MLHIM Documentation

Release 2.5.0

Timothy W. Cook

January 10, 2016

Contents

CHAPTER 1

MLHIM User & Reference Guide

Status - 2.5.0 Released - 2015-09-06

Contents:

1.1 MLHIM Docs Overview

Implementable semantic interoperability.

Status - **Final for 2.5.0** Released: 2015-09-06

1.1.1 MLHIM User & Reference Manual

Use the Table of Contents on the left for navigation.

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

Press the play button to hear **MLHIM** pronounced.

1.1.2 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 (INpcm-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 (1.0.x version)

1.1.3 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

1.1.4 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 for implementation of computable semantic interoperability 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 contains a growing number of demonstration projects and tools that research how this approach and focused 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.

1.1.5 Conformance

Conformance to these specifications are represented in a Language Implementation Specification (LIS). A LIS is a 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.

1.1.6 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

1.1.7 Availability

The MLHIM specifications, reference implementation and tools are available from GitHub

Previously released final versions are available, packaged as .ZIP files from Launchpad

From Release 2.4.7 on, you may download a zip file from the GitHub repository using the "Download ZIP" button. Click here to download the 2.5.0 *current development* version.

Official release are listed here.

The ninety and nine are with dreams, content but the hope of the world made new, is the hundredth man who is grimly bent on making those dreams come true. - Edgar Alan Poe

1.2 Governance

1.2.1 Specifications & Reference Implementation

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. You can also find us (occasionally) on the IRC network, Freenode channel #mlhim.

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

1.2.2 Other Artifacts

Governance of knowledge models is a local user issue. Where local user can be defined as an individual or a nation. In other words, governance of CCDs is up to the modeler and or their governing body. The MLHIM team does not control these models. As a modeler you may create them and share them (or not) as you wish.

The point that must be re-iterated here are:

- that the source CCD is expressed in an XML Schema 1.1 file and is valid against one MLHIM Reference Model schema.
- the PCMs (complexTypes) are named in accordance witht he specifications using a Type4 UUID and preceeded with the string 'pcm-'.

The CCD-Gen maintained by the MLHIM team does have its own governance policies.

1.3 Executive Summary

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

MLHIM is an agile and open framework for computable semantic interoperability in healthcare. This capability is required in order to advance the usage of clinical (and other) decision support services (DSS) and Big Data analysis across data coming from multiple points. These points of data creation are increasing every day as a result of more institutions bringing new digital capabilities to more functions, in addition to the plethora of new devices arriving in the personal healthcare marketplace.

MLHIM is not a *standard* in the historical sense of the term. The standards development process is far from agile; healthcare is too dynamic to wait for the 20th century standards processes. In addition, healthcare is too broad for a top-down, one-size-fits-all solution for information interoperability. MLHIM *is based on* widely used standards for *Linked Data* exchange and expression on the semantic web.

MLHIM uses these tools in conjunction with the Unix/Linux model of iterate and release often. A new release does not invalidate the previous models or data.

In designing MLHIM we took the view that data is produced by people that have a need to record that data. Often that data is useful to others. In healthcare this is certainly true. To transfer raw data is not enough for the receiver

to determine if it is useful for their needs. They need to know much more about the contexts (ontological, spatial, temporal) in which the data was captured.

If I have a leaky faucet in my home and need a washer to repair it. I go to the local hardware store to purchase a new one. The attendant there doesn't know what I need because they do not know what type of faucet I have. However, I can look at the package information and determine based on thickness, inside and outside diameters and shape (flat, conical, round, etc.) if a particular washer fits my needs. In this sense, MLHIM is the packaging for data. It allows the data consumer to determine if the data is useful to them as is or maybe it can even be adjusted because the consumer knows the context in which the data was captured. Of course there are certain base standards in the plumbing industry. Just like we have datatypes (string, integer, dateTime, etc.) and XML constructs (element, complexType, etc.). But the overall concept is similar.

MLHIM can be implemented as the information model for applications ranging from mobile apps to enterprise suites. MLHIM is also useful to enhance semantic interoperability of existing systems and healthcare IT interfaces such as those based on HL7 messaging.

MLHIM specifically addresses the need to communicate syntactically sound data with the intended semantics. The larger, point to point issues of information exchange are already handled by other formal and de-facto standards such as SOAP, REST, etc. and MLHIM is fully capable of utilizing systems built on those standards.

We have a growing amount of *peer-reviewed* material on our website as well as invited presentations and educational material on SlideShare and YouTube.

MLHIM is the foundation for retro-fitting your current applications to interoperate with your future applications using the semantic web tools and Linked Data, without requiring wholesale replacement.

For every complex problem there is an answer that is clear, simple, and wrong. - H. L. Mencken

1.4 Introduction

1.4.1 History

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.

See the Governance document for more information on the history of MLHIM.

The 1.x *attempts* are available on Launchpad if anyone is interested in reviving the effort.

1.4.2 Current Versions

MLHIM 2.x (this document and related artifacts) introduces **further innovation and a departure from previous approaches** that haven't worked, 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 interoperability, available to everyone in all languages.

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.rst for the Python language or MLHIM-Java-LIS.rst for the Java language. Add the LIS to the docs directory of the current development version and submit a pull request.

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, not define specific application behaviors.

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 any SQL, NoSQL or other persistent storage. For devices or other small apps even the file system will suffice as a storage solution.

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. CCDs are immutable. This insures data validity for longitudinal records.

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 [1] concept using XML Schema constraints and contains the semantics defined by the modeler in the form of RDF/XML 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 MLHIM approach allows systems developers to build the data model they need for their application and the shared CCDs tell you how and what was captured, not what to capture. This sharable information model allows any other data consumer to determine if the data is correct for their needs.

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 Modeling Concepts document for further discussion of Concept Constraint Definitions (CCDs).

