Proposal for HL7 Version 3 Data Types

Gunther Schadow
Regenstrief Institute for Health Care, Indianapolis

WORKING DRAFT

Id: v3dtm.tex,v 1.5 1998/09/29 01:45:48 schadow Exp

Preface

This document is a mess and I apologize for it. The reason why I publish it anyway is because I want the project team on version 3 data types to get up and running. Originally I intended to come to San Diego with a complete document, but the project was too big to accomplish just a few weeks before the meeting. So I fell back to producing slides and while doing so, I came to new ideas of how to break down the problem into parts.

I want to use this document as a basis for working through the many issues involved in data types. The table of contents sets forth an agenda. Except from an introduction there is only one section written, section 2.1 on text. Some lengthy but incomplete thoughts on real world concepts are found in section 2.2, and the rest is empty yet. As a compensation to the lack of text, this document contains the slides that I presented in San Diego. These provide the keywords along which we can tackle the problem space.

As we discuss these issues, more and more of the messy text should become straight and the controversial text should become reflecting our consensus. As we move on, the slides are going to be replaced by text and useful drawings if we want. The list of authors will expand as input from other people gets into the document. Although lots of thoughts originate in a dialog with Mark Tucker, I did not list him now, because there are possibly many things in here to which he would object.

This document is going to be collaborative work. I will edit the document as people make suggestions and as the phone conferences move forward. The document will be distributed as PDF, which is a platform independent format. My Web server `http://aurora.rg.iupui.edu/v3dt` will be the distribution site. The small line "Id: ..." in the title tells you the revision number and date. I use the Revision Control System (RCS) so that a complete history of our changes is maintained. We thus need not be afraid from removing large chunks from the text as we can recover those later, if we want to.

Contributions are welcome in various forms. If you want to comment, you can do this simply by faxing the respective pages with your scribbeling to the number (317) 630 6962, please write my (or Mark's) name in the upper left corner or otherwhise the fax is falsely routed to Clem's desk. You can also cut and paste a couple of paragraphs or more out of the PDF reader and modify those. Do not fight with formatting or a couple of screwed letters — the contents is what counts most. If you want to contribute chunks of text, please do so by sending raw ASCII e-mail. You can just post to the list `hl7-cq@list.mc.duke.edu`. If you absolutely think that you have to contribute marked up text, by all means do so in HTML. Please use Word or RTF texts as the absolute last resort. If you want to contribute drawings, the easiest thing is to scratch something on paper and fax it. If you want to share to the list, use whatever tool you want and post in a format that others can read.

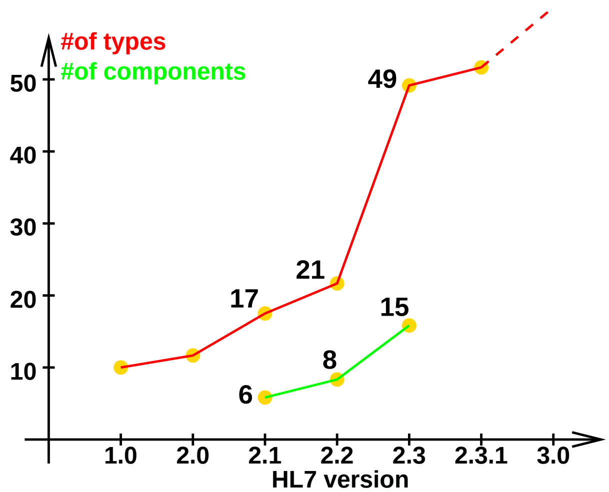
We will start our e-mail discussions on `hl7-cq@list.mc.duke.edu`. If the traffic becomes high, or if there is too much distraction from parallel threads, I will set up an e-mail list on `hl7-v3dt@aurora.rg.iupui.edu`. Now please scan through this document and let your thoughts flow. I am looking forward to working with you on this important part of HL7 version 3.

Contents

1 Introduction

Redesign Goals

- **Simplification**
  How can we contain the explosion of data types in number and complexity?
- **Rationalization**
  How can we keep track of the meaning?
- **Maintainability**
  How can we manage consistency and coherence in the set of types?



1.1 Requirements

Requirements

- **Semantics first** — Communication is to exchange meaning through signals - unambiguously.
- **Usefulness and Reuseability** — The basic data types should be equally useful to all technical comittees.
- **Stability** — Because one data type is used many times, the impact of changes is extremely difficult to control.
- **Coherence** — There should not be two competing data types for one given use case. Relationship between types should be clear.
- **Minimality** — There should be just as many data types as there are independent semantic concepts to support.
- **Completeness a priori** — We should aim for coverage to every conceivable logical extent - anticipate the future.
- **Simplicity** — We want to produce an easy to implement standard.

1.1.1 Semantics first

Data types are the basic building blocks of information exchanged in messages. Information is exchanged in the form of signals which are ordered according to lexical and syntactical rules. These signals are exchanged to convey a meaning (semantics) and to eventually serve a purpose (pragmatics). Therefore, data types must have a precisely defined semantics that is unambiguously related to their syntax (including the rules for building lexemes).

1.1.2 Usefulness and reuseability

The basic set of data types must be equally useful for all HL7 technical committees. This means, the data types must be meaningful enough so that the technical committees can use them directly as the data types for encoding the attributes of their information model classes. It also means that the basic set of data types must be reuseable for many purposes and should not be too highly specialized. This does not preclude a highly specialized data type to be defined by a technical committee that uses it.

1.1.3 Coherence

The set of all data types should be coherent. There should not be two or more competing data types for a certain use case. The relationships between the data types should be well defined. This means that data types should be organized similarly to the organization of domain information models (DIM) in the reference information model (RIM). The RIM and RIM harmonizations make sure that the DIM classes are in a close relationship and that there are no competing alternatives to express the same information in different ways.

1.1.4 Minimality

From the coherence reqirement it follows that the number of data types in the set should be minimal. There should be just as much data types as there are independent basic semantic concepts to support. The lower boundary of minimality is that each data type should have a well defined semantics on a level that is relevant to the application domain of HL7. For example, we could have only one data type "string of bits", but bits do not have a general relevant meaning on the application level of HL7.

1.1.5 Stability

It follows from the reusebalility requirement that every basic data type will be used my many classes and attributes of almost every technical committee. It becomes extremely difficult to coordinate changes to the data types and to estimate the effect that those changes would have on the many different areas in which the data types are used. Therefore the set of data types must be designed for high stability.

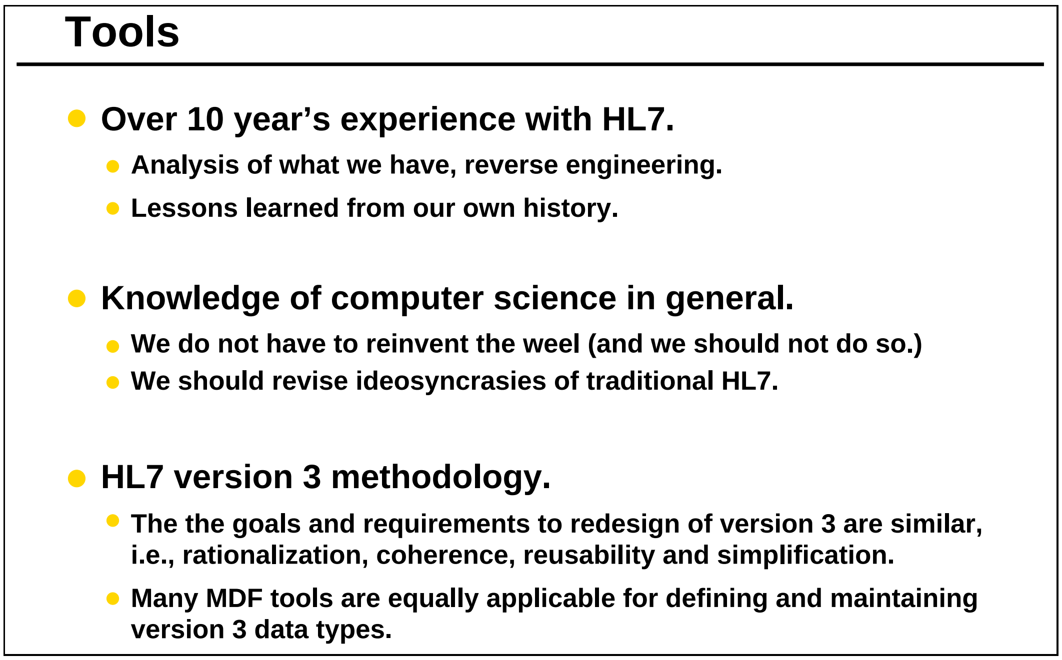
1.1.6 Completeness

Usefulness, reuseablity, coherence and stability can be achieved by aiming for maximal completeness a priori. This means that the data types of each basic semantic area cover that area to every logical extent conceivable by the time of design. Conversely completeness a posteriori would only make sure that every current concrete use is covered by the design. Stability can only be achieved by aiming for complete coverage of every conceivable current and future use case.

