = 0..1
+start-time: xs:dateTime = 0..1
+end-time: xs:dateTime = 0..1
AttestationType
+attested-view: mlhim2:DvMediaType = 0..1
+proof: mlhim2:DvParsableType = 0..1
+reason: mlhim2:DvCodedStringType = 0..1
+committer: mlhim2:PartyProxyType = 0..1
+time-committed: xs:dateTime = 0..1
+is-pending: xs:boolean = 1..1
FeederAuditType
+originating-system-audit: mlhim2:FeederAuditDetailsType = 0..1
+originating-system-ids: mlhim2:DvIdentifierType = 0..unbounded
+feeder-system-audit: mlhim2:FeederAuditDetailsType = 0..1
+feeder-system-ids: mlhim2:DvIdentifierType = 0..unbounded
+original-content: mlhim2:DvURIType = 0..1
FeederAuditDetailsType
+system-id: mlhim2:DvIdentifierType = 0..1
+provider: mlhim2:PartyIdentifiedType = 0..1
+location: mlhim2:ItemType = 0..1
+timestamp: xs:dateTime = 0..1
+subject: mlhim2:PartyProxyType = 0..1
+version-id: xs:string = 0..1


Structures Package
ItemType
ClusterType
+cluster-subject: xs:string = 1..1
+items: mlhim2:ItemType = 1..unbounded
DvAdapter
+Element-dv: mlhim2:DvAnyType = 1..1


Datatypes Package
DvBooleanType
+valid-trues: xs:string = 0..1
+valid-falses: xs:string = 0..1
+DvBoolean-dv: xs:string = 0..1
ReferenceRangeType
+referencerange-definition: xs:string = 1..1
+data-range: mlhim2:DvIntervalType = 1..1
+is-normal: xs:boolean = 1..1
DvIdentifierType
+id-name: xs:string = 0..1
+issuer: xs:string = 0..1
+assignor: xs:string = 0..1
DvStringType
+DvString-dv: xs:string = 0..1
+language: xs:language = 0..1
DvURIType
+DvURI-dv: xs:anyURI = 1..1
+relation: xs:string = 1..1
DvCodedStringType
+terminology-abbrev: xs:string = 0..1
+terminology-name: xs:string = 0..1
+terminology-version: xs:string = 0..1
+terminology-code: xs:string = 1..1
DvAnyType
+data-name: xs:string = 1..1
+ev: mlhim2:ExceptionalValue = 0..1
+valid-time-begin: xs:dateTime = 0..1
+valid-time-end: xs:dateTime = 0..1
DvIntervalType
+lower: xs:anyType = 1..1
+upper: xs:anyType = 1..1
+lower-included: xs:boolean = 1..1
+upper-included: xs:boolean = 1..1
+lower-bounded: xs:boolean = 1..1
+upper-bounded: xs:boolean = 1..1
DvEncapsulatedType
+size: xs:int = 1..1
+encoding: xs:string = 0..1
+language: xs:language = 0..1
DvParsableType
+DvParsable-dv: xs:string = 0..1
+formalism: xs:string = 0..1
DvMediaType
+mime-type: xs:string = 0..1
+compression-type: xs:string = 0..1
+hash-result: xs:string = 0..1
+hash-function: xs:string = 0..1
+alt-txt: xs:string = 0..1
+uri: xs:anyURI = 0..1
+media-content: xs:base64Binary = 0..1
DvOrderedType
+reference-ranges: mlhim2:ReferenceRangeType = 0..unbounded
+normal-status: xs:string = 0..1
DvQuantifiedType
+magnitude: xs:decimal = 0..1
+min-magnitude: xs:decimal = 0..1
+max-magnitude: xs:decimal = 0..1
+magnitude-status: xs:string = 0..1
+error: xs:int = 0..1
+accuracy: xs:decimal = 0..1
DvCountType
+DvCount-units: mlhim2:DvStringType = 1..1
DvOrdinalType
+DvOrdinal-dv: xs:decimal = 1..1
+symbol: xs:string = 1..1
DvTemporalType
+dvtemporal-date: xs:date = 0..1
+dvtemporal-time: xs:time = 0..1
+dvtemporal-datetime: xs:dateTime = 0..1
+dvtemporal-day: xs:gDay = 0..1
+dvtemporal-month: xs:gMonth = 0..1
+dvtemporal-year: xs:gYear = 0..1
+dvtemporal-year-month: xs:gYearMonth = 0..1
+dvtemporal-month-day: xs:gMonthDay = 0..1
+dvtemporal-duration: xs:duration = 0..1
+dvtemporal-ymduration: xs:yearMonthDuration = 0..1
+dvtemporal-dtduration: xs:dayTimeDuration = 0..1
DvQuantityType
+DvQuantity-units: mlhim2:DvStringType = 1..1
DvRatioType
+ratio-type: xs:string = 1..1
+numerator: xs:decimal = 1..1
+denominator: xs:decimal = 1..1
+numerator-units: mlhim2:DvStringType = 1..1
+denominator-units: mlhim2:DvStringType = 1..1
+ratio-units: mlhim2:DvStringType = 1..1