1.5 MLHIM Abstract Model

MLHIM is by name as well as by definition and design a multi-level modeling approach. This means that there are multiple models with increasing specificity to get to the instance data point. MLHIM is constraint based which provides a complete syntactic validation path back to the reference model for the instance data. The semantic model is designed around the concepts of this multi-level model approach.

When answering the high-level question: *How do we elaborate the components required for a generic, implementation independent interoperability platform?* These few components were the answer.

[1] 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

1.5.1 MLHIM2

Multi-Level Healthcare Information Modeling

The root concept. The abstract idea of MLHIM 2.x. All of the MLHIM2 classes are subclasses of this class.

1.5.2 RM

Reference Model

A set of components (CCMs) to provide structural integrity for a domain concept. Some CCMs are mandatory in CCDs and some are optional. Optionality is defined in each RM implementation.

1.5.3 CoreCM

Core Concept Model

A composable model contained in a reference model. A CoreCM represents a specific core type of component that further contains elements with base datatypes and/or other CoreCMs to define its structure.

1.5.4 CoreCS

Core Concept Symbol

A CoreCS represents a CoreCM in instance data. In practice, it is usually substituted for by a PluggableCS. This substitution is due to the fact that constraints are expressed in a PluggableCM which is then represented by a PluggableCS. However the overall constraint model (CCDInstance) must match with components of the RM.

1.5.5 CCDInstance

Concept Constraint Definition Instance

A set of selected PluggableCMs that are constraints on the RM components (CoreCMs) in order to represent a domain concept. In the implementation language there may be additional syntactic conventions required.

1.5.6 PluggableCM

Pluggable Concept Model

The name given to a CoreCM that has been constrained for use in a CCDInstance. Through the constraints, a PluggableCM defines a single concept based on syntactic data constraints as well as specified semantics. It is *pluggable* because it can be reused in multiple CCDInstances.

1.5.7 PluggableCS

Pluggable Concept Symbol Represents a PluggableCM in instance data. Can be considered as a data container for the components of a PluggableCM.

1.5.8 DataInstance

A set of data items that reports via *isInstanceOf* property that it conforms to a CCDInstance. In this state it has not been tested for validation.

1.5.9 DataInstanceValid

Subclass of DataInstance. A set of data items that conforms to a CCDInstance to represent an instance of that concept **AND** the data values are valid according to the CCD Instance constraints.

1.5.10 DataInstanceInvalid

Subclass of DataInstance. A set of data items that conforms to a CCD Instance to represent an instance of that concept **AND** the data values are **NOT** valid according to the CCD Instance constraints. An Invalid Data Instance must contain one or more children of an Exception.

1.5.11 DataInstanceError

Subclass of DataInstance. A set of data items that **DOES NOT** conform to the CCD Instance it represents **OR** it contains invalid data and does not contain one or more children of an Exception.

1.5.12 Exception

Indicates that some data is outside of the parameters defined by the CCD Instance.

1.6 Modeling Concepts

1.6.1 Approach

The MLHIM Reference Model reference implementation 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, validation and processing 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 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 dependning upon the needs of the implementation.

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. Th applicatiuon UI as well as any transport layers between applications is outside the scope of the MLHIM eco-system.

The interoperability layer remains at the CCD instance level.

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. This is assuming the reference implementation based on XML Schema.

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. So, the various applications are now part of a larger infrastructure or eco-system of interoperability. Any application that understands the reference model can now interpret the the meaning conveyed in constraint definitions.

Constraint Definitions

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

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.

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 focuses primarily around the ability to capture temporal and ontological semantics for data instances and maintain them for all time (future proof data). 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 PCM 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, all ancestors (enclosing complexTypes) also must be assigned a new name along with its global element name. For example if the enumerations on a DvStringType restriction are changed, the DvStringType, the DvAdapterType, the parent ClusterType and any enclosing ClusterTypes, the EntryType and the CCDType must all get new UUIDs.

Pluggable complexTypes (PCMs)

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 (PCMs) 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 PCM as well as many others can be reused in many CCDs without modification. For this reason, the semantic links for PCMs are directly expressed in an xs:appinfo section in each PCM. This approach lends itself very well to the creation of RDF triples from this information. For example:

```
<xs:appinfo>
 <rdf:Description rdf:about='&mlhim2;pcm-3a54417d-d1d6-4294-b868-e7a9ab28f8c4'>
  <rdfs:isDefinedBy rdf:resource='http%3A//purl.obolibrary.org/obo/RO_0002371'/>
 </rdf:Description>
</xs:appinfo>
```

In this example the subject is &mlhim2;pcm-3a54417d-d1d6-4294-b868-e7a9ab28f8c4 the predicate is rdfs:isDefinedBy and the object is http%3A//purl.obolibrary.org/obo/RO_0002371

Every xs:appinfo section must begin with the rdf:Description element and have the rdf:about attribute to define the subject, as the containing complexType. This is then followed by one or more predicate/object components. The predicates can be from any vocabulary/terminology. Just be certain that the namespace prefix is correctly defined in the CCD header. The CCD-Gen defines common namespaces by default but others may be added as needed. Also be certain that any URLs are properly encoded so that they will be valid inside the CCD. RDF triples are a cornerstone of the semantic web. For more information see this tutorial. Of particular interest here is the section titled; Introducing RDF/XML. RDF/XML is one of the syntaxes used to describe semantic links and it is what we use in MLHIM. Another popular syntax you may see is called Turtle.

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 1.1. Each release package contains the reference model schema as well as this and other documentation. The release and current development schemas live at the versioned link on MLHIM.org. For example 2.5.0 is at: http://www.mlhim.org/ns/mlhim2/mlhim250.xsd A full release is available from GitHub. The previous release is 2.4.7

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

1.7 The Reference Implementation Reference Model

Note: Any discrepancies between this document and the XML Schema implementation is to be resolved by the XML Schema RM. 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 Then checkout the most recent branch.

