Semantic Model Driven Engineering

David Milward

Wolfson College

University of Oxford

submitted for the degree of
DPhil in Computer Science
Michaelmas Term 2020

Abstract

Semantic interoperability is a concern in the design and implementation of modern information systems. Whenever two systems communicate it is important that they share an adequate, common interpretation of any information transferred. Similarly, whenever data created or acquired in one context is used in another, it is important that the provenance of the data is consistent with its intended usage.

Medical researchers depend on understanding good quality data, most of which is kept in information processing systems designed for multiple purposes by multiple organisations and individuals with different motivations in mind. As a result the original meaning and context of the data being used in analysis can be difficult to establish, in addition there are many ways in which data can be corrupted, missing or otherwise altered, resulting in research data scientists spending the majority of their time in activities variously referred to as 'data-cleaning', 'data-wrangling' and 'data-munging', sometimes leading to a lack of trust in the quality of the data.

This thesis is a response to that challenge. The approach combines and builds upon existing work in two distinct areas. The first is that of data standards and metadata registries, in which a consensus has been reached regarding the abstract representation of the 'semantics' of a data element. The second is that of software modelling and model-driven engineering, in which a consensus has also been reached regarding the abstract representation of data structures and the relationships between data elements.

The key contributions of this thesis are firstly a formal language for reasoning about semantic interoperability in terms of data models. In addition a formal grammar is provided on which metadata registries can be built guaranteeing consistency of models. A modelling language for metadata description is introduced, based upon the widely-used Eclipse modelling framework, a de-facto standard in model-driven engineering. The thesis then demonstrates that this language, used in combination with a suitable set of constraints for metadata registration and management, can be used to achieve and maintain semantic interoperability in practice.

Contents

Chapter 1

Introduction

1.1 Big Data

Advances in computing technology have led to an explosion in the availability and importance of data. The widespread use of 'smart' phones, watches, and tablets has meant that data is being created and consumed by billions of people as they go about their daily lives. The deployment of new, autonomous sensing devices, as an 'internet of things', connected to high-bandwidth wireless networks, has enabled the collection of detailed information about these people, their society, and the environments they inhabit.

This explosion has led to the use of the term *big data*. A systematic review of the literature [15] proposed a 'consensual' definition:

> big data represents the information assets characterised by such a high volume, velocity and variety to require specific technology and analytical methods for its transformation into value

While this definition includes the four *v*'s—volume, velocity, variety, and veracity—often used to characterise big data from a non-technical, management perspective, it includes also the need for specific technology. For clarity, we define and describe the four *v*'s as follows:

> Volume - Terrabytes and Pepibytes of data being ingested into a system on a daily basis, contributing to exobytes of data being transferred over the internet.
>
> Velocity - The speed that the data needs to be processed. This aspect concerns the amount of data which is incoming and the amount of information which needs to be extracted from that data quickly, for instance data from credit card swipes incoming to a bank.

> Variety - Different types of data structure. This aspect concerns the type of data which may vary from highly structured XML or database data, to logfiles, mp4 video and audio files. The type of data being transfered today is in many different forms, and to derive information from it a variety of rules need to be applied quickly and effectively.
>
> Veracity - The trustworthiness of the data. Is the data indeed useful? Will it help add value to the organisation processing it?

Other definitions of big data have identified the same need for technology: for example,

> data that cannot be handled and processed in a straightforward manner [22]:

and

> datasets, primarily in the characteristics of volume, velocity and/or variety, that require a scalable architecture for efficient storage, manipulation, and analysis [29]

New hardware and software architectures, such as Graphics Processing Unit (GPU) computing and the Apache Hadoop software library, have been developed to address the challenges of volume and velocity. To date, the challenges of variety and value have been addressed using existing technologies and methodologies.

1.2 Semantic Interoperability

The challenge of variety is one of interoperability or coordination. For the two systems to work together effectively, to interoperate, requires an adequate, shared understanding of the communications between them. The question of what constitutes an 'adequate' understanding will depend upon the purpose, or the situation: two systems may be interoperable for some purposes, but not for others.

This shared understanding may be reflected in the implementations of the systems concerned in two quite separate aspects of design. The first of these concerns the format of the communication: how any data sent between the two systems is formatted, encoded, or structured; how any message received is to be parsed. The second concerns the interpretation of any message or data communicated: how any message, once parsed, is to be understood and acted upon.

As the successful implementation of the first aspect of the design is a prequisite for a successful implementation of the second, it is understandable that much of the work on interoperability has focussed upon this first aspect, which may be termed *syntactic interoperability*: the two systems communicate using a shared language, a common set of structures and terms; the two systems can work together.