Content Package
EntryType
+entry-links: mlhim2:DvURIType = 0..unbounded
+entry-audit: mlhim2:FeederAuditType = 0..1
+language: xs:language = 1..1
+encoding: xs:string = 1..1
+entry-subject: mlhim2:PartyProxyType = 1..1
+entry-provider: mlhim2:PartyProxyType = 1..1
+other-participations: mlhim2:ParticipationType = 0..unbounded
+protocol-id: mlhim2:DvIdentifierType = 0..1
+workflow-id: mlhim2:DvURIType = 0..1
+attestation: mlhim2:AttestationType = 0..1
+entry-data: mlhim2:EntryType = 1..1
CareEntryType
DemographicEntryType
AdminEntryType


Constraint Package
CCDType
+definition: mlhim2:EntryType = 1..1

Using MLHIM

Examination of an instance document

It is useful to go through a data instance to help understand the value and ease of use of MLHIM as an approach to semantic interoperability. We will not cover every element in the data instance but enough so that you may explore on your own. See the file example-instance.xml. It is an example that will validate against the CCD, ccd-f2b7c24b-39c0-4ff4-b528-e7a509429572.xsd

I suggest that you open the Reference Model file **mlhim243.xsd**, the CCD **ccd-f2b7c24b-39c0-4ff4-b528-e7a509429572.xsd** and the instance document **example-instance.xml**, preferably in an XML/XML Schema editor but a text editor will do. Having line numbers turned on will help. In the following discussion, line numbers are enclose in brackets, []. We will use **this color** to indicate if we are discussing a component from the CCD. **This color** when referring to the Reference Model. We will use **this color** when discussing the elements in the instance data.

Before we begin I want to point out that while they may look intimidating the UUIDs in the element names **do not** get in the way of human readability. The first element is obviously the CCD outer element. The other elements that begin with 'el-' followed by a Type 4 UUID are completely structural. I have added comments here just for the help in learning MLHIM.

As part of the first element, ccd-f2b7c24b-39c0-4ff4-b528-e7a509429572 **[2]**, there are a few attributes. These define the default namespace, the namespace for XML instance documents and the schema namespace with file location used to validate this document. Because we have a default namespace defined, xmlns="http://www.mlhim.org/xmlns/mlhim2/2_4_3" **[3]** and all of our elements are from the same namespace, we do not need to include a prefix.

Looking at the first inner element, el-da1b3744-3e15-4c9a-8dd1-30e9077de89e **[4]**, *you do need to know* that a CCD has one element and it is called 'definition'. In a CCD there is always one element that via a substitution group definition, replaces the CCD.definition element. It is always one of the three subtypes of the EntryType. Either AdminEntry, CareEntry or DemographicEntry.

The next element el-28856084-574f-488f-88e9-5ed2f1fc9eac **[5]** is for the entry-links element of the AdminEntry. If you check the CCD you will that the first element inside the AdminEntry refers to another element via ref='mlhim243:el-28856084-574f-488f-88e9-5ed2f1fc9eac' **[72]**. Of course that element needs a definition and you can find it on line **[2305]** of the CCD. Notice here that this element is part of the substitution group for mlhim243:entry-links. The type definition for this element is type='mlhim243:ct-28856084-574f-488f-88e9-5ed2f1fc9eac'.

Notice that the UUID is the same for the element and the complexType. The only difference is the first two characters; 'el' for element and 'ct' for complexType. This aides in being able to easily locate the associated components. Depending upon your editor you can search for the UUID 28856084-574f-488f-88e9-5ed2f1fc9eac and easily find the complexType definition **[91]** as well as any element definitions in the CCD. This is especially important where elements are referred to from inside a ClusterType via a DvAdapterType.