1.7.1 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

boolean

Two state only. Either true or false.

string

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

anyURI

Specifies a Unique Resource Identifier (URI).

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

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)

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.

1.7.2 2.5.0 Reference Model Documentation

The complete documentation in a graphical, clickable format is available on the MLHIM website Documents page.

An EMF Ecore project is available in the docs folder of the distribution. It can be imported into Eclipse and used as a base for modeling CCDs. However, developers need to be aware that there are slight differences due to the fact that Eclipse XML tools do not support XML Schema 1.1

Further research is needed to determine if valid CCDs can be produced from Eclipse. Please let us know if you can help with this issue.

RM complexTypes

The reference implementation complexType descriptions.

Each complexType definition below has a Details. This link goes to a page with detailed documentation on that complexType.

DvAnyType

Details

Derived from: n/a

Abstract: True

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

DvBooleanType

Details

Derived from: DvAnyType by extension

Abstract: False

Description: An enumerated type which represents boolean decisions. Such as true/false or yes/no answers. Useful where 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 but are presented to the user as a list of options. The possible choices for True or False are enumerations in the CCD. The reference model defines 'true' and 'false' in a choice so only one or the other may be present in the instance data. The DvBooleanType should not be used as a replacement for enumerated choice types such as male/female, etc. Such values should be modeled as DvStrings with enumerations and may reference a controlled vocabulary. In any case the enumeration often has more than two values. The elements, 'true' and '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.

DvLinkType

Details

Derived from: DvAnyType by extension

Abstract: False

Description: Used to specify a Universal Resource Identifier. Set the pattern facet to accommodate your needs in the PCM. The primary use is to provide a mechanism that can be used to link together CCDs. The relation element allows for the use of a descriptive term for the link with an optional URI pointing to the source vocabulary. In most use cases the modeler will define all three of these using the 'fixed' attribute. Other use cases will have the 'relation' and 'relation-uri' elements fixed and the application will provide the 'link'.

DvStringType

Details

Derived from: DvAnyType by extension

Abstract: False

Description: The string data type can contain characters, line feeds, carriage returns, and tab characters. The use cases are for any free form text entry or for any enumerated lists. Additionally the minimum and maximum lengths may be set and regular expression patterns may be specified.

DvFileType

Details

Derived from: DvAnyType by extension

Abstract: False

Description: A type to use for encapsulated content (aka. files) for image, audio and other media types with a defined MIME type. This type provides a choice of embedding the content into the data or using a URL to point to the content.

New in 2.5.0

DvEncapsulated and its children were consolidated into this one concept and implemented as one complexType to represent any type file based artifact.

DvOrderedType

Details

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 ('==').

DvOrdinalType

Details

Derived from: DvOrderedType by extension

Abstract: False

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

- implied ordering,
- no implication that the distance between each value is constant, and
- 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 and other 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

Also used for recording any clinical or other datum which is customarily recorded using symbolic values. Examples;

- 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}.

Elements ordinal and symbol MUST have exactly the same number of enumerations in the PCM.

DvQuantifiedType

Details

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.

DvCountType

Details

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. The *thing(s)* being counted must be represented in the units element.

Misuse: Not used for amounts of physical entities (which all have standardized units).

DvQuantityType

Details

Derived from: DvQuantifiedType by extension

Abstract: False

Description: Quantified type representing specific 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.

DvRatioType

Details

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 forward slash ('/') 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.) "20/20" 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.

DvTemporalType

Details

Derived from: DvOrderedType by extension

Abstract: False

Description: Type defining the concept of date and time types. Must be constrained in PCMs 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 both maxOccurs and minOccurs to zero causes the element to be prohibited.

DvIntervalType

Details

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. In MLHIM, they are primarily used in defining reference ranges.

InvlType

Details

Derived from: n/a

Abstract: False

Description: In the CCD, the modeler creates two restrictions on this complexType. One for the 'lower' value and one for the 'upper' value. Both restrictions will have the same element choice and the value is 'fixed' on each representing the lower and upper value range boundary. The value may be set to NULL (unbounded) by using the xsi:nil='true' attribute. The maxOccurs and minOccurs attributes must be set to 1, in the CCD.

For more information on using this approach see these tips

InvlUnits

Details

Derived from: n/a

Abstract: False

Description: The units designation for an Interval is slightly different than other complexTypes. This complexType is composed of a units name and a URI because in a ReferenceRange parent there can be different units for different ranges. Example: A DvQuantity of temperature can have a range in degrees Fahrenheit and one in degrees Celsius. The derived complexType in the CCD has these values fixed by the modeler.

ReferenceRangeType

Details

Derived from: DvAnyType by extension

Abstract: False

Description: Defines a named range to be associated with any ORDERED datum. Each such range is sensitive to the context, e.g. sex, age, location, and any other factor which affects ranges. May be used to represent high, low, normal, therapeutic, dangerous, critical, etc. ranges that are constrained by an interval.

AuditType

Details

Derived from: n/a

Abstract: False

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

PartyType

Details

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.

AttestationType

Details

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 may be coded.

ParticipationType

Details

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.

ExceptionalValueType

Details

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 to allow for domain related reasons for errant or missing data.

NIType

Details

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

Details

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

Details

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

Details

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

Details

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

Details

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

Details

Derived from: ExceptionalValueType by restriction

Abstract: False

Description: Negative Infinity: Negative infinity of numbers

PINFType

Details

Derived from: ExceptionalValueType by restriction

Abstract: False

Description: Positive Infinity: Positive infinity of numbers

UNKType

Details

Derived from: ExceptionalValueType by restriction

Abstract: False

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

ASKRType

Details

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

Details

Derived from: ExceptionalValueType by restriction

Abstract: False

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

QSType

Details

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

Details

Derived from: ExceptionalValueType by restriction

Abstract: False

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

ASKUType

Details

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

Details

Derived from: ExceptionalValueType by restriction

Abstract: False

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

NAType

Details

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.