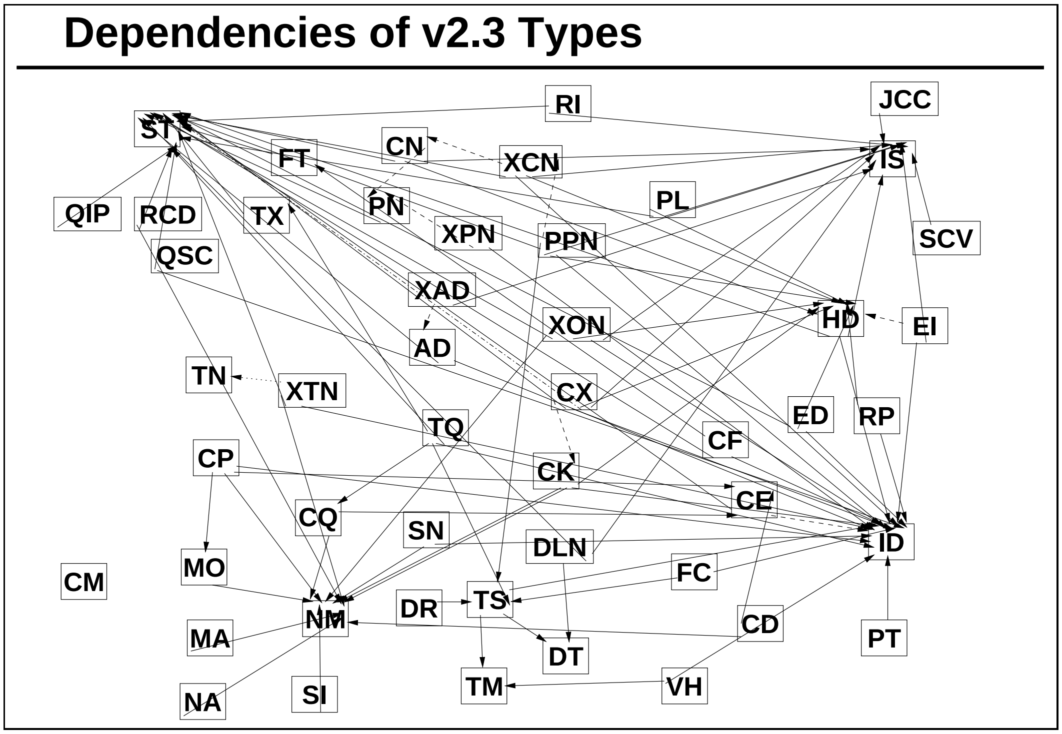
1.1.7 Simplicity

The data types should be as simple as possible to ease implementation and

1.2 Material and Methods



1.2.1 Prior Knowledge and Experience



Dependency Graph - lessons learned

- There is a hierarchy of types:
  - inbound arrows indicate importance.
  - outbound arrows indicate composition.
- Some types are basic ...
  - ST, ID, IS, HD, NM are heavily reused by other types.
- ... some are advanced.
  - CP, VH, RCD, RI, and many others are not reused by other types.
- Some types are primitive ...
  - TX, DT, TM, TN, etc. do not have components.
- ... some are composite.
  - PN, AD, EI, do have components.

History - many lessons to learn from

- Imprecision and ranges: TS-DT, DR, SN.
- Historical dimension: FC (effective date), DLN (expiry.)
- Type codes: PN-XPN, AD, TN-XTN.
- Different kinds of identifiers: ID/IS, CE, EI, HD, CK, DLN.
- Assigning authority and type code: CK, CX, CN, XCN.
- New uses: TN-XTN (e-mail.)
- Multimedia: ED, RP.
- Correcting past misdesigns: TN-XTN, CM-*, AD-XAD.
- Coping with incoherence: TX-CE, FT-CF.
- Minor distinctions: ID-IS, CE-LCE.
- Avoiding new segments: XON, XCN, PPL, PL, TQ, CD.

Lesson #1 from history

Do not design a type system based only on currently evident use cases.

- **Aim for completeness.**
- **Aim for consistency.**
- **Aim for generality.**
- **Anticipate the future!**

Work plan

- **Define a concise basic type system.**
  - **Rationalize: explain your decision.**
  - **Watch out for principles, classify, divide and conquer.**
  - **Seek exhaustive coverage of logical domains.**
- **Watch out for orthogonal (i.e., independent) dimensions of data.**
  - **Collections (List, Set, Bag, etc.) as generic data types.**
  - **Incomplete information, update semantics, uncertainty, history.**
- **Divide the plethora of advanced types into maintainable sections, delegate maintenance.**
  - **Reuse MDF approach: RIM, DIM, stewardship, harmonization, MDF.**

For our design of HL7 data types we can build on two kinds of prior knowlege and experience. There is more than ten years of experience with data types in version 2 of HL7 and there is more than 40 years of experience with data types in general computer science. In this proposal we will try to maximize leverage of these two rich sources of knowledge.

1.2.2 Reuse of MDF Methodology

To further improve the overall coherence of the design of HL7 as a whole, we will try to reuse the modeling methodology and tools of HL7 version 3 as described in the message development framework (MDF). We will formulate the relationships between data types as object oriented models in the unified modeling language (UML). We will have one common data type model (CDTM) that is divided into subject areas or domains (domain data type models, DDTM).

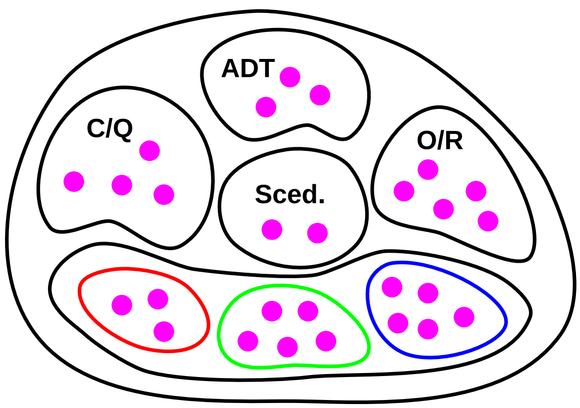
1.2.3 CDTM and DDTM

The CDTM plays the role that the RIM plays in HL7 information modeling, i.e. to provide proof for coherence of the overall type model. The DDTMs are used to focus the definition of domains of data types and thus resemble the DIMs. However, the requirement for reuseability and coherence prohibit any direct relationship between a DIM and a DDTM; we want most data types to be generally used by all technical committees of HL7 instead of every technical committee defining their own ideosyncratic set of data types in isolation. Thus, the partitioning of the set of data types in domains is not the same as the partitioning of the set of information model classes in domains. The domains of the information model are application domains. Conversely, the domains of the data type model are fundamental logical domains.

1.2.4 Control query and other technical committees or BDTM and EDTMs.

Maintenance

- **CQ maintains basic types.**
- **Release advanced types to stewardship of Technical Committees that use those types.**
- **CQ keeps responsibility for coherence.**



The control/query committee offers a basic data type model (BDTM) as a service for all other technical comitees. Control/query does, however, not try to rule over the other comitees. The BDTM will be build towards maximal use for all technical comittees. This means, control/query appreciates suggestions and tries to address the concrete needs of the other comittees. However, control/query will be responsible for the coherence and stability of the BDTM. Contentions will be resolved in harmoizations similar to RIM harmonizations.

Control/query will not prevent technical committees to develop their own extensions to the basic data types (EDTM) as needed. Our basic data type model shall be enabling not restricing. In HL7 version 2 we had some highly specialized data types for timing and quantity (TQ) and waveform information. These data types are not fully addressed in this proposal. It is up to the technical committees who uses these types to define and maintain them as they see fit. However, we will make suggestions as to how these types can be defined on the new basis of version 3 data types.

The data type(s) for coded information does not automatically fall under the same category of highly specialized data types that would be maintainable seperately by one technical comittee. Coded values are used by all technical committees as often as numeric types. Therefore, data types for coded values must be considered fundamental, are subject to the requirements of reuseability and coherence, and are defined as part of the BDTM not an EDTM. However, we appreciates and seeks close guidance in the recent work that has been done in this field by the technical comittee on vocabulary. It is also possible for a distinct variant of a data type for coded values to be used in only a few fields by a few technical committees. Such a type could be maintained in an EDTM.

1.2.5 Abstract properties of data and abstract data types

There are common properties of all data that is independent from the specifics of the data types and can therefore be specified in an abstract manner. These issues fall into two categories: (1) fundamental properties of all data and (2) aggregations of data types.

Fundamental properties of all data include incomplete information, uncertainty and update semantics. To give account for these fundamental properties is very imortant not only to clinical medicine but whenever data about the real world is collected honestly. We are honest about data when we do not want to pretend certainty where we are in fact unsure, not pretend completeness where we know we are missing some information. These fundamental properties are elaborated in a fundamental data type model (FDTM).

There are different ways in which we can aggregate data, unordered and ordered sets of fixed of variable cardinality with or without multiple occurance of the same object. These collection data types are also called abstract or *generic* data types. We will define those in the generic data type model (GDTM).