You will now see that this complexType is a restriction **[103]** of the DvURIType which is found in the reference model schema **[75]**. While we are in the reference model schema, take a look at line **[1323]** where you will see that the entry-links element is defined as type="mlhim243:DvURIType. So now we can see that entry-links is supposed to be a DvURI-Type and it is defined as such because the complexType ct-28856084-574f-488f-88e9-5ed2f1fc9eac is a valid restriction of it.

In the CCD **[105]** says that the element data-name is fixed (by the modeller) to the string 'Connected To'. If you are curious as to where this element, data-name came from, check the definition for DvURIType. It is an extension **[80]** of the abstract complexType DvAnyType. DvAnyType is the abstract parent of all datatypes and provides the global elements; data-name, ExceptionalValue, valid-time-begin, and valid-time-end. You will find their definitions for use in the schema documentation as well as in the schema itself.

Back to the CCD **[105]**. The modeller defined this DvURIType to restrict the data-name to the string Connected To. It is required (from the reference model **[15]** as can be seen from the min and max Occurs attributes. Notice that for the Exceptional Value we use the global element mlhim243:ExceptionalValue. We will briefly cover Exceptional Values later. The valid time elements are not required and the modeller has chosen to make them required in the CCD.

Up next are the two elements that DvURIType *extends* from DvAnyType. These are DvURI-dv and relation. A convention used in the MLHIM Reference Model is that the primary data content for each complexType is named with the type name followed by '-dv' (for data value). However, this convention is broken in the DvQuantifiedType children because we use the more descriptive term 'magnitude' for quantified types.

The DvURI-dv element must be a URI as defined by xs:anyURI. The relation element must be a string as defined by xs:string. In the case of the relation element the modeller will usually fix the value for this element as it defines the type of relationship that this CCD (actually instance data) has to the URI defined at runtime. This will become clearer in the sections on **Reusable Components** and **Semantic Links**.

Continuing with our AdminEntry on line **[12]** of the instance data we see the element entry-audit. Which is substituted by el-e479cd4b-8a05-4f4c-b55c-fde68d6f0dcb **[2306]** definition in the CCD. The complexType is ct-e479cd4b-8a05-4f4c-b55c-fde68d6f0dcb **[118]** which is a restriction of FeederAuditType from the reference model **[695]**. You can see that there is quite a lot of structural as well as datatype instance information in this and the FeederAuditDetailsType **[643]** in order to provide facilities for functional audit trail data. We will leave it to the reader to follow these connections through all of the element and complexType definitions. So much data in fact, that we go all the way down to line **[257]** in our instance data before we reach the next element in the AdminEntry. Here we find the language element and on the next line we find the encoding element. Both values are fixed by the modeller during CCD design.

The next element **[259]** el-83cfaa04-2563-43c0-8411-5a3adc7b12b4 is substituting for entry-subject line **[76]** in the CCD and line **[1276]** in the Reference Model. The complexType definition is ct-83cfaa04-2563-43c0-8411-5a3adc7b12b4 **[881]** and is a restriction of the Party-SelfType. Next **[262]** is the entry-provider substitution element el-83fc62c8-8df8-472a-b590-d8ecea82d695 and its type is a restriction of PartyIdentifiedType.

This basically covers how the Reference Model, CCD and instance data are connected for all complexTypes and this establishes the validation chain for the instances. There are a few other details so be prepared to skip around, following the line numbers and switching between documents.

The ClusterType from the Reference Model **[1197]** is very important in order to build CCDs to meet any requirement. It is elegant in its simplicity. Its parent is the abstract ItemType. The ClusterType has one element 'items' **[1210]** defined as type ItemType. Therefore, Clusters can contain Clusters. A Cluster can also contain the other child of ItemType, DvAdapterType. The number of Clusters and DvAdapters[3] is unbounded. Therefore any *shape of data structure* may be defined.

There are several Clusters in the example CCD but a good one to review **[260]** is ct-c4665b57-b737-4eef-b1a4-46ecd2298121 since it is something that is used often. Notice that the cluster-subject **[272]** is fixed to a string, PartyIdentified Details (BR), by the modeller. This is analogous to the data-name element in datatypes and both are part of the semantics of the CCD. This also makes reading the XML instance a bit easier since when you see the data-name or cluster-subject elements you can easily tell which part of the data you are looking at.