The second aspect, however, is essential if the systems are to work together effectively. The two systems should share an adequate, common interpretation of any information transferred. Whenever data created or acquired in one context is used in another, it is important that the provenance of the data is consistent with its intended usage. We will refer to this as *semantic interoperability*. The term 'semantics' encompasses notions of provenance, interpretation, and intended usage.

These properties are contextual, rather than inherent in the data itself. The importance of semantic interoperability, as a core objective in systems design, increases with the number and variety of contexts in which data may be acquired, created, or used: any variation could mean that the data is not fit for the purpose to which it is being put.

The challenge presented by the increasing value of the data is akin to that presented by critical systems: errors in design or implementation can have significant consequences for users, data subjects, organisations, or societies. The response must be the same: to apply methods and tools that can help reduce the potential for error, or help detect errors, or even prove that a design or implementation is error-free.

'Formal methods', employing languages and tools with formal, mathematical foundations, have been used to establish the correctness of hardware and software architectures for big data [61], helping to ensure that value is maintained despite the volume and velocity of the data. To ensure that the variety of data, and the variations in context, do not lead to critical errors, and the destruction of value, we require methods and tools that will help assure interoperability—and, in particular, semantic interoperability.

1.3 Existing Approaches

Interoperability is an ongoing research topic in computer science which has been studied since the beginnings of the discipline of computer science. The emergence and development of the internet in thelast 15 years has resulted in a resurgence of research in both semantic web technologies and interoperability of web services. The idea of semantic interoperability, the notion that different computers can automatically interoperate on a loosely coupled basis, has been

tantalising researchers over this period, however to date no clear methodology to achieve this has emerged. About 10 years ago large efforts were made in developing semantic web technologies based on description logics, and this has led to progress in achieving semantic interoperability, with one caveat. The results involve embracing the whole semantic web technology stack or space, and industry is reluctant to do this *en masse*. Most healthcare organisation still use relational databases for over 90% of their storage needs. In general they do not use triple stores, Resource Description Framework (RDF) [87] or the Web Ontology Language as recommended by the W3C (OWL) [2] in any of the mainstream data processing. We need to increase the level of interoperability in current systems.

The importance of semantic interoperability has been widely recognised, and many different approaches have been developed. Most of these involve agreement upon a single data model, or a single ontology, for a particular domain or 'community of interest'. These approaches have met with some success, where the meaning of the data that is to be shared is largely unaffected by context, and the nature of the data to be shared remains relatively constant.

The difficulty in achieving a single, unified view was a concern for those working upon the 'Semantic Web', where a single ontology was often the ultimate aim. In [74], for example, the authors, Myriam Ribiere and Rose Dieng-Kuntz consider information or views in terms of *perspectives*, which are typically complementary and stem from an expert's incomplete view of the world; and *opinions*, which are likely to be collectively inconsistent and reflect an individual's convictions. A methodology is presented for integrating consistent views into a single ontology.

Similarly in [83, 84], ontologies are registered against a user; the collective facts may be inconsistent or conflicting, but ultimately "... the correct knowledge is taken to be equivalent to the consensual knowledge, and thus the knowledge engineer's task becomes one of finding consensus within the divergent knowledge of the experts". Changes within an ontology can only be carried out if they are consistent with the rest of the ontology, and this extends to multiple ontologies if they are combined.

A particular challenge in the application of the semantic web ontology language OWL is the semantics of the $owl : imports$ construct. This can be used to combine multiple ontologies, losing the context of each, and potentially introducing conflicts. In [10], the authors extend OWL with $\mathcal{E}$-connections in order to achieve modularity when combining ontologies. However, in these approaches, no reasoning is possible in the presence of contradictions: a single knowledge base must remain entirely consistent.

The joint International Standards Organisation (ISO) and International Electrotechnical Commission (IEC) 11179/3 standard for metadata registration [33] set out to provide a single

language for describing individual data items in data standards. This would allow users to achieve semantic interoperability one data item at a time, in a data standard or schema.

This standard was the original inspiration for our approach, and the catalogue language was developed in an attempt to address three particular, decisive shortcomings:

- The standard has no notion of the composition of data items: each attribute is defined separately. This means that every attribute needs to be described in full, and that there is no re-use of textual explanation or logical constraints across the items of a class, or the classes in a model. Furthermore, it means that there is no basis for inferring context from the structural relationships in software artefacts, nor is there any basis for generating those artefacts.

- The standard has no support for links between attributes. Instead, attributes are grouped according to 'data element concepts', requiring that all of the definitions involved are considered to be equivalent. This requires the same, impossible degree of coordination as that required for agreement upon a single data model, except that now we are proceeding one attribute at a time.

- The standard supports only a single ontology for the organisation of 'data element concepts', meaning that all users of a particular registry must agree upon the same set of relationships between these concepts.

Nevertheless, a number of ISO11179-3 compatible metadata repositories or catalogues have been implemented. [30] argues that such tools "should result in fewer errors and less effort required for designing clinical studies".