1.2.6 From semantics to implementable specifications (ITS)

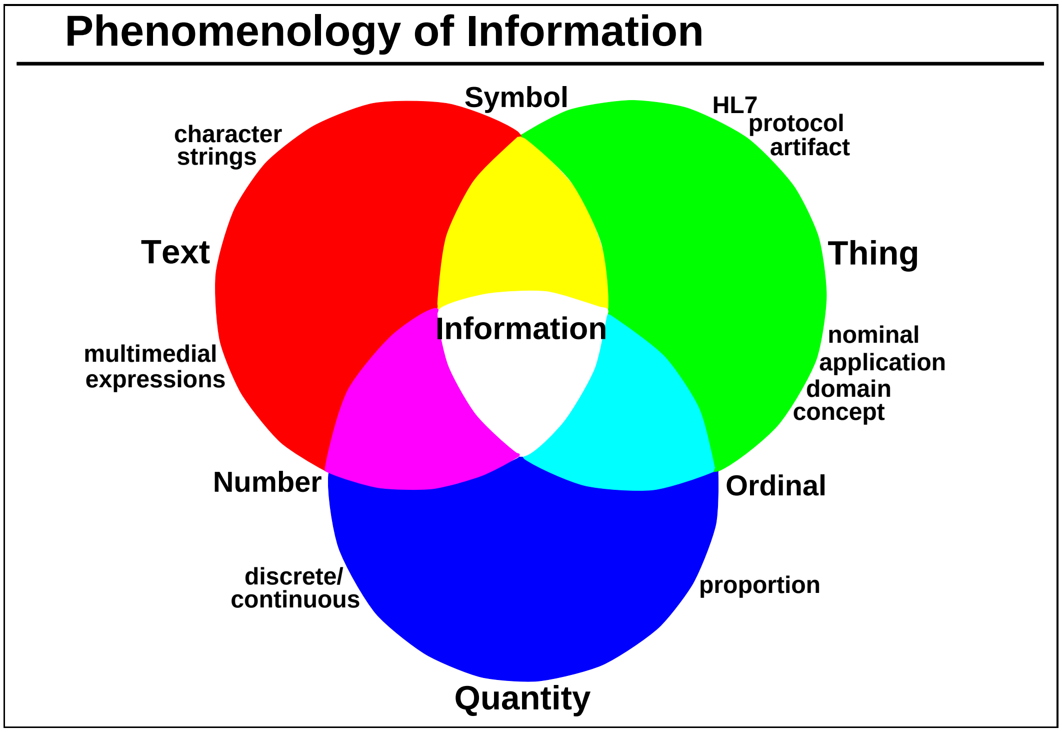
For every domain data type model we will give account for the semantic properties aiming for complete coverage a priori. We will outline the version 2 approach and what we can learn from computer science in general. We will then define data types by their semantic properties. From the semantic structure of the field, we defere a description of the abstract syntax of the data types. This abstract syntax serves as the basis for the mapping to different implementable technology specifications (ITS).

We will sometimes show examples and caveats as to how these abstract types can be incarnated in different ITSs and what the problems might be. The ITSs that we look at include XML, object broker technologies, and the simple encoding rules that may eventually replace the traditional HL7 encoding rules as the default encoding for HL7 messages. We do, however not aim to completely specify those ITS mappings. And we will seek for maximal independence from any specific ITS.

1.2.7 Literals

For many ITS it may be useful to have concrete lexical rules for specifying literals. Literals can be used to specify data type instances in character oriented encoding rules. It is useful in our oppinion to have a single standardized form of literals to be used by different ITSs. Literals are not only useful in inter-system messaging but also in discussions about the design of HL7 messaging, as we need them to write down example messages. The guideline for the specification of literals is to be concise and easily understandable by humans.

1.3 Roadmap


Phenomenology of Information
Symbol
character
strings
HL7
protocol
artifact
Text
Thing
Information
multimedial
expressions
nominal
application
domain
concept
Number
Ordinal
discrete/
continuous
proportion
Quantity

Summary of basic types

Text	string	ST
	text	TX, FT, ED, (HTML, ...)
Object	real world concept	CE, CF, ID, IS
	real world instance	
	person name	PN, XPN
	organization name	XON, HD
	id number	CK, CX, DLN
	general location	PL
	residential address	AD, XAD
	technical concept	ID, IS
	technical instance	TN, XTN, EI, HD, RP
Quantity	integer	NM
	rational	SN
	float	NM
	measurement	CQ, MO*
	point in calendar	TS, DT
	calendar modulo	TM, ID (VH)

Summary of generic types

list	NA, MA
set	QIP*
bag	
interval	SN, DR, RI
uncertainty	
incompleteness	(all, ‘‘not present’’)
update semantics	(all, ‘‘null’’)
history	FC

- All ‘‘repeated fields’’ will be one of the collections.
- Remaining types:
  - merge CN and XCN
  - decompose PT (processing type) into two technical concepts
  - upgrade/merge CN, XCN, PPN, and TQ into RIM classes
  - manage QSC, QIP, RCD, SCV elsewhere.

Gutman (1944) and Stevens (1953) identified four categories of data. Their classification coined the methodology for all sciences including biology, medicine, and psychology. Guttman and Stevens identified four "scales" on which we perform "measurements" or observations: the nominal scale, the ordinal scale, the interval scale, and the ratio scale. We observe qualities on nominal scales. A nominal scale is a colletion of all possible outcomes of an observation with no particular order. For example, gender, colors, or diagnoses are determined on nominal scales.

We have an ordinal scale when we can sensibly arrange the set of possible outcomes of an observation in an order. For example the NYHA classification of heart failure or tumore stagings are ordinal scales. We can determine the stage of the disease, we can tell the worse condition from the better, but we cannot measure distances.

Intervall scales are ordered quantitative scales, where you can measure distances (intervals) between two points. The paradigmatic example are temperature scales Fahrenheit and Celsius. It does, however, not make sense to say 100 degree are twice as much as 50 degrees. However, the concept of the absolute zero temperature allows to make those decisions on the Kelvin scale.

For an information standard in medicine it would be appropriate to reflect these fundamental categories of observations. However, there are some problems with this classification. You can artifically try to upgrade the scale property. Thus you can define an order of qualitative observations (e.g., male = 0, female = 1). It often depends on the scope of observation how you classify it, e.g., you can classify colors in any of those scales depending on what you think colors are (qualitative observations, up to wavelengths of visual light). The distinction between ratio and interval scales seems artificial because a simple translation of temparatures to the Kelvin scale is all that makes the difference.

Common sense will justify to distinguish qualitative from quantitative observations although the examples above show that even this boundary can be blur. We can further distinguish between observations that are discrete and those that are continuous. Many qualitative observations are continuous (i.e. colour) but continuous qualitative observations are best understood by quantization. This need not be a single dimensioned scale, as the color example shows: the RGB color quantization is a three dimensional vector of numbers.

Since qualitative and quantitative, discrete and continuous observations are important in science as well as in everyday life, we will have to devote one domain in our BDTM to discrete qualities and another domain to quantities, both discrete and continuous. We will then have to show how to express continuous qualitative observations.

There are other important kinds of information. Text is not just an abstracted observation and does not fall into the distinction between qualities and quantities, discrete and continuous. Text is ultimately exchanged between humans. Computers and automatic messaging may be used to exchange text, but after having been entered by a human user, text is passed through unchanged to be displayed to another human user. Text can express many observations, but this information content is not unlocked for the purpose of messaging and computer processing.

Text does not only include letters, words and sentences of natural human language, but can also be graphics or pictures (still or animated) or audio. Also, the same information content of natural language text can be communicated in written (characters) or spoken form (audio). Thus, one DDTM will cover text data of all those kinds. Since one property of text data in messaging is that it is passed through unchanged and uninterpreted and without respect to the destination or purpose, we can mention all other uninterpreted (encapsulated) data in the category of text.

In the exposition of the BDTM we start with text data, continue with qualities and end with quantities. We then describe three extended data type models (EDTM) for technical support data types (entity descriptor, object identifier, communication address, bits and bytes), for demographic types (person name and person address) and the EDTM on waveform data.

2 The Basic Data Type Model (BDTM)

2.1 DDTM on Text

Text

- **Text in is data that captures human utterances between humans.**
- **Text is written human language.**
  - **Human producer, human recipient.**
  - **Machine interpretation is difficult and limited.**
  - **Information theory: bits, bytes (octets), octet streams.**
- **Constituents of text as written human language:**
  - **Language: characters, words, sentences, paragraphs.**

► **Characters, character strings.**

► **What about paragraphs, fonts and other elements of style?**

► **When do we use style elements?**

► **Beyond written language.**

2.1.1 From Bits to Characters

Character strings

- **Character string is the basic type for written text.**
  - **HL7 needs no data type ‘‘character’’ - too low abstraction level.**
  - **Single letter codes (‘M’, ‘F’, ...) are just strings.**
- **Characters relate to bits through character codes.**
  - **American Standard Code for Information Interchange (ASCII) had great success over competitors (EBCDIC).**
  - **International extensions: code pages, ISO 8859, JIS, EUC JP/TW ... - a variety of different character codes for different languages.**
  - **Unicode (ISO 10646) - one code for all languages of the world.**

➡ **The equation ‘‘1 char = 1 byte’’ is no longer true.**
  - **We assume the semantics of the Unicode.**
  - **Converting bits to characters is a task of the ITS-layer.**
  - **Yet, we can not be totally silent on it.**

All information can be expressed by sequences of bits, this is the fundamental new discovery that started the era of digital information processing. Written text consists of a strings of characters and characters are by themselves expressed by sequences of bits. Eight consecutive bits are called octetts or bytes. Although we usually identify one byte with one character, this identification is not an eternal law of nature and we have to distinguish bytes from characters.

The ease by which we express characters as bytes and bytes as characters is due to the success of the American Standard Code for Information Interchange (ASCII) [**?**]. Most computers interpret bytes as characters according to the ASCII code. But this does not mean complete peace of mind. On the one hand, although ASCII is by far the most important character code, there is another one: EBCDIC.