In the instance data you can see an example of the PartyIdentified Details (BR) Cluster element el-c4665b57-b737-4eef-b1a4-46ecd2298121 **[327]**. From the comments in the CCD you can see that this Cluster has a number of DvStrings that define some demographic information. This is because it is the details of an identified party. Notice that the details element in PartyIdentifiedType is a ClusterType. Again, any data shape can be used to properly identify a PartyIdentified. All depending on the needs of the modeller.

---

3 DvAdapters were called Elements in previous versions of MLHIM. This caused a great deal of confusion when discussing XML 'elements' and MLHIM 'Elements'.

We previously mentioned the DvAdapterType **[1223]** from the Reference Model. Remember that the Cluster item element can only have components defined as ItemType. The DvAdapter-Type is an extension of ItemType **[1228]** and therefore can be included in a Cluster.

Notice that there is one element defined in a DvAdapter **[1239]** and its type is DvAnyType which is the parent of all datatypes. So a DvAdapter, adapts a datatype so it can be included in a Cluster. In the CCD **[273]** you see the element el-7117e7ac-d879-4f8b-8719-2892050409e3 and the comment says <!-- DvString Bairro -->. So you might assume that el-7117e7ac-d879-4f8b-8719-2892050409e3 is defined as a DvStringType. But this isn't actually true. There must be an adapter in the middle and you can see that in the type definition **[307]**. The complexType ct-7117e7ac-d879-4f8b-8719-2892050409e3 is a restriction **[310]** of DvAdapterType and its DvAdapter-dv element is substituted **[2311]** by the reference to el-cc5c08c4-4426-4a68-a59a-b7bef285624d **[312]** which is defined as a complexType that is a DvStringType restriction **[295]**. This data definition is realized in the instance under the el-cc5c08c4-4426-4a68-a59a-b7be-f285624d element **[330]**.

As seen in the instance **[332]** the DvString-dv value is just a meaningless group of characters. There is no restriction on what can be valid in that position. Often you want to have better control over what may or may not be entered in a certain place. The long used xs:enumeration is one way to provide a list of the only valid choices for an element. Another way to provide more control over the data and or format is to use the new xs:assert in XML Schema 1.1.

An example of the xs:enumeration approach is shown in the CCD **[385]** definition for Brazilian State abbreviations. Only those strings in the enumeration values **[406 – 594]** are valid in instance data. If you are using a validating editor (like oXygen) or validation tools such as Xerces or Saxon, you can see this by editing the instance data **[53]**. Change the string 'PI' to 'PJ' (or anything that is not in that list of enumerations) and your data is no longer valid. Be certain to change it back to one of the valid choices so that you have a clean instance.

The other approach to formating data is by applying a boolean XPath statement in an xs:assert element in the CCD **[318]**. Here the assert **[336]** is a Regular Expression[4] (aka. Regex) test to insure that the string matches a format that represents a valid Brazilian postal code, five digits followed by a dash and three more digits. In the instance data **[41]** if you change one of the digits to any other character or have the wrong number of digits in each place, your data is no longer valid. The dash is optional.

In this CCD there are also asserts for latitude **[755]**, longitude **[798]** and Brazilian telephone numbers **[699]**. We leave it as an exercise for the reader use the UUID to find the elements in the instance data.

4 See http://goo.gl/xYEg and http://goo.gl/b8wnT

Reusable Components

You may have noticed that there are more than one reference to some of the elements in the CCD. For example the element ref='mlhim243:el-83cfaa04-2563-43c0-8411-5a3adc7b12b4' is referenced here **[76]** and here **[872]**. The complexType **[881]** tells us that this is a PartSelfType restriction. It is used to refer to the subject of the record in an abstract, non-identifying manner. This is useful when you want to know inside the originating application who the subject of the record is but when sharing this data, e.g. research, you want it to be anonymous.

This is an example of reusing a complexType inside of a CCD. It is only defined once and referenced in multiple places. This is possible via the XML Schema 1.1 multiple substitution group capability. Here **[2324]** you can see that element el-83cfaa04-2563-43c0-8411-5a3ad-c7b12b4 can be substituted for mlhim243:subject **[872]** and for mlhim243:entry-subject **[76]** and is defined as the type ct-83cfaa04-2563-43c0-8411-5a3adc7b12b4 defined here **[881]** as a restriction **[893]** of PartySelfType. We call these *Pluggable Complex Types* (PCTs).

However, PCTs are not restricted to reuse in one CCD. They can be reused in many CCDs. Think about how many different CCDs might need a postal code complexType. The process is to plug in your PCT and create the element definition with the correct substitution groups and references.