In [19], the authors explain how they augmented the ISO/IEC standard with Clinical Data Interchange Standards Consortium (CDISC) Operational Data Model (ODM) data models. The resulting catalogue contained around 800,000 terms with semantic annotations, derived from around 5,800 models. They concluded that "Uniform manual semantic annotation of data models is feasible in principle, but requires a large-scale collaborative effort due to the semantic richness of patient data."

There have been other formally-based approaches platform-independent modelling and analysis. The Heterogeneous Tool Set (HETS) [60] defines a language for the analysis of models and transformations, based upon the modelling language Unified Modelling Language (UML) and the associated transformation language Query View Transformation (QVT). The toolset

is focussed on the automatic construction of code from models, and supports the verification of some simple structural and non-structural constraints.

HeteroGenius [59] represents a similar approach, although focussed instead upon the analysis of software specifications and the management of proofs. In [6], the authors present an abstract notation suitable for reasoning about refinements in a variety of different formalisms. This category-theoretic approach can be used for building language-independent models, but with a focus upon behavioural properties, rather than the structural aspects we are concerned with in this research.

Our work thus far has focussed upon establishing semantic interoperability based upon data abstractions of software artefacts. We have identified an approach that can be applied at scale, having support for both automation and compositional, potentially-federated development. We have developed a formal semantics for the language that underpins this approach, and used this semantics in the development of a metadata catalogue that supports multiple, potentially-conflicting perspectives and yet can be guaranteed to provide consistent information.

This work has developed from the domain of healthcare research. However, the approach set out here is equally applicable to data management challenges in other domains, the need for new technologies to support reliable interpretation and re-use of data at scale exists also in areas such as electronic government, defence, supply chain management, and financial regulation.

1.4 Research Contribution

The research reported here is focussed upon the development of methods and tools for semantic interoperability. The central thesis is that by combining notions of data standards and semantics with the data modelling and code generation techniques of model-driven engineering, we can produce a scalable approach to the challenge of semantic interoperability.

More precisely, we formulate a formal specification and notion of semantic model-driven engineering, and then show how it may be implemented: in a metadata modelling language, and in a set of principles for the use of that language. Finally, we show that the approach does indeed work in practice, by demonstrating its application in a key domain, that of large-scale, data-driven health research.

The term ‘semantics’ encompasses notions of provenance, interpretation, and intended usage, as opposed to the syntactic representation of the information transferred, or the structure and format of the data to be used. The contextual nature of these properties means that achieving and maintaining semantic interoperability is a significant challenge.

This thesis is a response to that challenge. It shows that semantic interoperability may be achieved and maintained through the production of abstract models of data and context and the subsequent use of these models in the automatic generation and configuration of software components: that is, through 'semantic model driven engineering'. It also demonstrates the practical application of this approach within the domain of health research informatics.

The approach combines and builds upon existing work in two distinct areas. The first is that of data standards and metadata registries, in which a consensus has been reached regarding the abstract representation of the 'semantics' of a data element. The second is that of software modelling and model-driven engineering, in which a consensus has been reached regarding the abstract representation of data structures and the relationships between data elements.

The thesis begins with an account of model-driven engineering and semantic interoperability, and an explanation of the need for semantic interoperability in health research. It continues with detailed examination of the existing work on metadata standards, focussed upon the leading international standard ISO 11179. It identifies key shortcomings of this standard, and presents a formal, mathematical account of how they may be addressed. The novel research contributions can be summarised as:

- An improved metamodel for metadata management in the form of a domain specific language - Metadata Modelling language (MDML), based on a formal specification, for metadata registries, enabling automatic registration and classification of data elements.

- A registry design based on MDML which allows improved semantic mapping of data items from diverse sources to be registered, defined and mapped in a way which is machine-accessible for automatic classification and code-generation tasks.

- A deep-search capability for researchers to access useful information about research data before being able to access the data directly.

- The ability to contain other descriptive metadata standards within the MDML model, allowing standards such as DCAT to be incorporated within MDML.

These contributions, while adding to the formal theory of semantic interoperability, are significant in that they allow an advance in semantic interoperability in practise by improving the standard metadata registry model defined in ISO11179 thus allowing semantic matching and aligning of datasets to be carried out automatically. This process will still need close expert management, but it allows the tedium to be taken out of dataset matching.