On the other hand, ASCII does not define sufficient characters to meet the needs of non-english languages. ISO 8859-1 defines an international extension to the ASCII code that fits most languages of the world that use roman charcters (Latin-1). However, there are numerous other such extensions. And there are numerous other languages, including Greek, Russian, and Japanese.

We cannot even count on the truth that one character is expressible in one byte, as we learn from Japanese and Chinese character sets.

The solution to the Babylonian coding chaos seems to be the Unicode standard [?, ?]. Unicode is a 16 bit per character code that covers all languages of the world, with even the rarest being added in upcoming versions of Unicode. Unicode seems to be accepted in all major language communities including America and western Europe, Russia and the China, Korea, Japan. The latter three have submitted a newly created unified character set, called Han, to the Unicode, which includes more than 20000 characters.

Although Unicode uses 16 bit per character, there are encodings which allow variable length encoded characters. UTF-8 and UTF-7 both encode the basic 7 bit ASCII characters unchanged. UTF-8 uses the high bit as a marker to introduce a special sequence of variable length (2 up to 6 bytes) to encode up to 32 bits, while UTF-7 uses 7 bit ASCII characters with intermittent base64 encoded sequences. The advantage of the UTF encodings are that messages containing only of 7 bit ASCII characters are not longer, i.e. Unicode with UTF-8 and UTF-7 is backwards compatible to 7 bit ASCII. Unfortunately neither UTF-8 nor UTF-7 is backwards compatible for ISO 8859-1 (Latin-1) even though the numeric character codes are the same.

Because the data type model must be independent from the bits that are sent over the wire, we must depend on the ITS to reconstruct characters from bits. We should require of every ITS for HL7 to be able to encode all 16 bits of Unicode characters. Individual HL7 applications might need to restrict their supported character set to a subset of Unicode, for example, to 7 bit ASCII or to ISO Latin-1. These restrictions must be mentioned in the conformace claim. However, HL7 in general and all its ITSs should support the full range of Unicode characters.

2.1.2 From Characters to Strings

While single characters are data types defined by most programming languages, HL7 messages did not use single characters in the past and probably will not do so in the future. A single character is on a too low level of abstraction. There is not clinical or administartional information expressed in one character that stands for itself. There are single character codes, but in those codes the characters do not stand for themselfes but for some other meaning (e.g. 'M' for male and 'F' for female). Those codes are not text and single character codes are but a subset of codes relevant to HL7 messaging.

Therefore we will probably not need single characters as a data type.

2.1.3 Display Properties

Text and appearence

- **Text is more than just a character string.**
  - **Lines, paragraphs, pages, displays, tables (flow objects.)**
  - **Font family, style, size, alignment, color.**
  - **Underlines, rules, and other simple graphical elements.**
  - **Footnotes, cross references, logical markup.**
- **When is it useful to have control over appearence?**
  - **The difference is in semantics, not in the length of the string.**
  - **Text: a message from human to human.**
  - **String: merely an information element (name, address, code.)**

➡ **Text and string are our only distinctions.**
  - **Distinguishing TX from FT is a non-rational decison.**

- **How do we encode style elements?**

A character code like ASCII, ISO 8859, or Unicode codifies only characters, i.e., the basic graphemes from which written utterances are constructed, regardless of the variations in style. Often we are only interested in transmitting the semantics of a few words or sentences. But sometimes we want to enhance the expressiveness of text through a altered appearence of characters. Options are font family (e.g., Times Roman, Helvetica, Computer Modern), font style (e.g., roman, italics, bold), font size (e.g., 8 pt, 10 pt, 12 pt), alignment (e.g. subscript, superscript) or any other display properties.

The question is, for what use cases we need only plain character strings and when do we need control over its appearance? When a data field contains only one or a few words, we will probably not need control over appearance. However, who is to say how many words may appear in a given data element of type string? And what is the exact limit of words that do not require formatting? Clearly the length of the string is no good criterion for when formatting is required or not. Instead we need to look at fine semantic nuances to find the answer: A string that encodes a value from a code table, or a string that encodes a person's first name or address will not need formatting.

This information is readily encoded only in the characters.

Conversely, there is no reason to prevent formatting for those data elements that are placeholders for free text. Controlling appearence of text will be useful in those data elements whose purpose it is to be shown to human users. Even of only two words, we sometimes want to emphasize one word by undelining or emboldening it. Thus we have to distinguish between formalized information and free text to find out when we need control over appearence.

2.1.4 Encoding of appearance

Encoding of appearence

- **Available options:**
  - **Intrinsic features of the character code (return, backspace, formfeed.)**
  - **Non-standard reserved code positions (e.g., WordPerfect files.)**
  - **Escape sequences are most flexible:**
    - **Terminals and printers (ASCII ''escape'' character.)**
    - **C-language string literals (e.g., ''\t\234\g\n''.)**
    - **Troff commands (''.sp 1i'') or escape commands (''\s+2''.)**
    - **TeX commands (''\it italics\footnote{a syle}\par''.)**
    - **HTML/SGML/XML (''<p><it>italics</it></p>''.)**
- **Many alternatives require a general approach.**
  - **MIME is a complete solution, we should attach to it.**
  - **We should recommend using ''text/plain'' or ''text/HTML''.**
  - **For backwards compatibility we define ''text/x-HL7-FT''.**
- **Do not define different types for different encodings.**

The format of a text is encoded in three different ways. (1) Through deploying certain intrinsic features of the underlying character code, (2) through specially reserved positions in the underlying characters code, or (3) through escape sequences.

An example for (1) is the use of the ASCII control character number 8 ("backspace") to overstrike an already printed letter. Thus one can print the same letter twice or three times to yield an emboldened appearance on a simple typewriter or dot matrix printer. One can also print the underbar character over the previous letter to yield the effect of underlining. There are simple software programs that emulate the behavior of a typewriter to render this kind of simple formatting. The UNIX "more" utility, used to display online manual pages, and some terminal devices have this emulation built in.

(2) Many text processor use other control character in non-standard ways to encode the formatting of the text. For example if you look at the raw file of a Word Perfect text, you will find the words and characters interspersed with control characters that obviously encode the style of the text. The problem with this approach is that it is not standardized.

(3) Escape sequences are used by various printers and terminals. Originally these are sequences separated from the normal text by a leadng ASCII character number 27 ("escape"). But escape sequences have since been used in many different styles. In C string literals, troff, TeX and RTF we see the backslash character (\) introducing escape sequences. Troff has a second kind of escape sequences started by a period at the beginning of a new line. HL7 version 2 also uses the backslash at the beginning and end of escape sequences. SGML uses angle brackets to enclose escape sequences (markup tags) but in addition there is another kind of escape sequence opened with the ampersand and closed with a semicolon (entity references).

From the many choices to encoded formatted text HL7 traditionally used a few special escape sequences and troff-style formatting commands. This has the disadvantage that it is not very poweful and somewhat outdated by the more recent developments. HTML has become the most deployed text formatting system, available on virtually any modern computer display. HTML has been designed to be simple enough to allow rendring in real time. Thus HTML seems to be the format of choice to transmit style-enhanced free text.

A considerable group of HL7 members also pursue using SGML or XML to define text, although the purpose to using general SGML is slightly different from using HTML. Where HTML is used to control logical appearance of text, SGML is another way to structure information. Thus HL7 will use SGML as one of its message presentation formats. General SGML in free text fields is so powerful and general, that it comes with the risk of not being interoperable. However we might want to allow for it in special circumatstaces.

It will be difficult to limit the HL7 standard to just one of the possible alternative encodings of appearance. There is an issue of backwards compatibility that requires to keep the nroff-style formatting of HL7's FT data type. There is a tremendous and reasonable demand for supporting HTML, and we should not exclude general SGML and XML upfront despite the concerns for interoperability.

There are, in principle, two ways to support the multiple encodings of apperanace. Either we define multiple data types, one for old FT, one for HTML and one for general SGML/XML, or we define one data type that can contain formatted text in variable encodings. Defining multiple data types has the disadvantage that we need to descide at design time for one of those alternatives whenever a free text data element is defined. This decision is unchangeable at the time an individual message is constructed. This inflexiblility seems to overweigh the conceivable advantage that special types might accomodate the intrinsics of the special encoding formats in greater detail and accuracy.

2.1.5 From appearance of text to multimedial information

Beyond written language

- **Appearence of text and graphics - blur distinction.**
  - A quick drawing can say more than a thousand words.
  - Pen&Pad input devices will soon be in wide use.
- **Graphics and handwriting - blur distinction.**
  - Handwriting is often more user friendly.
  - Technology is already used widely and will increase.
- **Graphics and images - blur distinction.**
  - Imaging is especially important in medicine.
- **If written language why not also spoken language?**
  - Dictation is the #1 input method in anciliary medical services.
  - Audio tapes will soon become obsolete.
- **It can all appear in place of free text.**
  - The technology is available, HL7 can only benefit in using it.

Being able to format the appearance of free text adds a great deal of expressiveness. But having control over graphical appearence of text begs the question whether graphics, drawings and pictures should not also be considered part of free text, for a picture says more than thousand words. In human written communication, especially in business and science, we often use drawings to illustrate the points we make in our words. The technology to do these things on computers is available, HL7 only has to support it.

Another use for multimedial information is that this is the only way to capture the state of a text that precedes its typed form: dictation and handwriting. An HL7 message that is sent of from a Radiologist's or Pathologist's workplace will usually contain very little written information, but rather the important information will be in dicated form. Again, the techology to capture voice data, to communicate, and replay it is available on almost any PC now, HL7 only has to support it.