This approach of reuse extends to XQueries and application code that might need to refer to the same data in different contexts. The UUIDs are the same. So where I need to find the details of identification of a party, my XPath always looks like this; //el-83fc62c8-8df8-472a-b590-d8e-cea82d695/el-c4665b57-b737-4eef-b1a4-46ecd2298121 to get to that specif details Cluster. If I want the postal code I now that //el-83fc62c8-8df8-472a-b590-d8ecea82d695/el-c4665b57-b737-4eef-b1a4-46ecd2298121/el-96017054-3fdb-4819-af9b-892d386c132c/DvString-dv/text() will always return it.

Semantic Links

The entire purpose behind the *Multi-Level Healthcare Information Modelling* (**MLHIM**) project, which began in 2009, was to provide a way to produce a syntactically and semantically coherent framework using industry standard tools; for healthcare. We chose XML as the industry standard because it is ubiquitous across data management tasks. It is also the foundation of much of the *big data* and *semantic web* movements.

When we look at the healthcare (or any biological) domain, the data management landscape is different from domains that are engineered by humans. Humans build things as simply and economically as possible to meet the requirements. In the biological domains, evolution builds things as complex as necessary to insure survival of the species. In addition to this, our understanding of the biological sciences is constantly changing. However, as our understanding improves, we need to maintain previously collected data in the temporal, spatial and ontological contexts of the moment it was captured. Without these semantic contexts it is impossible to reason over it in the future with any degree of certainty.

In healthcare we use controlled vocabularies and ontologies to establish a shared meaning. But these artifacts evolve as knowledge changes. It is vital that we maintain a semantic link between data and its meaning *as it was* at the point in time it was captured. Migrating this data in software upgrades, etc. can distort the true meaning of the data.

As a *very small* example with a possibly large impact of the effects of this constant change. Take a look at how often reference ranges change for lab tests or other phenomena. These changes can lead to poor data quality and possibly even serious medical errors if the original reference ranges are not known when data is reviewed in the future. Medical errors is certainly the most important repercussion but the lack of correct reference ranges can lead to invalid malpractice claims.

The MLHIM solution to these issues begin with using UUIDs[5] to define complexTypes and their XML elements. This enormous number combined with the 'ct-' and 'el-' as well as specific namespaces for each Reference Model version, absolutely insures that there will never be a duplicate complexType/element name combination with different semantics; forever. This is why CCDs are not versioned. There is no need for the management overhead that versioned artifacts require.

Now that we have a known unique data model container, we can apply the semantics to it to further develop the meaning that is intended by the modeller at PCT creation time. We apply these semantics as two components of an xs:annotation element in the CCD.

First the xs:documentation contains a textual description in the language as defined in the PCT xml:lang attribute. Next, the xs:appinfo element contains RDF markup enclosed in an rdf:Description element with an rdf:about attribute pointing to the complexType name. This description can contain as many RDF elements and attributes as desired to fully define the PCT. An example is found in this CCD **[2207]**. Here we see that rdf:isDefinedBy and rdfs:member are used to expand the semantics of this concept model.

---

5 The number of possible UUIDs is 340,282,366,920,938,463,463,374,607,431,768,211,456 (16^{32} or 2^{128}), or about 3.4×10^{38}. Reference http://www.ietf.org/rfc/rfc4122.txt

Using this approach, we do not have to markup the instance data at all. The element name refers to a specific complexType definition with plain text and RDF semantics that describe what the modeller intended at the time of creating the PCT.

MLHIM in the Real World

Okayyyyyy …. this is all fine and dandy. But how do I really develop applications using MLHIM technology?

When considering MLHIM technology for application development or interoperability between legacy systems there are some things to consider. At the core you need to have some understanding of XML. Especially XML Schema 1.1 and XPath. You should also consider what type of application you are developing.

If you are developing a mobile application and you need to communicate with other systems then MLHIM is perfect for you. Once you develop your CCD, you share it with those that need to know about your data format. Then your application needs to only produce a valid instance file. You do not need to do validation on the mobile device. When you know that your instances are valid from the app, you can just send them. The receiver has the option to valid or trust them depending on the receiving application.

If you have a device or a set of sensors that collect data, you can also define your data format as a CCD and share that with those that need to import your data. MLHIM is perfect for device manufacturers that need to send data of any type. There is no need for manufacturers to have to squeeze their data into a format determined by some abstract, top down standards group in a conference room a world away.