Table 1.1: Key standards (Medical and Data) referred to in this thesis

LOINC	Standard for medical laboratory results (International)
FHIR	Standard for medical messaging (International)
ICD10	Standard for disease codes and descriptions (International)
HL7	Standard for medical messaging (International)
OpenEHR	Standard for electronic medical records (International)
OPCS-4	Classification of Interventions and Procedures (UK-NHS)
READ codes	Standard for medical codes (UK-NHS)
COSD	Standard for cancer terminology (UK-NHS))
ISO/IEC 11179	Standard for metadata registry's (International)
Z (ISO/IEC 13568:2002)	Z Formal Specification Notation — Syntax, Type System and Semantics
RDF	Standard for data interchange (International - semantic web)
OWL	Standard for a web ontology language (International - semantic web)
SKOS	Standard for knowledge organisation (International - semantic web)

1.5 Thesis Structure

Chapter 2 introduces the language and concepts of model-driven engineering, the formal technique, and some aspects of the tools needed for similarity and semantic measurements.

Chapter 3 investigates the existing, de facto metadata standard ISO11179 and explains what is wrong with it—that is, why it is not the metadata language that we need. Relevant sections of the standard have been reproduced in Appendix A.

Chapter 4 gives is an account of our formal analysis of metadata interoperability issues. The analysis leads to the foundations of a metadata modelling language and the associated tools/registries.

Chapter 5 then presents the domain specific language which will address the shortcomings of the ISO standard while also supporting the approach to correctness and the core requirements.

Chapter 6 shows how the language has been applied to the description and comparison of real-world data standards and datasets. It also highlights how descriptive metadata can be handled within MDML.

Chapter 7 shows how it can be used to support data management through matching and linking diverse datasets and standards.

Chapter 8 completes the dissertation with a set of conclusions and an exploration of future research directions.

Chapter 2

Background

This chapter introduces the areas of study which have formed part of this thesis, in particular a brief introduction of Model Driven Engineering, Z, Information Retrieval and Health data engineering.

2.1 Model Driven Engineering

The term 'model-driven' was used by the Object Management Group (OMG), the standards body behind the UML [68], to indicate a degree of automation in the use of models: that is, models being processed automatically to produce some software artefact or to deliver some service. A particular approachm to model-driven engineering [54], called the Model-driven Architecture (MDA) was promoted by the OMG, based upon the Meta Object Facility (MOF) [69] used to define the UML language.

MDA was promoted as 'using models as source code', but did not achieve any significant adoption in software development. One reason for this is that the UML language, originally designed for the description of classes and relationships in object-oriented programs, is not well-suited to the description of behaviour. The various 'dynamic' semantics for UML are no more abstract, and considerably less familiar, than the corresponding constructs in modern programming languages: there was little to be gained from automatic generation.

However, the UML and the MOF are well-suited to the description of data structures and relationships, and the resulting models can be used for the generation of software artefacts pertaining to data acquisition, management, and exchange: for example, database designs, message schemas, and data forms. This is precisely the kind of model-driven engineering needed for semantic interoperability.

UML

The parts of UML [76] that most concern us here are the core concepts of classes and relationships. A *class* is a template for a set of objects or instances. It may have data attributes: in objects of a class, these may take values from primitive types such as integers, or represent links to other objects. It may also have operation attributes: these correspond to software functions that may be called upon objects of the class in question.

The most important *relationship* between classes is that of association. A data model may comprise many different classes, partitioning the description of data into a number of different, associated parts. An association between classes may have a 'multiplicity' constraint, limiting the number of associated objects to a particular number range. An association may be decorated (using a solid diamond at the source end) to indicate that it represents a *composition*: in this case, the associated objects are considered part of the current object, and have no independent existence.

Another important relationship is that of inheritance, extension, or *generalisation*. In a class diagram, this is indicated with a hollow arrowhead at the target end of the line. One class is a generalisation of another if it includes all of the attributes of that other class. There are different interpretations for inheritance—Meyer [56] lists more than a dozen—but the differences are concerned primarily with the interpretation of operations or behaviour; in terms of the description of data, inheritance or generalisation will always involve the extension of the list of data attributes, together with the possible imposition of additional constraints.

A major shortcoming of UML is the lack of any clear concept of *model* or *component*. Both of these terms are used in the UML standard [68], but without explicit definitions or motivating examples. This is particularly problematic for model-driven engineering using UML:

- automatic generation of artefacts at a lower level of abstraction requires inference and imputation; in many cases, this will require in turn a consideration of the sum total or closure of constraint information in a model or component;
- software engineering involves multiple models—or, at the very least, multiple versions of the same model—and multiple components; without the ability to manage collections of classes as models or components, and without the ability to compose components, the utility of the language is in question.

In our development of a language for semantic model-driven engineering, we will need to extend the concepts of UML with a precise notion of *model*.

MOF

The MOF [66] has a small set of core entities, shown in the class diagram of Figure 2.1. The ones that most concern us are the *class*, which will be used to define classifiers in languages, and *property*, used for attributes and relationships.