ItemType

Details

Derived from: n/a

Abstract: True

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

ClusterType

Details

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 can serve as the root component for arbitrarily complex structures.

DvAdapterType

Details

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.

EntryType

Details

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 important in healthcare for the de-identification of clinical information.

CareEntryType

Details

Derived from: EntryType by extension

Abstract: False

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

AdminEntryType

Details

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

Details

Derived from: EntryType by extension

Abstract: False

Description: Entry subtype for demographic information, i.e. name structures, roles, locations, etc. modeled 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.

CCDType

Details

Derived from: n/a

Abstract: False

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

RM simpleTypes

The reference implementation simpleType descriptions. These types do not have global element definitions. They are used to define other element types within the RM and are used as restrictions on a CCD.

MagnitudeStatus

Details

Derived from: xs:string

Abstract: False

Description: Optional status of magnitude with values:

```
equal : magnitude is a point value

less_than : value is less than the magnitude

greater_than : value is greater than the magnitude

less_than_or_equal : value is less_than_or_equal to the magnitude

greater_than_or_equal : value is greater_than_or_equal to the magnitude

approximate : value is the approximately the magnitude
```

These enumerations are used in they DvQuantifiedType subtypes.

TypeOfRatio

Details

Derived from: xs:string

Abstract: False

Description: Indicates semantic type of ratio.

- 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)

Element Groups

IntervalUnits

Used to state that if units are defined on a DvInterval based PCM then the units must have both a name and a URI.

Example CCDs

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

1.8 Linked Data Modeling

1.8.1 Background

Initial approaches to building ontologies for MLHIM used the XML Schema to OWL approach that has been published several times in academic literature. However, it was learned over these attempts that this is a single level mindset and approach. It simply does not express the richness of MLHIM. The results of those approaches represent the reference implementation of MLHIM and not the overall concept.

When we began MLHIM in 2009, we intended to use OWL as the basis on which to build the concepts. However, the Open World Assumption is in conflict with constraint based modeling at the MLHIM core.

As the Linked Data environment matures, graph based technologies are becomming mainstream for data discovery and analysis. By using both XML Schema 1.1 to model the structural and syntactic needs and RDF to model semantics we create the best of all worlds towards **computable semantic interoperability**.

In this document, when we talk about MLHIM we will use the term *MLHIM*. When we talk about modeling concepts in an area of interest we use the term *domain*. Though we are thinking primarily about the domain of healthcare and the information technology to support healthcare information exchange, MLHIM concepts may be applied to any domain of interest.

1.8.2 Syntactic Modeling

The complex nature of healthcare concepts and query needs requires a rigourous yet flexible structural approach to modeling. Using a multi-level approach built on a solid data model fulfills this need. The Reference Model consists of a minimum of components required to construct robust models. Desgined around the ubiquitous XML Schema data model provides a solid, standardized, implementable infrastructure. The Reference Model reference implementation is realized in XML Schema 1.1.

Components of the Reference Model can be assembled in virtually any structure need to express any level of granularity of healthcare or other domain concepts. These components are assembled in an XML Schema that contains only constraints (restrictions) of the Reference Model components. This constraint based approach guarantees that the structure and syntax of all domain concept models are valid against the Reference Model.

This guarantee means that it is easier to build persistence and query infrastructure that can accomodate unforseen domain concept models. This greatly reduces application complexity and maintenance.

1.8.3 Semantic Modeling

The MLHIM environment defines a few semantics to relate various components. Each Reference Model defines semantics for each component.

Each paticular domain concept model is based on one and only one Reference Model release. A domain expert determines the proper domain semantics for their domain concept model. In order to be MLHIM compliant there are required semantics relating the domain concept model to its parent Reference Model.

RDF provides an elegant and simple approach to expressing semantics. In addition, the variety of syntaxes available to express the *Subject, Predicate, Object* statements provides an excellent solution.

The Reference Model and domain concept models are authored in XML Schema and therefore the canonical RDF syntax of RDF/XML is used in these instances. The syntax used to represent these in implementations is left up to the systems developers. The specifications include the RDF/XML semantics in the XML Schema in order to facilitate easy exchange and governance of the Reference Model. For convienence there is also an extract of the semantics in RDF/XML and JSON-LD.

The Python utilities used to perform this extraction are included as examples of working with MLHIM models. There is also a utility for extracting semantics in RDF/XML and JSON-LD from domain concept models.

1.8.4 MLHIM2 Semantics

These are the entities defined in mlhim2.rdf

1.8.5 Top-Level

- MLHIM2
- RM
- ConceptModel
  - CoreCM
  - PluggableCM
- Symbol
  - CoreCS
  - PluggableCS
- CCDInstance
- DataInstance
  - DataInstanceValid
  - DataInstanceInvalid
  - DataInstanceError
- Exception

1.8.6 Other Properties

- isMLHIM2objprop
  - isCoreModelIn
  - isPluggableModelIn
  - isCoreSymbolOf
  - isPluggableSymbolOf
  - isSubSymbolIn
  - refersToSymbol

1.8.7 Datatype Properties

Some tools (e.g. Protégé) do not support the full range of XML Schema 1.1 datatypes directly. We defined these in mlhim2.rdf as well.

- duration
- yearMonthDuration

- dayTimeDuration
- gDay
- gMonth
- gYear
- gYearMonth
- gMonthDay

1.8.8 Annotation Properties

The most widely used (at this writing) metadata definitions come from the Dublin Core Metadata Initiative (DCMI) terms. However, the definitions for these do not meet the requirements for some reasoners. We have defined our own metadata properties and related them to other standards.

1.8.9 Context processing

Refer to the JSON-LD-API context processing specifications to understand how the mlhim2.jsonld, mlhim250.jsonld and the ccd jsonld work together.