Two alternatives exist to support multimedial information in HL7. Since HL7 version 2.3 we can use the "encapsulated data" (ED) type. The ED data type is powerful enough to communicate all kinds of multimedial information. The problem is that it is a special data type that can only be used in data fields that are assigned to the ED data type. Currently none of the HL7 data fields is explicitly assigned to the ED data type which considerably diminished its usefulness despite its power.

The only way to use the ED type is currently in the variable data type field OBX-*observation-value*. While this serves the communication of diagnostic data that is in image or sound form, it is not genereally useable. For any multimedial data we want to send per HL7 we have to pretend that it is diagnostic data even if it isn't. If we want to send some descriptive drawing to an order, we have to pretend it's diagnostic data and send it in an OBX. Furthermore, it isn't even clear whether there will be a variable data type in HL7 version 3.

The honest alternative to support multimedial data would be to admit that any free text data can possibly be augmented or replaced by multimedial information. This means, we have to allow for multimedial data in any free text field, and thus, that free text and multimedia data share the same data type. This is not hard to do since one flexible data type was already required to accomodate the different encodings of text formats.

2.1.6 Pulling the pieces together

Wrapping up text

- **Character string.**
  - **Used when we only want raw character information.**
  - **Defined based on the Unicode semantics.**
  - **Character encoding is an ITS layer issue.**
- **Text and multimedia data.**
  - **Used for all free text data.**
  - **Components:** **1 MIME media type (text, image, audio, application)**
2 MIME media subtype
3 encoding (string, bits)
4 data
  - **We can recommend and deprecate certain media types.**
  - **We can set a default media type ''text/plain'' and encoding ''string'' such that we are backwards compatible to old TX.**
  - **We do no longer need TX, FT, ED, and we can delete CF.**

In the previous exploration of the field of text, we separated out the difference between string data elements, where the raw inforamtion of characters is sufficient and free text, where there is use for formatting the text and augment or even replace the text with multimedia information. We also found that formatted text requires a flexible data type. This flexibility could be reused for multimedial data.

This means that there will be a string data type on the one hand, and a flexible data type that covers free text and multimedial data on the other. For the string data type we can use the generality of the unicode character set to define its semantics. We do not really care how string data is encoded in bytes over the wire, as long as the character information is the same at both ends of the HL7 communication link.

Formatted text can be defined on top of string data. Due to the backwards compatibility of Unicode to ASCII and ISO Latin-1, the simple typewriter-style formatting, the troff escape sequences and HTML/SGML formatting is possible on top of Unicode strings. In addition to the string data, we have to indicate the formatting method that should be used by the receiver to render a given string correctly.

This is slightly different, however, with multimedial types. Multimedia data is best regarded as an opaque sequence of bits (or bytes) that are rendered by a special application or protocol software that understands the given stream of bits. Consequently we have to specify the application along with the raw data we transmit in a message. The two semantic composnents for multimedia data is thus the media type and the raw data.

While we can built the formatted text data type on top of the string data type, for multimedia types we need to go back behind the character strings to raw bits and bytes. There are two options. Either we let it be the task of the ITS layer (the encoding rules) to support the communication of raw bytes data or we encode raw bytes in strings using the base64 and quoted-printable encodings of MIME. It seems to be wasteful to first construct strings from bytes and then transform the strings back to bytes. While in traditional HL7 encoding rules that were unable to encode raw bytes this was a sensible way to go, it is wasteful, for example, in an object broker technology ITS.

Another alternative is to send the byte data and have it interpreted as character strings in case of free text or as raw bytes in case of multimedia types. This has the advantage that data is sent as bytes over the wire anyway and the interpretation as string is already an aditional step beyond the wire-form of messages. These distinctions, however, are not as clear when looked at from the perspective of traditional HL7 encoding rules. This alternative is supported by a closer look to standards such as HTML or SGML which do not depend on abstract Unicode strings but have their own means to interpet byte streams (e.g. HTML has a `META` element and XML defines the character set in its `!XML` header element). More traditional formatting with troff is not even able to handle the full abstraction of characters that comes with Unicode and thus is also based on byte streams rather than character streams.

As a conclusion, we can uniformly define the free text / multimedia data type as the pair of media type selector and raw byte data, while we define the format-free string of characters as a having the semantics of Unicode character strings. If the sender does not want to use any of the format options for free text but just wants to send the raw characters, he can indicate this with a special media type. It seems justified to make the plain text media type the default.

2.1.7 Requirements to ITS

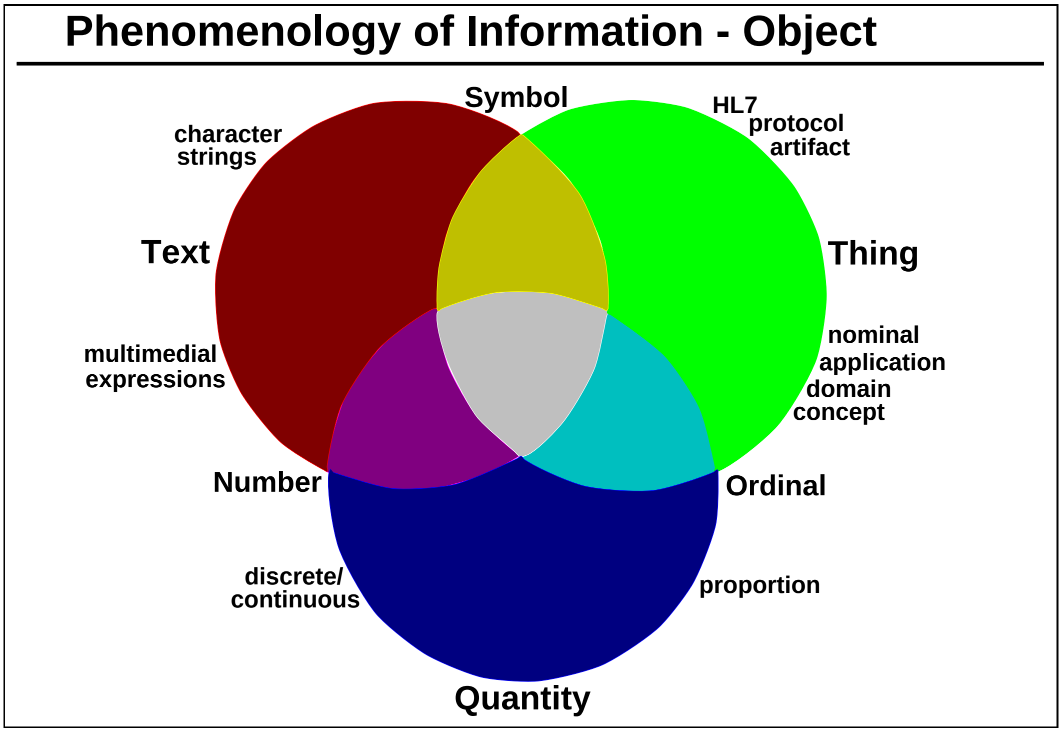
ITS must be able to transfer character strings and byte strings. ITS based on raw byte streams will supprt transmission of bytes in a straight forward manner. For character strings the ITS must interpret the bytes according to a character code. If it supports different character codes and encodings, it must be able to convert the codes to the standard Unicode and must provide ways to extend the underlying character set to include the Unicode characters.

By default the ITS should support raw UCS-2 (16 bit) or UCS-4 (32 bit) Unicode characters, or UTF-8 or UTF-7 encodings that allow to encode Unicode characters in 8-bit per character for only the basic ASCII compatible characters.

ITS based on character streams must define a way to transfer raw byte data through transfer-encodings such as hexadecimal digit pairs, base64, or quoted-printable encoding.

Although, these issuse are below the scope of the data type model, they had to be mentioned to assure the implementability of the data types specification.

2.2 DDTM on Symbols, Identifiers and Qualities



Things

- **Real world concepts.**
  - **Disease, anatomic structures, organisms, substances, roles.**
  - **Controlled vocabulary terms refer to ‘‘real world’’ concepts.**
  - **Such terms have ‘‘meaning’’ (semantics.)**
- **Real world instances.**
  - **People, organizations, locations, devices.**
  - **Names, identification numbers (SSN, DLN, ...), inventory numbers...**
- **Technical concepts.**
  - **HL7: Message type, segment tag, order control code ...**
  - **Fixed predefined sets of values, identifiers.**
  - **The meaning of such identifiers is their ‘‘effect’’ (pragmatics.)**
- **Technical instances.**
  - **Processes, agents, machines, communication devices, ‘‘users’’.**
  - **Internet address, URL, e-mail addresses, telephone numbers.**

The previous section dealt with the basic properties of textual data. Textual data is information that stands for itself. Text is usually interpreted directly by humans, though it may sometimes be the input to an automatic interpretation process (e.g., indexing, natural language processing). In any case, the meaning of text is best represented by the text itself. One use of character strings was to encode text. However, the other use of character strings is to act as symbols. Symbols are information that does not stand for itself but for something else, its meaning, that can be conceived separated from the symbol.