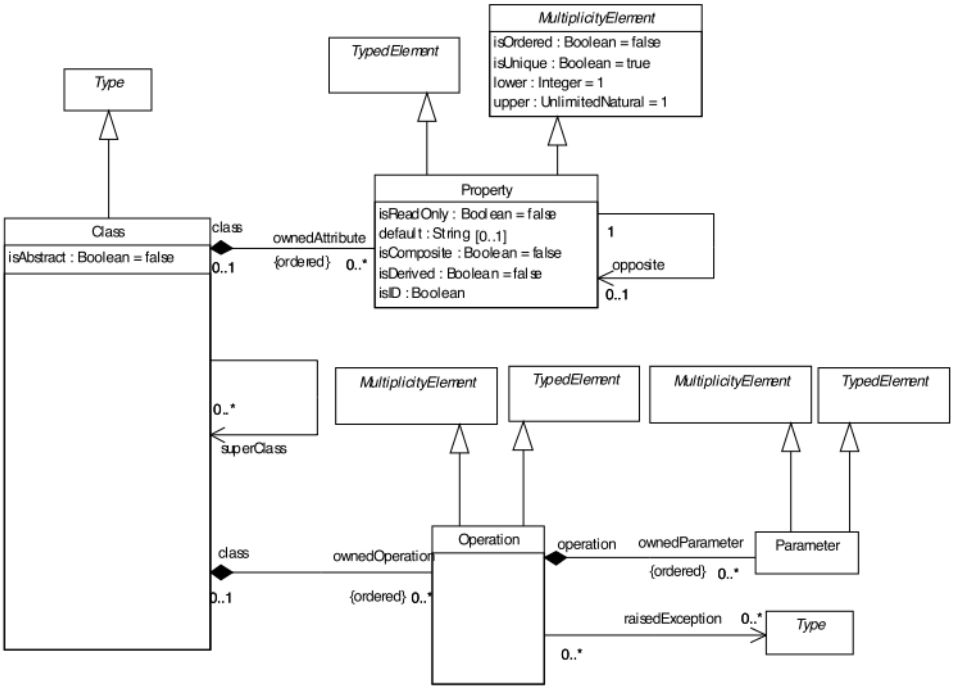

Figure 2.1: Essential MOF

This small set of entities is defined at the top level ('M3') of a four-layer framework, shown in Figure 2.2. It can be used to describe the metamodel or grammar of the UML language itself, which exists at the next level down ('M2'). That grammar, in turn, can be used to describe models written in UML, entities of the level below that ('M1'). These models describe actual data or software instances, entities of the bottom layer ('M0').

The Interface Definition Language (IDL), also shown in the diagram, is a notation intended to facilitate communication between objects implemented in different languages. The OMG's own language was informed by enterprise usage of the CORBA (Common Object Request Broker Architecture) [31]. Like UML, it has no notion of external semantics, provenance, or intended interpretation for the data to be exchanged.

The intention was that MOF should support any kind of metadata, organised in layers 'above' the data in question, which would exist at Level M0. From an OMG perspective, the data would be organised into objects, and the metadata would be organised into meta-objects.

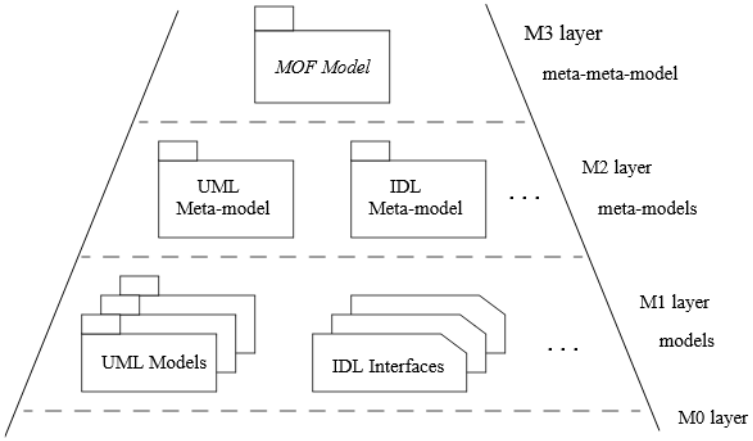

Figure 2.2: MOF four-layer Framework

So a data object at Level M0 could be an instance of a class at Level M1, and that class can be seen as a meta-object, an instance of a meta-class defined at Level M2.

In data description terms: M0 is the data layer; M1 is the model layer; M2 is the modelling language layer; and M3 the metalanguage layer—the layer containing the language in which modelling languages are defined. In UML, the notions of class and property are used at Levels M1, M2, and M3: the language is defined in terms of the core language concepts. However, the four-layer approach advocated for MOF could be applied equally well to other notations.