If you are trying to interface two existing legacy systems and are struggling with the current approaches to messaging in healthcare, MLHIM is here to save the day. You define the CCDs you need to exchange all the messages you need. These models are exactly the size and type of data that **you** need to exchange, not the idea of a group of people that have never seen your applications. When you need to extend that out to other data exchange partners you can hand them a syntactically and semantically complete model, without ambiguity.

If you are developing a new application or suite of applications; you are in the right place. While MLHIM is very useful in interfacing legacy systems; if you build your applications based on MLHIM, you will be ready to exploit big data tools and technologies for the semantic web. While it isn't mandatory to use a NoSQL database for MLHIM, doing so will provide you with a huge reduction in long term workload. This is especially true using the open source databases like eXist-db or the commercial MarkLogic products. Just think, no more database schema migrations that can take months to years to design. Even then, create potential problems with data quality as we have already discussed.

But how do I build my CCDs? Okay, it is possible to build CCDs in something as simple as a text editor. You can use something like a Python script to crank out UUIDs and create your CCDs by hand. In fact, the first several CCDs were created in just this manner.

The reality is though that you really need tools so that knowledge modellers, not XML experts, can build the CCDs. There is an open source project in the GitHub MLHIM tools section that is kind of out dated and still needs work. However, I am happy to help guide any developers that want to build open source tools.

Also, we have a web-based tool called **CCD-Gen** that is currently in closed testing phase and will be available by subscription in the coming weeks. The example CCD with this release as well as all of the 2.4.2 release CCDs were created with this tool. There are videos on YouTube where we show the CCD-Gen in use. You can find the link under the **Resources** section below.

We are still accepting a few more CCD-Gen testers. If you have an interesting project and agree to our current usage guidelines, which are basically agree to be an adult and not delete other peoples PCTs. We will enjoy having you work with this tool which will also earn you a free subscription period when we go live. Send me an email describing your project at tim@mlhim.org

Thank you for enduring to this point. Be sure to check out the Resources section. Follow us on Google + and join the community. Like us on Facebook. Above all, ask questions and feel free to contribute your ideas. MLHIM is completely, openly available and free forever for anyone to use.

Kind Regards,

Tim Cook and the MLHIM Lab Team.

Resources

Google+ Page: http://gplus.to/MLHIM

Google+ Community: http://gplus.to/MLHIMComm

Facebook: https://www.facebook.com/mlhim2

Launchpad Umbrella page: http://launchpad.net/mlhim

Launchpad MLHIM Specifications: http://launchpad.net/mlhim-specs

The primary development site is on GitHub: https://github.com/mlhim

The wiki on the specs repository is a worthwhile read: https://github.com/mlhim/specs/wiki

Primary YouTube channel: https://www.youtube.com/user/MLHIMdotORG

Healthcare IT Live! Is a series of interviews and discussions with health informatics experts from across the globe: http://goo.gl/SIIKB

Healthcare Modelling Q&A has a lot of inforamtion about building CCDs and about the CCD-Gen: http://goo.gl/dvhZM

A couple of CCD-GEn specific videos: http://goo.gl/fgOsv

Healthcare Semantic Interoperability Challenges: http://goo.gl/rrKE6

CODATA 2012 – Taipei presentation: http://goo.gl/EFTRK

FHIES 2013 Presentation on MLHIM: http://goo.gl/qut1gd

and, of course Twitter: https://twitter.com/mlhim2

Changes 2.4.2 to 2.4.3

Reference the Issues on the GitHub repository for details.

- Renamed ElementType and Element to DvAdapterType and DvAdapter.
- Fixed typo in Attestation.time-committed.
- Changed to naming the schemas with the version number, e.g. mlhim243.xsd
- Changed the namespace definition for each version to include the version number, e.g. xmlns:mlhim243="http://www.mlhim.org/xmlns/mlhim2/2_4_3"

- Changed Attestation Reason, Participation Function, Participation Mode to DvStringType from DvCodedStringType to allow for either.
- Removed the example assert from DvBoolean since it was useless. There are useful examples such as the assert in DvCountType.
- Changed the ExceptionalValue.ev_meaning setting to 'default' from 'fixed'. In some cases the long string is easily wrapped or ends up with extra whitespace. This cause an invalid instance when set to 'fixed'.
- What was Element-dv (now DvAdapter-dv) had maxOccurs is 'unbounded in RM schema. It should be maxOccurs=1 and minOccurs=1
- Removed min-magnitude and max-magnitude elements. They are irrelevant since the proper place is as facets on magnitude.