In semiotics and linguistics, the sciences about signs, symbols and meaning, there is no such hard difference between text and symbols, but in computer science we usually make up such a difference. We do so for good reason, because the meaing of our symbols can be described much easier than the meaning of a piece of natural language. For our discussion in HL7 there are two kinds of symbols: (1) symbols that refer to protocol artifacts of HL7 (e.g., message type, event code, data type code, segment tag, order control code, etc.) (2) symbols that refer to "real world" entities. Both kinds of symbols have different requirements.

Qualities are values on a nominal scale, i.e., a quality is one out of a set of possible qualities of one modality of reckoning. For instance if we have a set of five or six possible colors of urine, we select one of those colors when we report the color of a given urine specimen. We cannot abstract out much more about a quality than that it is one observation from a set. In this sense, a quality is not different from any term from a dictionary. Sometimes, there one can define an order relation on a set of qualities, which turns the nominal scale into an ordinal. Since the most often method to specify a set is to enumerate its elements, the enumeration introduces an artificial ordering. However, this does not turn just any nominal scale into an ordinal. The ordering should make sense for the terms of an ordinal scale.

2.2.1 Boolean values

Information theory according to Shannon and Weaver (1949) defines the smallest amount of information as one value out of a set of two values. This smallest unit of information is called 1 bit. The amount of information I in one quality depends on the cardinality n of the set of possible values, where $I =_2 \log n$. For example, one value out of 4 possible values contains the 2 bit of information. One value out of a set of only one possible values contains 0 bit of information. Thus the smallest set of possible qualities contains two elements.

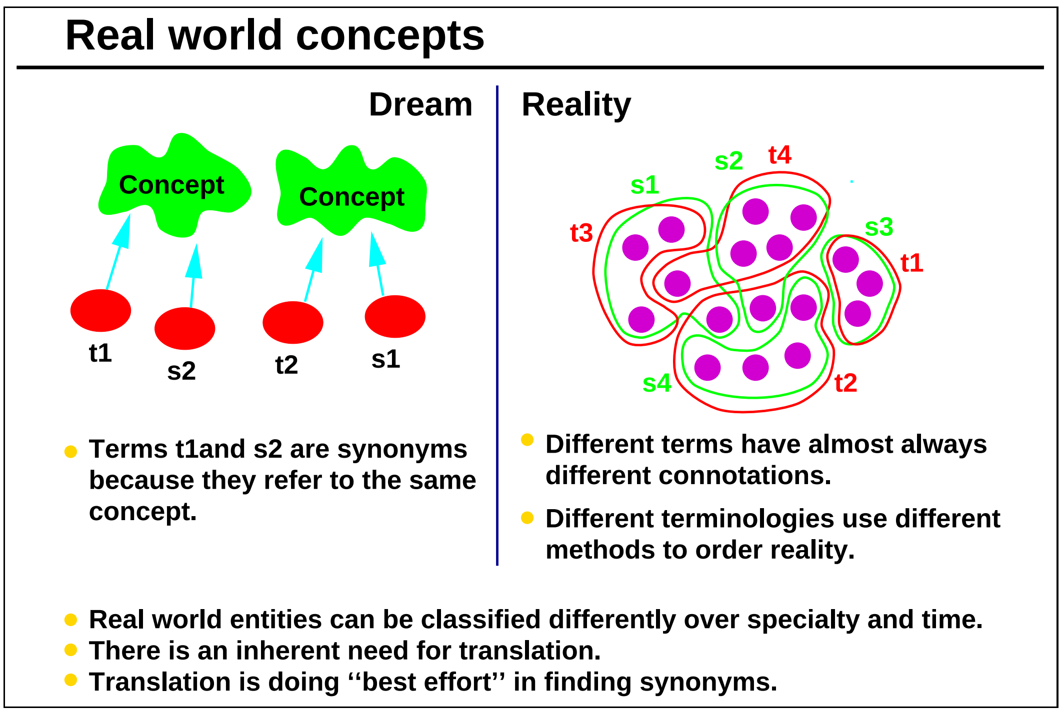
Many informations of the real world are codeable in one bit. This is every question that can be posed in such a way that either "yes" or "no" answers the question. For example, the question is the patient dead is answered completely by either "yes, (he is dead)" or "no, (he is not dead)". HL7 uses to call those attributes that are answered by either yes or no "indicators".

Because the bit as the smallest unit of information assumed such a fundamental practical importance in digital computers, all machine languages and almost all higher level programming languages have the notion of the boolean data type, with the possible values true or false. The boolean data type is often the data type that is most easyly handled, most quckly manipulated, and uses up the least amount of memory.

For an unknown reason, HL7 never defined the "indicator" data elements explicitly as of type boolean, but rather treated them as one kind of quality among others. Yet the set of possible values for an indicator contained two elements 'Y' and 'N'. Hence, the indicator data elements encode are equivalent to boolean values.

HL7 version 3 will have a boolean data type to be used for "indicator" kind of data elements. One objection to this might be that sometimes we cannot answer a question with either "yes" or "no", because we do not know the answer or are unsure about it. The problem of incomplete informartion or uncertainty, however, is not a problem only for boolean data, but any kind of information can be subject to incompleteness of uncertainty. This is a general problem that will be dealt with by general solutions specified in the Fundamental Data Type Model in section **??**. In no way does the problem of incomplete information or uncertainty preclude the notion of a boolean kind of quality or the use of the boolean type.

2.2.2 Real world concepts (Nominal values)



Real world concepts

- **The ‘‘meaning’’ of a term is difficult to define.**
  - **The approach of the code system (i.e. clinical, biochemical, genetic)**
  - **Every term is in opposition to the other terms in the system.**
- **Inherent need for translation.**
  - **... leads to accumulation of multiple translations.**
- **Synonymity is a problematic relation between terms.**
  - **But we can score the quality of a translation ...**
    Masarie FE, Miller RA et al. 1991; Rocha RA, Rocha BH, Huff SM 1994; or Nolan WA, Rector AL et al. 1994, and others.
  - **... though there is not one ‘‘standard’’ score.**
- **The real world concept is communicated as:**
  - **a list of ‘‘best effort’’ synonymic terms**
  - **a term is a pair of term, code system (including version)**
  - **a term may have a textual name (character string)**
  - **a term may have a quality or translation order component**
  - **we also want to keep the original text.**

Most medical information comes as qualitative information: complaints, symptoms, signs, diagnoses, goals, interventions, surgeries or medications, all of these are informations on a nominal scale. But not only medical information, administrative data often is on nominal scales, patient class (inpatient, outpatient, etc.), insurance, health plan, and many other data elements. These nominal scaled values are variables that can take on one value of a list of possible values. Information theory can not tell us more about those qualities than the amount of information represented by one such variable, yet, what we are often more interested in is the meaning of each of the values that the variable can take on.

The branch of science that teaches us about the codes and their meaning is semiotics and linguistics, on which we have to give a condensed overview to prepare the necessary background on the nature of qualitative information. The first Authors on modern Semiotics and Linguistics were Ferdinand de Saussure (1916) and G. W. Pierce. They draw on a treasure of work in philosophy reaching from Aristoteles to Gotlob Frege. Semiotics and linguistics has grown rapidly to an elaborated science tightly interlinked with logics, epistemology and, of course, computer science. It is impossible to present a short overview. But it is still necessary to point out some problems with current semantic theories used in medical informatics.

The assumption that the meaning of a sign has an objective reality in the world is not quite true. The traditional example is the “unicorn.” We know that unicorns do not exist in nature nor, we assume, did unicorns exist at any time in the past. Nevertheless, “unicorn” certainly is still a sign. Of course, science is supposed not to treat unicorns, but scientific terms are to be reproducible and verifiable through experiments. However these experiments still do not show the referent per se, but must rather be interpreted such that the existence and nature of the referent becomes plausible. The referent is a problematic notion and is not essential in the understanding of signs and meaning. According to Eco, signs refer to “cultural entities” the meaning of a sign can be explored through denotation and connotation.

In linguistics this exploration of signs through denotation and connotation has led to the component analysis, where each token is linked with its connotations or semantic markers (Katz and Fodor). Regardless of the details of these theories, we have to recognize one fact: the meaning of a sign is in turn codified by other signs. We can never expose meaning without using signs. The human ability to use signs is a powerful tool to convey meaning and to create meaning, but meaning is always “locked” into signs. Thus denotation of a term is basically translation into anoter code.

To understand the nature of meaning we are thus thrown back to understanding the nature of codes and translation between codes. Although “code” is a term even more general than “language”, we often use code in a more narrow sense, meaning a set of terms about a given scope, a so called “semantic field.” A code divides its semantic field into elements that are considered atomic in the sense of the code. Thus, we say that a code assumes a certain segmentation of its semantic field.

We can clarify this point more formally. Let R be the set of objects in the “real world” (“referents” in the sense of Richard). Let S be the set of symbols.[1] The human mind approaches reality trough methods which find similarities or equivalences in certain respects. These methods can be understood as relations on R where some of them are equivalence relations $\sim$ (symmetric, reflexive and transitive relations). These relations allow us to determine whether two individual entities in the real world $r_1, r_2 \in R$

[1]We do not argue whether the set R of referents exists “in reality” nor do we argue whether there is an injective mapping $f' : S \mapsto R$ so that every sign $s \in S$ refers to an objective entity in the real world $r \in R$. We do argue, however, that we can not discuss about real world entities $r \in R$ directly.

are equivalent $r_1 \sim r_2$ regarding the relation. The relation $\sim$ generates equivalence classes $c_i \in 2^R$. Let $C_\sim \subset 2^R$ be the set of equivalence classes generated by the relation $\sim$. Any code system S may assume a different equivalence relation $\sim$ so that there is a one-to-one and onto mapping $f : S \mapsto C$.