Indeed, the term *metamodel* is described in MOF [66] as *an abstract language for some kind of metadata*. One can view Extended Backus-Naur Form (EBNF) as being a metamodel for a programming language such as Java, and place EBNF at Level M3. The Java language itself would exist at M2, a program written in the language would exist at M1, and a running instance of the program would exist at M0: see Figure 2.3.

Similarly, we could position the RDF schema language at Level M3, OWL at M2, and an OWL model at M1. OWL, and the other semantic web notations, lack any clear separation between the language of classes and the language of instances, and we have omitted any M0 entity here, instead extending the Java instance to indicate that it could also map to an OWL model, for a given application.

As an example of the location of data entities within the MOF framework, consider the data that might be associated with an airline reservation system: see Figure 2.4. At the lowest level, M0, we have the the actual reservation data itself, represented in the diagram by a list of reservation IDs. The table containing this data would be defined at the level above, M1. This definition, the model or schema for the data, is metadata: describing the structure of the data and giving its intended type.

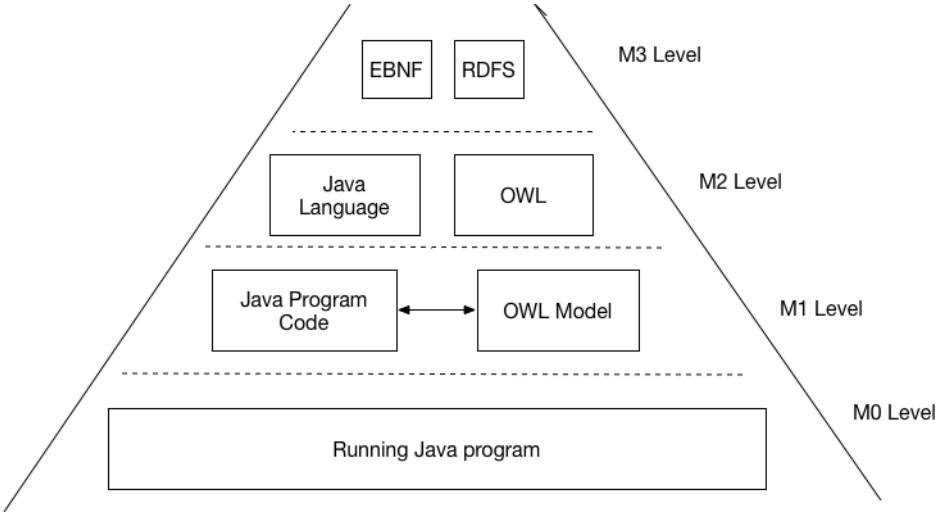

Figure 2.3: EBNF and OWL in the MOF Framework

The language in which this metadata is expressed—the language of the model or schema—is defined at the next level, M2. In this case, the language is not UML but a relational database description language such as SQL. This database description language is defined in turn at the next level up, using the core concepts of MOF: although these are defined as classes and attributes, we have written 'MetaClass' and 'MetaAttribute' in Levels M2 and M3, to emphasise that they are not the classes (tables) and attributes (columns) of Levels M0 and M1.

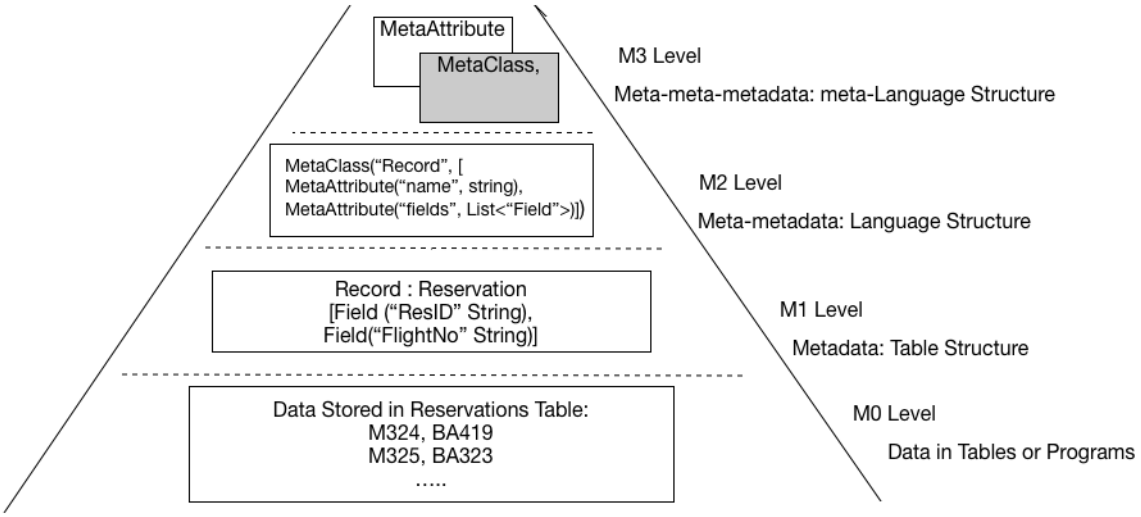

Figure 2.4: Example: MOF instances

Domain Specific Languages

The concept of a domain specific language (Domain Specific Language (DSL)) has been explained as:

> *A computer programming language of limited expressiveness focused on a particular domain* [25]

with the following intentions:

- the language should be used for programming by humans—it should form part of the human interface to the technology;
- it should not be a general-purpose language, but should instead be focussed upon one particular aspect or area of development.