For setting your JSON-LD processor for the correct location of context files, see this StackExchange discussion. The options for compliant processors is discussed in the JSON-LD Specs

1.8.10 Linked Data Tools

To reduce the learning curve for working with MLHIM data in your Linked Data environment we have included a few simple Python scripts to get you started. See the utils/README.md for details.

1.9 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, from wearable devices to enterprise EHRs.

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 explore 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 boundary-less information 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 described below. Keeping in mind that the granularity of MLHIM is much finer than these abstract architectures. However, this does serve as a good starting point.

1.9.1 TOGAF® 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.

1.9.2 TOGAF® Common System Architectures

The MLHIM CCDs are composed of multiple, reusable (pluggable) complexType restrictions of the reference model. These PCMs 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.

1.9.3 TOGAF® 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 (PCMs and CCDs).

1.9.4 TOGAF® Organization Specific Architectures

CCDs that are used in applications within one organization are in this category.

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 focus more towards information interoperability of applications, regardless of their platform and location. Here we can discover the importance of ubiquitous technology in providing interoperability. This chapter focuses on a specific enterprise whereas with MLHIM we work with a global healthcare information scenario. There are multiple terminologies and ontologies to use, multiple languages, cultures and geographies to consider and the data essentially has no expiration date. Therefore, *The Boundaryless Information Flow problem space* for us is significantly larger than the TOGAF® specifications cover. However, the above descriptions should add some context to the MLHIM concepts for those developers that are familiar with existing enterprise architectures. Always keep in mind that implementations are local, CCDs and data instances are global.

1.10 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.

1.10.1 The Core

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.

1.10.2 Key to Interoperability

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 infer, based on the reference model version, information about the data by using the names of elements nested within an element with the prefix 'pcs-'. Because these element names are unique to certain RM complexTypes. If there is a <dvcount-value> element then that data is from a DvCountType and *name* is in the preceding <label>.

1.10.3 Model Publication

We are not implying 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 or in a company. 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 and how to use XML Catalogs to reference any local copy.

1.10.4 Bottom Up vs. Top Down

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 and the Standards Development Organization (SDO) approach.

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. Several years of participation in and observation of these attempts to build consensus has led to the development of the Cavalini-Cook Theory - **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. The Cluster complexType allows for any data structure [2] to be built to match implementation needs. There is no inherent idea of a specific application such as an Electronic Health Record (EHR), Electronic Medical Record (EMR), etc. in the RM although the MLHIM specifications can also be adopted for the development of these types of applications. This approach provides an opportunity for development of small, purpose specific apps such as mobile or portable device software as well.

1.10.5 How Many CCDs?

In MLHIM, the bottom-up approach makes room for dozens, hundreds or even 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. the top-down approach that requires 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) prefixed with the string, 'ccd-'.

[2] Used here to mean; trees, lists, tables, etc.

CCDs are assembled out of pluggable concept models (PCMs) so that modelers can use granular definitions to create any size application model or models, as 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. Reuse of existing PCMs in multiple CCDs and reusing CCDs across multiple applications makes data sharing and analysis easier. However, given that the semantics are in the CCD, data consumers can decide if the data fits their needs or how to include certain components (PCM based data) from multiple CCDs.

Over time, **the cream will rise to the top** and the most useful CCDs will be used most often.

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.

1.10.6 A Valid CCD Must:

- Be a valid XML Schema 1.1 schema as determined by widely available parser/validators such as Xerces or Saxon
- Consist of complexTypes that only use the *restriction* element of complexTypes with a *base* attribute from the associated reference model
- use Type 4 UUIDs for complexType names, with the prefix of, 'pcm-'. Example [3]

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

- publish a global element for each complexType where a substitutionGroup is required [4]. The element **MUST** be defined using the same UUID as the complexType with the 'pcm-' prefix replaced with 'pcs-'.
- Include the reference model schema from www.mlhim.org using the appropriately defined namespace. Example for release 2.5.0 and later releases, MLHIM uses the namespace http://www.mlhim.org/ns/mlhim2/ with the standard prefix of mlhim2 [5]

```
<xs:element name='pcs-8c177dbd-c25e-4908-bffa-cdcb5c0e3888' substitutionGroup='mlhim2:DvAdapter-
```

- use the correct substitution group(s) as in the example above
- define the required namespaces used in the CCD as in Figure 1.
- define the minimum DCMI metadata items as shown in Figure 2.

[3] The language attribute is optional.

[4] Substitution groups are required where the base type allows multiple elements and where the base type allows an abstract element.

[5] Some previous releases had a specific namespace for the RM and each CCD. This was changed to a single namespace for all of MLHIM 2.x versions to improve query and processing interoperability.

```
<xs:schema
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:mlhim2="http://www.mlhim.org/ns/mlhim2/"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:sawsdlrdf="http://www.w3.org/ns/sawsdl#"
  xmlns:sch="http://purl.oclc.org/dsdl/schematron"
  xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:skos="http://www.w3.org/2004/02/skos/core#"
  targetNamespace="http://www.mlhim.org/ns/mlhim2/"
  vc:minVersion="1.1" xml:lang="en-US">
```