Thus any code $\mathbf{C}$ is a triple $\langle S, \sim, f \rangle$ consisting of a set of symbols S, the equivalence relation $\sim$ and the mapping $f : S \mapsto C$. Every symbol $s \in S$ refers to one set of equivalence classes c_s generated by the relation $\sim$, and not directly to elements r in the real world R. Another code $\mathbf{C}' = \langle S', \dot\sim, f' \rangle$ will imply a different segmentation $C' \subset 2^R$. Translation between $\mathbf{C}$ and $\mathbf{C}'$ is thus problematic. The translation may not be injective or surjective. Hjelmslev has given a remarkable simple example of the problem by depicting the different segmentations of the semantic field around wood, tree, forrest in different languages shown in Figure **??**. We can easily find similar examples for medical coding systems.

<table>
<tr><th>Französisch</th><th>Deutsch</th><th>Dänisch</th><th>Italienisch</th></tr>
<tr><td rowspan="2">arbre</td><td rowspan="2">Baum</td><td rowspan="3">trae</td><td rowspan="2">albero</td></tr>
<tr></tr>
<tr><td rowspan="4">bois</td><td rowspan="2">Holz</td><td rowspan="2">legno</td></tr>
<tr><td rowspan="5">skov</td></tr>
<tr><td rowspan="4">Wald</td><td rowspan="2">bosco</td></tr>
<tr></tr>
<tr><td rowspan="2">forêt</td><td rowspan="2">foresta</td></tr>
<tr></tr>
</table>

2.2.3 Technical concepts

Technical concepts

- **The ‘‘meaning’’ of a symbol is defined in a model.**
  - **Tags for messages, ‘‘segments,’’ and data types: MET-graph (HMD)**
  - **Order status & control codes: state–transition–diagram.**
  - **Media type code: MIME specification.**
- **The symbol does not make sense apart from the model.**
- **Symbols of other models do not apply.**
- **Translations are impossible, synonyms do not exist.**
- **Generally symbols are just distinct elements in a set.**
  - **The set must be selected in a declaration, but is implicit in an instance.**
  - **The set of allowable symbols and their function must be defined.**
  - **Having two different data types (ID vs. IS) does not help.**
  - **The way a symbol table is defined may vary however.**

2.2.4 Technical instances

Technical instances

- **Such symbols have their ‘‘meaning’’ through a technology that can resolve the symbol.**
  - **A telephone number is resolved by the global telephone network.**
  - **An Internet address is resolved by the Internet, DNS, routers, etc.**
  - **A URL is resolved by the specified protocol (e.g., ‘‘http:’’.)**
- **Technical instances are agents of real persons.**
  - **... not the persons (individuals or organizations) themselves.**
  - **E.g., devices (computer, phone, fax), mailboxes, processes.**
- **No difference between ‘‘agents’’ and ‘‘things’’.**
  - **Things (e.g., messages, files) can be seen as ‘‘living’’ objects.**
  - **No difference between, e.g., ‘‘show(obj)’’ and ‘‘obj.show().’’**
  - ⇒ **No fundamental difference between EI and HD.**
  - **We may want to retain an HL7 object id for ‘‘order number’’.**
- **We should look at the URL/URI mechanism to see if we really need an HL7 specific reinvention.**

- entity descriptor (actor)
- object identifier (object)
- entity address (channel to actor)

2.2.5 Real world instances

Real world instances

- **People (individual persons)**
  - **Indiv. person name: first, last, prefix ...; type (maiden, legal, alias ...)**
  - **Person id: id string, type, issuing organization.**
    patient id, SSN, driver's license #, medical record # ...
  - **Organizational affiliation: org., role (chair, CEO, ...)**
- **Organizations**
  - **Org. name: name string, legal status (Inc., e.V., B.V., AG, ...)**
  - **An org. is a legal person: person id applies just as for people.**
  - **Organizational affiliation applies just as for people.**
- **Locations**
  - **Residential address: country, state, city, street, house #, app #**
  - **Postal address: coutry, ZIP, (PO.box; (street, house #, app #))**
  - **General locators: loc1.loc2. ... .locN (building, tract, floor, room)**
- **Things**
  - **Thing id: id string, type, issuing organization.**
    inventory #, lot #, equipment id, book signature, ...

- individual person name
- person address
- organization name
- (official) person identifier

2.2.6 Ordinal values

Ordinals

- **Ordinals are sets of symbols with an order relation.**
  - Severity: ‘‘mild’’ < ‘‘moderate’’ < ‘‘severe’’.
  - Detection: ‘‘-’’ < ‘‘0’’ < ‘‘+’’ < ‘‘++’’ (‘‘2+’’) < ‘‘+++’’ (‘‘3+’’).
  - Clinical stagings: NYHA (1-4), lymphoma, other tumors.
  - Clinical scores: GCS, APGAR, APACHE, etc.
- **Ordinals are different from numbers.**
  - Differences (‘‘distances’’) between values are undefined.
  - Ratios are even ‘‘less’’ defined.
- **Ordinals are different from real world concepts.**
  - The ‘‘real world’’ meaning is usually hidden (e.g., GCS).
  - Translation is often very difficult.
- **Components:** (tentative)
  1. value (string) - required;
  2. scale id (real world concept);
  3. scale enumeration (list of strings);
  4. scale position (integer);
  5. scale size (integer).

2.3 DDTM on Quantities

Quantities

- **Numbers**
  - **Integer: the result of counting, exact by definition.**
  - **Rational: the result of division of two integers, exact by definition.**
  - **Real: only as floating point approximations.**
- **Measurements**
  - **An approximation to a physical property of objects or processes.**
- **Time**
  - **Point in time measured on ''arbitrary'' calendars.**
  - **A duration is a measurement of time.**
- **Ordinals**
  - **Semiquantitatives such as ''-'', ''0'', ''+'', ''++'', ''+++''; ''mild'', ''moderate'', ''severe''; clinical stagings and scores.**

2.3.1 The number concept

Numbers

- **Integer**
  - **The result of counting, exact by definition.**
  - **''SI'' was a positive integer is this distinction really important?**
  - **Integers need quite a lot of bits (32 bit for only 4 billion.)**
  - **however, we should not impose an arbitrary limit.**
- **Rational**
  - **Numerator (integer) / denominator (integer), exact by definition.**
  - **Used in expression like 1:64, 1:128 for titers.**
- **Floating point**
  - **The number approximation of a real number, imprecise.**
  - **Number of significant digits is an approximation to the precision.**
  - **Significant digits are implicit in digit strings (by standard rules,)**
  - **must be made explicit in binary encodings (IEEE float.)**
  - **A more exact approach to imprecision will rarely be necessary.**

2.3.2 Dimensioned quantities

Measurements

- **Approximation to a quantitative physical property objects or processes.**
  - **Needed for clinical observations, orders, scheduling, administration.**
  - **A measurement value v is a the result of a comparison of a quantity Q with a standard object or process U according to a precept of measurement and an apparatus of postulates.**

    $$v = \frac{Q}{U} \quad \text{thus, the quantity is} \quad Q = v \cdot U$$

    A measurement Q is a pair of value and unit.
- **Units are quantities themselves (not just codes.)**
  - **Units form an Abelian group regarding multiplication.**
  - **A code for units is an term algebra on an infinite set ...**
    ... not just a finite set of concepts.
- **''The Unified Code for Units of Measures'' completely defines syntax and semantics of a units code.**

also deals with currencies

2.3.3 Enumerations

see section on Ordinals.

2.3.4 Calendar Date and Time

Time

- **A point in time is a position in an arbitrary calendar.**
  - **Durations can be measured on ratio scales ...**
    ... point in time is an interval scale arbitrarily fixed at some ‘‘Epoch.’’
  - **The time axis is ‘‘convoluted’’ in arbitrary cycles:**
    second, minute, hour, day, week, month, year, decade, century, ...
  - **A point in time is a pair of calendar and calendar expression.**
  - **Default: Gregorian calendar and Universal Coordinated Time (UTC.)**
- **A point in time is always an approximation.**
  - **Similar concept as significant digits, but not a decimal system.**
  - **Precision is usually rounded to calendar unit.**
- **Various ‘‘modulo’’ expressions are defined in calendars.**
  - **Time of day, weekday, week of year, month of year, ...**
  - **in general: a point in time and a pair of calendar units.**
  - **Useful defaults are hour and minute of day and day of week.**
- **Time durations are simply measurements.**
  - **Measurments of time: 1 s, 1 min, 1 h, 1 d, 1 a, ...**
  - **Age of life is a measurement of time, e.g., 40 a.**

3 Abstract Data Types

Orthogonal issues

- Collections: the true meaning of ‘‘repeated’’ data.
- Intervals: applicable for all ordered types.
- Probability: the challenge of the ‘‘real world.’’
- Incomplete information: the other challenge.
- Update semantics: how messages affect databases.
- Historical data: providing for change.

- These are generic types, used with other types in message specification.
- Message instances can send a related generic type for a proper type.
- Typecast operations can be defined to use one for the other.
- Semantics and constraints of typecast operations must be defined, to prevent:

  ‘‘Diagnosis: PNEUMONIA (P=0.01)’’ to yield ‘‘Diagnosis: PNEUMONIA.’’