In this thesis, we will be introducing a DSL for semantic model-driven engineering.

Such a language may be defined in terms of concrete and abstract syntax. The concrete syntax comprises the actual keywords, punctuation symbols, and other structural devices (spacing, alignment, or even graphical structures) used in the language implementation. The abstract syntax may be seen as a defining data structure or model for programs written in the language. While some refer to the abstract syntax as the 'semantic model' for the language [25], the notion of semantics implied is rather more limited than the one that we consider here. We may instead refer to the abstract syntax as a metamodel for the language, in the sense of the MOF hierarchy.

In the 'parser-based' approach to language definition, an abstract syntax will be derived from a grammar. This is the approach used for the Ecore metamodel that underpins that Eclipse toolset [20]. A parser will populate an abstract syntax data structure from the programme source, breaking a stream of characters into keywords (such as tokens, literals, identifiers, or comments) and mapping these into a data structure. A grammar provides the production rules needed to do this: for example, the rule

```
<text> ::= "" | <character> <text>
```

tells us that a text string may be the empty string, or it may consist of a character followed by a text string.

2.2 The Z Notation

In chapter 4, we will use a formal, set-theoretic notation to formalise and derive key requirements upon our language of models. The notation in question, Z [90], has been applied in the design and development of large software systems [13, 28, 40]. It is particularly suitable for the definition of formal, denotational semantics: it allows more concise descriptions, through the naming and re-use of patterns of declaration and constraint, and it is supported by a range of open-source tools.

In the Z notation, we write $[S, T]$ to introduce S and T as basic sets or given types of symbols. We write $\mathbb{P}\, S$ to denote the set of all subsets of a set S, and we write $S \leftrightarrow T$ and $S \nrightarrow T$ to denote the set of all binary relations between S and T, and the set of all partial functions from S to T, respectively.

If r is a function or relation, we write: $\operatorname{dom} r$ to denote the domain of r, as the set of all elements a in s for which there is at least one element b in t such that the pair (a, b) appears in r; $\operatorname{ran} r$ to denote the range of r, as the corresponding subset of T; and $r^{\sim}$ to denote the inverse of r.

In addition, if A is a subset of S and B is a subset of T, then we write: $r(\!|\ A\ |\!)$ to denote the relational image of A under r, being the set of all elements t of B for which there is at least one element a in S such that the pair (a, b) appears in r; and $r \rhd B$ to denote the subset of r in which the second component of each pair is an element of B.

Sets may be written in extension using curly braces—{ and }—or as set comprehensions in the format $\{\, decl \mid cons \bullet term \,\}$, where *decl* introduces a set of variables ranging over sets identified in *decl* or already in scope, satisfying the logical constraint *cons*, to produce a set composed of the possible values of expression *term*.

For example, if $S = \{1, 2, 3\}$ and $T = \{3, 4\}$, then

$$\{x : S;\ y : T \mid x < y \bullet (x, y)\}$$

produces the set $\{(1, 3), (1, 4), (2, 3), (2, 4), (3, 4)\}$ containing all pairs of numbers such that: the first component comes from the set $\{1, 2, 3\}$; the second component comes from the set $\{3, 4\}$; and the value of the first component is strictly less than that of the second.

Furthermore, if we define relation r such that

$$r = \{x : S;\ y : T \mid x < y \bullet (x, y)\}$$

then $r(\!|\ \{2, 3\}\ |\!) = r(\!|\ \{2\}\ |\!) = \{3, 4\}$ and $r \rhd \{3\} = \{(1, 3), (2, 3)\}$. To simplify the presentation of sets of pairs, we write $a \mapsto b$ to denote the pair (a, b).

Sequences may be written in extension using angle brackets—$\langle$ and $\rangle$. If s and t are sequences, then we write: *head* s to denote the first element of s; *last* s to denote the last element of s; $s\,n$ to denote the nth element of s, for any number n between 1 and the length of s; and $s \frown t$ to denote the concatenation of the two sequences.