Figure 1.

```
<!-- Metadata -->
  <xs:annotation><xs:appinfo><rdf:RDF><rdf:Description
    rdf:about="http://www.ccdgen.com/ccdlib/ccd-b98cbabb-e4b5-4bb5-93a7-16dd52cebcf4.xsd">
    <dc:title>CareEntry CCD</dc:title>
    <dc:creator>Tim Cook</dc:creator>
    <dc:subject>CareEntry CCD  - Testing Only</dc:subject>
    <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-b98cbabb-e4b5-4bb5-93a7-16dd52cebcf4</dc:identifier>
    <dc:description>Testing CCD-Gen Implementation of the MLHIM RM functionality.</dc:description>
    <dc:publisher>MLHIM LAB, UERJ</dc:publisher>
    <dc:date>2015-08-13 19:17:47.798671+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.

1.10.7 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 or include any XML Schema document other than its parent reference model schema.

1.10.8 CCD Functionality

Structure

A CCD is just an XML Schema. It uses the xs:include element to reference the RM schema from the MLHIM website. For performance a local copy should be used via an XML Catalog The design of CCDs allows us to separate the structure from the domain semantics of a complexType. This is key in having a small RM that represents structural components that provide a well defined data query platform which is essential for analytics and decision support.

Prior to XML Schema 1.1, other languages (such as Schematron) were *needed* to provide for complex validation scenarios. The xs:assert element now takes care of those issues. This in addition to gaining additional data model types and the ability to use multiple substitutionGroups is why we specify XML Schema 1.1 as a requirement for CCDs.

Schematron may still be useful for defining business rules in your implementation. But these definitions are implementation specific and do not apply to the MLHIM semantic interoperability goals. Experience shows that these implementation details *leaked* into the data structure definition creates a barrier to interoperability.

The use of UUIDs has been controversial and is a perceived barrier by some people. In reality though they are what allows MLHIM to be such a simple, yet powerful solution. This is how we separate the structure and domain semantics. By using UUIDs for the complexType and element names we build a structure that has only *structural semantics*. As an example take a look at HL7 CDA or FHIR schemas or NIEM schemas. Notice how they mix domain semantics into the names of complexTypes and elements. This creates a nasty chain of optional domain elements because you cannot know a priori what is going to be needed where. There is an attempt to use attributes to provide some uniqueness to types of entries. But this has grow out of control to a point where those schemas are very complex. NIEM attempted to solve the problem by using a multi-level model approach. But then *specified* that domain element names **MUST** be terms from the Oxford English Dictionary. So that limits its usefulness to 5.4% of the global population. To be fair, NIEM is designed and named to be used in the US. But that is a bit of a short-sighted development approach considering the global world we live in today.

In designing MLHIM we had the advantage of being able to analyze HL7 v3.x, openEHR, ISO 13606 and other interoperability attempts and use these as lessons learned points. NIEM was started about the same time as MLHIM.

We realized that this mix of structure and domain semantics was a key problem in the complexity of the models. In openEHR the focus is specifically about EHR systems but it introduced multi-level modeling as a constraint based approach. MLHIM takes the constraint based, multi-level approach along with the data package view. We say data package because you may view a data instance as a message or as a document *or* as a component of a message or a document. A *MLHIM data instance* is just that. They can be very small or very large and they can be combined into documents or messages or standalone; depending upon the implementation needs. This is part of the *multi-level paradigm*.

The need for multiple substitutionGroups arises because, one PCM may be reused in multiple places in a CCD. For example a DvLinkType based PCM may be reused as a link in an EntryType as well as used in a ClusterType based PCM. In this case the since *element* of type *complexType* must be defined as substituting for the RM elements *DvLink* and *DvAdapter-value* elements from the RM. Example:

```
<xs:element name='pcs-a05e8d88-a6dc-43d5-b1b8-723cdc9bf680' substitutionGroup="mlhim2:DvLink mlhim2:D
```

Semantics

With the reusability and structural simplicity out of the way we can now discuss the issue of **what does the data mean?**

NOTE: This section is in active transition from RDF Semantics to OWL2 DL using the RL Profile.

If you are not familiar with RDF you may want to read more at LinkedDataTools or directly from the W3C Specifications.

The world of data exchange is composed of two primary players; *data providers* and *data consumers*. Whether those two parties are people or software applications they require this knowledge to turn *data* into useful *information*. We discussed approaches to transferring this information in the **Semantic Models vs. Semantic Markup** section of *Modeling Concepts*.

Adding the semantics to the model allows all of the meaning of the data to be in one known location. Each data instance has a pointer to its parent CCD. Example:

```
xsi:schemaLocation='http://www.mlhim.org/ns/mlhim2/ http://www.ccdgen.com/ccdlib/ccd-00605c3e-cd14-49
```

this example from a data instance says that the schema in the namespace *http://www.mlhim.org/ns/mlhim2/* is located on the CCD-Gen and is named *ccd-00605c3e-cd14-492e-9891-6ad3ad26230e.xsd*. Typically CCDs are located locally and an XML Catalog is used to resolve these locations.

The first part of the semantics describes the model itself. This is accomplished using the DCMI metadata elements. See the example above in Figure 2.

Taking a *simplistic* example CCD (the sequence of appearance of the complexTypes is not important) we can see a PCM with a DvLinkType restriction:

```
<xs:complexType name='pcm-a05e8d88-a6dc-43d5-b1b8-723cdc9bf680' xml:lang='en-US'>
<xs:annotation>
  <xs:documentation>
    This is a test DvLink used for an example.
  </xs:documentation>
  <xs:appinfo>
    <rdf:Description rdf:about='mlhim2:pcm-a05e8d88-a6dc-43d5-b1b8-723cdc9bf680'>
      <rdfs:subClassOf rdf:resource='&mlhim2;DvLinkType'/>
      <rdfs:isDefinedBy rdf:resource='http://www.mlhim.org/generic_PCMs'/>
      <rdfs:label>Test DvLink</rdfs:label>
    </rdf:Description>
  </xs:appinfo>
</xs:annotation>
<xs:complexContent>
  <xs:restriction base='mlhim2:DvLinkType'>
    <xs:sequence>
      <xs:element maxOccurs='1' minOccurs='1' name='label' type='xs:string' fixed="Test DvLink"/>