3.1 Generic Data Types

Collections

- **We should be specific what ‘‘repeated’’ data is:**
- **List: an ordered collection of data,**
  - **used to construct numeric array, multiplexed array, etc.**
- **Set: an unordered collection without duplicates.**
  - **Set of likely diagnoses, set of status flags, set of allowed values, etc.**
- **Bag: an unordered collection with duplicates.**
  - **Alias names, telephone numbers, identifiers, etc.**
- **Array, stack, queue, dqueue are not relevant,**
  - **for those collections differ from lists only in dynamic behavior.**
- **Type cast rules can be used to seamlessly take a collection for a single value and vice versa.**

3.1.1 Structures

This is the message element type – outside of the data type model. We define specific composite types here but we won’t come up with the CM again. The CM was a generic data type for structures. Defining a generic CM is quite screwed. Not much to say here: kill this section.

3.1.2 Vector

A vector is a collection with a fixed number instances of a single type. Well, that’s not even true. In LISP a vector is just a fixed length sequence of any type (isn’t it Mark?). In mathmatics a vectors form an algebra. In physics we think of a vector as a directed quantity. However, the RGB color scheme is a vector. And the TNM tumor staging is a vector. So what?

3.1.3 List

Listst are collections of items (of the same type (except in LISP)) where the order does matter and the same value can occur multiple times.

3.1.4 Set

unordered collection w/o duplicates.

3.1.5 Bag

unordered collection with duplicates.

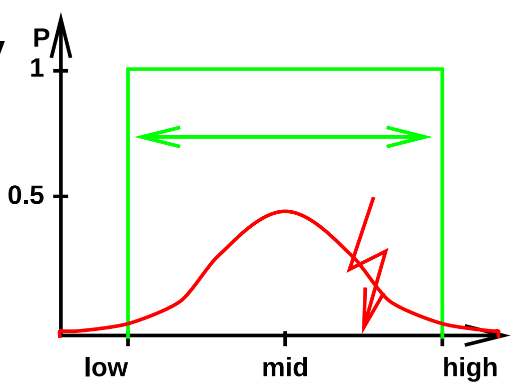
Other common collections, stack, queue and dqueue, are irrelevant for us because these are dynamic properties of lists.

3.2 Fundamental Data Properties

3.2.1 Intervals

Intervals

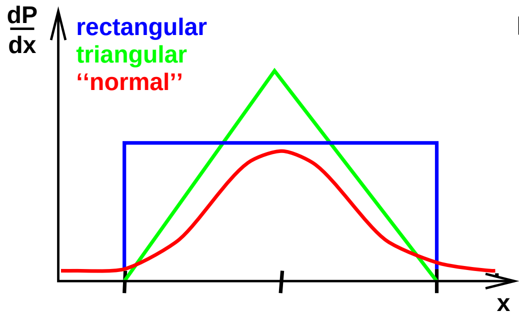
- **Intervals can be formed of all ordered sets.**
  - **Integers, rationals, floating point, measurement, time, ordinal, ...**
    ... always in the same way: lower limit, upper limit.
- **Different kinds of intervals:**
  - **open or closed on either side.**
  - **limited or unlimited on either side.**
- **Interval versus probability**
  - **Intervals are used for regularly continuing processes.**
  - **Probability distributions are used when there is uncertainty about what the ''actual'' value was or what it will be.**



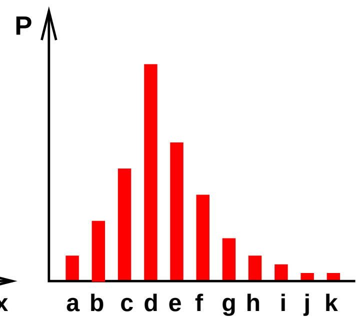
3.2.2 Uncertainty

Probability distributions

- **Continuous scale: probability density function.**
  - **lab tests, all kinds of measurements, statistic outcomes.**
- **Discrete scale: histogram.**
  - **set of likely diagnoses.**



distribution type and 1, 2, or more parameters (mean, variance, etc.)

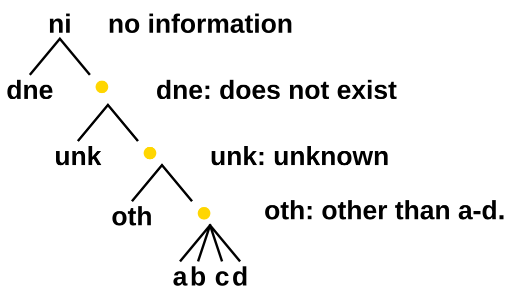


a set of pairs (value, probability)

3.2.3 Incomplete information

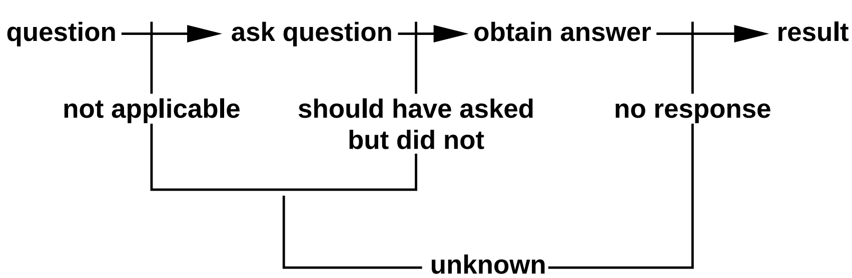
Incomplete information

- **The ambiguous ‘‘NULL’’, what does it mean?**
  - **missing information, for various reasons.**
  - **not applicable information (e.g., my medicare number).**
- **Domain extensions are the way to deal with those.**
  - **HL7 v2.x: ‘‘not present’’ (||) and ‘‘null’’ (|""|) are domain extensions.**
  - **Domain extension apply to all types.**
- **Notation to understand domain extensions:**
  - **Decision tree is a way to sort things out.**



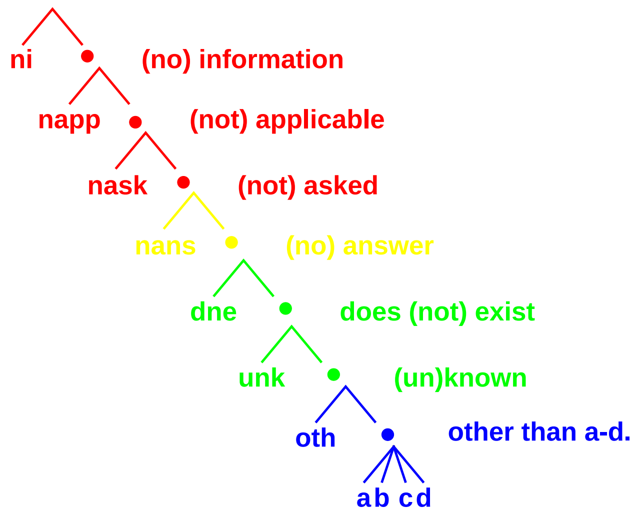
The ‘‘human factor’’

- **There is a difference between ‘‘no’’ and ‘‘not asked,’’**
  - **which is quite important in clinical medicine,**
  - **but problematic to document honestly in practice:**
  - **Only the one who didn’t ask can document ‘‘not asked,’’**
  - **but it is usually an (embarassing) mistake not to ask.**
  - **instead of documenting ‘‘not asked’’ one could as well go and ask.**



Incomplete information

- Merging the human factor into the decision tree:



3.2.4 Update semantics

Update semantics

- If I know X and you tell me Y what should I believe?
  - Should I update?
  - Should I complain?
  - "Not present" vs. "null" were distinguished by update semantics.
  - Should you be able to wipe out my information?
- Update semantics must be specified for every piece of data for every message definition.
- Update semantics should be modifiable for many pieces of data in message instances.
- Relation to uncertainty and incomplete information:
  - If A knows nothing and B knows X, than A should belive X.
  - If A believes X (0.8) and B believes Y (0.9) and A believes B (0.8), then P(Y) < P(X) for A, hence A will continue to believe X.
  - But we may sometimes want to "downgrade" status or probability.

3.2.5 Historic dimension of data

The historical dimension

- **We may sometimes want to know old information.**

  For example: old name, old address, old identification number.
  HL7 v2.3 ''financial class'' has ''effective date,''
  driver's license or credit card numbers have ''expiry date.''

- **Abstract syntax:**
  - **Pair of base data and interval of calender time.**
  - **No low limit means: expiry date.**
  - **No high limit means: effective date.**
  - **Current time within interval: this is the actually valid data.**

- **Historical data often comes in a set,**
  - **with the actual value being an element of the set.**
  - **a history set can be sent for all simple data, the typecast picks the actual value or ''no information'' if actual value is not an element of the set.**

4 Strawman types for other technical comittees

4.1 CQ

Are there any left? RP, OID (EI) ?

4.2 PAFM

Real world instances: Person name, address, organization name ... It is important to see those types in the context of PAFM, because some of those former types are now modeled as RIM classes (e.g., stakeholder identifier, person alternate name, ...)

4.3 ORD

4.3.1 TQ

We do not want Wes to commit suicide and thus we want TQ to be modeled in the RIM or at least clean TQ up so that it is a "'good"' data type. The goal is again simplification and to design TQ so as to have only one way to express every meaning. We can draw from the Work that Linda Quade, Tim Snyder and I did for the USAMP initative.

4.4 Automated data

MA, NA, and CD need to be brought in line with what we define. This is not a big deal.