    ...
</xs:complexType>
```

Notice that inside the xs:annotation there are two child elements; xs:documentation and xs:appinfo. The xs:documentation element has a free text, human readable description of the purpose of the PCM. The xs:annotation element has a child element rdf:Description this element has an rdf:about attribute with a value of the namespace and the complexType name. This forms the *Subject* component of the RDF statements to follow.

The first child of rdf:Description is a rdfs:subClassOf element. This element name is the *Predicate* component of the first RDF statement. This element has an attribute of rdf:resource and a URI of mlhim2:DvLinkType which forms the *Object* component of this RDF statement.

The second child of rdf:Description is a rdfs:isDefinedBy element. This element name is the *Predicate* component of the second RDF statement about the PCM. The rdf:resource attribute points to a resource on the MLHIM website. Give it a try. It is just a simple plain text document used as a resource for these examples. Note that it is not a requirement that all URI resources be resolvable URLs. But we think it is a good idea that they are whenever possible.

The third child of rdf:Description is a rdfs:label This element defines a fixed text label to this PCM. So the *Predicate* is rdfs:label and the *Object* is the string "Test DvLink".

So now we have three statements about the unique subject called *mlhim2:pcm-a05e8d88-a6dc-43d5-b1b8-723cdc9bf680*. We know it is a subtype of the MLHIM DvLinkType that is defined in the MLHIM Reference Model schema that is included (via xs:include) in this CCD. We can also find a definition of this PCM in the resource at http://www.mlhim.org/generic_PCMs.

So now we have some machine processable semantics as well as some documentation. All available from the model. Also note that there is the *label* element. When a modeler defines a PCM they give it a fixed name as a human readable string. This is included in the data instance and even though the XML element is a UUID, this readable text

is immediately below it and describes what the modeler defined for the name. The rdfs:label and the label **SHOULD** be the same string.

Example from the instance data:

```
<mlhim2:pcs-c05e8d88-a6dc-43d5-b1b8-723cdc9bf680>
  <label>Test DvLink</label>
  ...
</mlhim2:pcs-c05e8d88-a6dc-43d5-b1b8-723cdc9bf680>
```

The next section describes how all of this can be used in an operational setting.

MLHIM in Operation

We have a Reference Model, a Concept Constraint Definition and some data instances; all in XML. How does all of that fit together, especially since I use JSON with my REST Services and Turtle for my RDF semantics syntax?

Glad you asked ....

Remember that the XML and RDF/XML syntaxes are used because they are robust. They are the canonical definitions for the models and the data because the tools are available for validating the syntax and the semantics.

Because MLHIM XML data has a *very specific structure* it is quite easy to perform lossless conversion to and from JSON. So you can send and receive JSON data. The points in your data chain that need require validated data can be converted back to XML for validation.

So (a snippet) like this:

```
  <mlhim2:pcs-d4079097-c68b-4c99-9a5e-b85628d55897>
<!-- Party -->
<party-name>A. Sample Name</party-name>
<!-- PI external-ref -->
<mlhim2:pcs-ab51a8c0-ba5c-4053-8201-ae29c1a534bb>
  <!-- DvURI -->
  <label>External Reference for Party</label>
  <!-- Use any subtype of ExceptionalValue here when a value is missing-->
  <valid-time-begin>2006-04-15T04:18:21Z</valid-time-begin>
  <valid-time-end>1981-12-10T19:35:00Z</valid-time-end>
  <DvURI-dv>http://www.ccdgen.com</DvURI-dv>
  <relation>Party Associated with the record</relation>
</mlhim2:pcs-ab51a8c0-ba5c-4053-8201-ae29c1a534bb>
```

can be converted to:

```
    },
"mlhim2:pcs-d4079097-c68b-4c99-9a5e-b85628d55897": {
  "party-name": "A. Sample Name",
  "mlhim2:pcs-ab51a8c0-ba5c-4053-8201-ae29c1a534bb": {
    "label": "External Reference for Party",
    "valid-time-begin": "2006-04-15T04:18:21Z",
    "valid-time-end": "1981-12-10T19:35:00Z",
    "DvURI-dv": "http://www.ccdgen.com",
    "relation": "Party Associated with the record"
  },
```

and back again. All depending upon the needs of your information flow.

Two of the MLHIM projects on GitHub demonstrate combining the model semantics with the data instances to create a Graph and storing it in a Triple Store. The connections can then be visualized using open source tools and / or queried using SPARQL.

For deeper details on using MLHIM in various scenarios you should refer to the GitHub repository specifically this project as well as the MXIC demo and the MAPLE demo These all use older versions of MLHIM but the concepts are the same for any 2.x version.

Two projects that may be of particular interest is adding semantics to FHIR models and adding semantics to HL7v2 messages

1.11 MLHIM Data Analysis

The real 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 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 programming language is specifically designed for data analysis and supports XML data via an XML package using the commonly used libxml2 library used by most of the other languages. Libxml2 includes complete XPath, Xpointer and Xinclude implementations. There is also a companion library called libxslt based on libxml2 that is used to provide support for XML transformations

The CCD-Gen provides example R code, automatically generated for each CCD as an R project for use in R Studio or at the R command line.

1.11.1 Semantic Models vs. Semantic Markup

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 terms, the Data Catalog Vocabulary, Schema.org 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 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 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 bytes; every day**.

Regardless 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 immutable. In other words, this is what the modeler intended for the data to mean at this time and in this context. The model 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 solution. 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.

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 or in the ability to even markup the data at all, using various syntaxes. In MLHIM this is solved, as a result of the 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, misrepresented or misinterpreted.

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 MXIC project 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. However, we know from previous experience that the W3C process is a slow one.

In a few years, there may be widespread adoption and tools for validation of RDF syntaxes and/or the various levels of 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 and RDF mix of technologies.

1.11.2 Support for Legacy Systems

MLHIM is designed to provide semantic interoeperability for future systems. Based on small, granular models taht can be well defined. However, it is quite capable of supporting the transition of legacy data as well. Several examples are available from the MLHIM GitHub site.

A capability often over looked is the ability to create models for CSV data in order to build Linked Data graphs. An often perplexing problem is how to support validation of data when that data may have come from a printed vocabulary or a terminology server. An example of this is in the TB demo where the Brazilian Death Certificate and Hospital Discharge data are modeled from CSV files. Of particular note are the DvStringType models that constrain the ICD 10 codes via an xs:assert for validation in the XML. For Linked Data a vocabulary root is provided via mlhim2:vocabRoot predicate and the resource is the URL to the WHO online repository. In cases where there is no online repsository (such as a published PDF) and the number of enumerations would be to large to be practical as constraints, we suggest creating a URI prefix for the link to the PDF and use a # plus the values found in the data.

1.12 Concept Constraint Definition Generator (CCD-Gen)

The CCD-Gen is not part of the specifications. However, since it is a primary tool for authoring CCDs we include this chapter.

1.12.1 Registration & Access

The CCD-Gen does require free registration to gain access to be able to view and download current CCDs. Note that CCDs are also available on GitHub.

Access to authoring your own PCMs & CCDs requires posting a request in the MLHIM Google+ Community. See the Support page for links.

1.12.2 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 complexType (PCM) definitions from existing

PCMs or ones that you design yourself. You assemble the pieces into the concept definition you need. By re-using PCMs you improve the data exchange and analysis capability.

Many of the PCMs 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 many others. Some HL7 FHIR® models have been translated to PCMs. More are planned and you can contribute to the open content effort. The Siscolo application has been modeled and example of translating a controlled vocabulary using ICD-10 in an XML-DB is on GitHub.

The CCD is an XML Schema that is compliant with the W3C XML Schema 1.1 standards. It is valid against one MLHIM Reference Model release. However, many of the PCMs 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.*

The CCD-Gen has a library of some of the CCDs created using this tool. Because of the governance approach to CCDs there is no need for a centralized repository. It is helpful though to have some place for potential users to review published examples. The CCD-Gen Library page displays the CCDs in a table and an included XSLT renders the details directly from the CCD (click on the Title link) in a familiar data dictionary format. This is an example of how applications can extract this information from the CCD for user help pages and tooltips and even decision support.

CCDs created by the CCD-Gen are also commited to a GitHub repository Each reference model version has a separate branch in the repository. See the README for instructions on searching the CCD Library on GitHub.

1.12.3 XML Data Instances

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

1.12.4 JSON Data Instances

The CCD-Gen executes an XML to JSON conversion to create the example JSON data instance. The namespace entries are treated as data so that they are maintained for conversion back to XML.

1.12.5 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 a priori 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 the parent complexType 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 PCM. This provides the developer with the same context that the modeler used when defining the PCM.

1.12.6 R Code

In conjunction with each CCD, the CCD-Gen creates source code for use with the R statistical programming environment [6]. Specifically the code is created in a project suitable for use in R Studio[#f2]_, a package development and management tool for R.

The R code is package complete and compliant for distribution via the The Comprehensive R Archive Network[#f3]_ . 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 PCM level and will return a dataframe from all matching instances of a PCM. For example you may search across a complete repository of XML instances and receive a dataframe containing all occurrences of a specific PCM in all CCDs. 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. **Customizing these basic routines allows the analyst to apply any of the thousands of algorithms available as R packages 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.

1.13 Eclipse Ecore

There is now an Eclipse Ecore model included in the documentation. If people find it useful please expand on it. I have very little experience with Eclipse.

I could really use some help making this more useful. Maybe you can even generate a Java code base from it? Maybe you can edit CCD models in it? Maybe you can generate Python code? I have no idea ...

The things missing are:

1. DvInterval upper and lower are not typed checked because the Eclipse XML Tools can't handle asserts.
2. The extra durations from XML Schema 1.1 are not present in DvTemporal for the same reason.

Everything should be there.

If you can help with using this Ecore model to 'do something' please let me know.

[6] http://www.r-project.org/

1.14 Glossary

1.14.1 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.

1.14.2 Reference Model (RM)

The RM is a small set of structural concept definitions that allow for building arbitrarily complex models without introducing domain semantics into the structure. Domain concepts are modeled as *restrictions* on these RM concepts. Then RM therefore defines a common set of concepts that allow for query and knowledge discovery across data without prior knowledge of the actual content. See CCD below.

1.14.3 Pluggable Concept Model (PCM)

The name comes from the fact that it is a complete XML Schema complexType that represents a simple concept 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 PCM names begin with the prefix 'pcm-' followed by the UUID. This also facilitates the association with public element names in instances since they reuse the UUID but are prefixed with 'pcs-' in place of the 'pcm-'. The use of a UUID allows the constraint on a reference model type to be reused many times in a CCD with different parameters such as enumeration constraints. The semantics for a concept modeled as a PCM is represented using Semantic Web technologies. The PCM name is the subject in each of the *Subject, Predicate, Object* RDF statements.

1.14.4 Concept Constraint Definition (CCD)

This is a domain concept data model that 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 complexTypes using the *restriction element* with its *base attribute* set to the appropriate RM complexType. CCDs are immutable, once published they are never edited because once data is published in conformance with a CCD this is where the sharable semantics are located. CCDs are uniquely identified by the prefix 'ccd-' followed by a Type 4 UUID. They are valid against one version of the MLHIM Reference Model XML Schema. **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*. The semantics for a concept modeled as a CCD is represented using Semantic Web technologies. The CCD identifier is the subject in each of the *Subject, Predicate, Object* RDF statements.

1.15 Support, Publications & Social Media

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

1.15.1 Peer Reviewed

See the Documents area of the MLHIM website for the current listing.

1.15.2 Specifications Wiki

Read various pieces of content not in the documentation.

1.15.3 CCD Library

Examples of CCDs

1.15.4 Presentations

SlideShare

1.15.5 Videos

YouTube

1.15.6 Primary disscussion venue

MLHIM Community

1.15.7 Issues Reporting

Please report all issues here.

1.15.8 FaceBook

Just announcements and a few questions answered on our FB page.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search