The declaration $a : S$ introduces a variable a as an element of the set S. A declaration may appear as part of a universal ($\forall$) or existential ($\exists$) quantification. For example, we may assert that every element a of S satisfies the property $a < 4$ by writing $\forall\, a : S \bullet a < 4$. A global constant c, drawn from set S, with property p, may be declared as

$$\begin{array}{|l} c : S \\ \hline p \end{array}$$

A free type U, akin to a type union, containing images of sets S and T under injective (1 to 1) functions f and g, may be declared as

$$U ::= f\langle\langle S \rangle\rangle \mid g\langle\langle T \rangle\rangle$$

A schema, as a pattern of declaration and constraint, may be named in 'vertical' form using the following syntax:

Name

$$\begin{array}{|l} \textit{declaration} \\ \hline \textit{constraint} \end{array}$$

or in 'horizontal' form as

$$\textit{Name} \mathrel{\widehat=} [\textit{declaration} \mid \textit{constraint}]$$

Once a schema name has been introduced, it may be used wherever a declaration, constraint, or set-valued expression is expected.

We will include a schema name *SA* within the declaration part of another schema *SB* to introduce the variables declared within *SA*, under the constraint of *SA*. We will use this approach to build up a complex declaration, bringing new functions or properties into scope step-by-step, one schema at a time.

We may write *SC* to denote the set of all objects of 'schema type *SC*'. Each of these objects—often called 'bindings'—has a component or property for each of the variables introduced in the declaration part of the schema. For each object, the combination of values for

these components satisfies the constraint part of the schema. If o is an object of schema type SC, and variable c is introduced in the declaration part of that schema, then we may write $o.c$ to represent the value of component c in object o.

2.3 Information Retrieval

In this work we use various methods for matching and managing data items that have developed from the Information Retrieval (Information Retrieval (IR))domain. Much of the work here is about managing data through registering metadata. One of the key use case is to be able to match or migrate data from one system to another system, or to take data from 2 systems and merge that data for research or other purposes. This is almost the same problem as with the problem of retrieving information, which is centred on measuring the similarity or ranking of a query and a document - for our purposes this can equally well be a term in one dataset to a term in another dataset. The basic TF-IDF ranking algorithm and variants [27, 77] is well established and will be described in the next sub-section.

In the IR world the key use-case revolves around users who are seeking access to content, normally via queries. Content is retrieved by accessing metadata which is common to both the users query and to the document which they are retrieving, and this metadata is stored as an index, so that the user's query is then matched quickly against the indexed term to give access to one or more documents. The indexing process may store both terms and attributes of the documents being stored. In fact both individual words may be stored as indexed, but also the whole vocabulary of a closed system such as a bibliographic database may be curated into a controlled vocabulary or thesaurus, which acts as an initial indexing system, in the same way that most books will have a contents at the front, and a word index at the back. A number of processes may be employed in the indexing process to aid the retrieval stage of the operation, which is in essence a matching process of query with document.

Relevance is a key concept used in IR to measure how useful a query result is, and this is calculated using measures of *recall* and *precision*. If we suppose that there are 100 documents relevant to a query Y, then recall is the number of documents R out of these 100 which are present in the query result, lets say $R/100$, whereas the precision is the proportion of relevant documents R of those returned, so if 50 documents are returned in the search result then $P = R/50$. There is a trade-off between the two, and different users will be interested in different aspects, a researcher may want a higher recall to make sure they find some illusive

research hidden in a document, whereas a user browsing the web may want all the 3 top results to be relevant.

One of the other key measures used in IR, which builds on Precision and Recall is the **f-factor** which is given as

$$F = \left(\frac{(1+\alpha)RP}{(\alpha P)+R}\right)$$

David Powers provides a good overview of these measures, and some of the resultant biases in his evaluation paper : "From Precision, Recall and F-Factor to ROC, Informedness, Markedness and Correlation" [71].This measure is the harmonic mean of recall and precision and uses a parameter α that gives added value to recall as it increases. When $\alpha = 1$, the measure is called F1, and it represents the harmonic mean of recall and precision.

TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (a.k.a. the total number of terms in the document) as a way of normalization, so for nt - number of times a particular term appears in a document, and nt - number of terms in the document:

$$TF(t, d) = (nt)/(nd).$$

IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following where N is the total number of documents in the corpus, and dt is the number of documents which contain the term.

$$IDF(t, d) = log_e(\frac{N}{dt})$$

2.4 Health Data Engineering

There are several reasons why health data presents a particular challenge for semantic interoperability. One of these is the very complexity of the organisms being measured or observed: the health of a human being is determined through the interaction of many factors, at many different scales—from the scale of the human body, and its environment, to that of molecules and microbes.

Another lies in the subjective and contextualised nature of the individual observations, and the way in which observations made at different times, by different people, in different situations must be combined in order to produce an account of the health of a particular individual.

At the same time, the accurate interpretation of this information is particularly important, in both health research and healthcare delivery. In health research, an inaccurate interpretation of data may lead to incorrect conclusions regarding the nature of a disease or the effectiveness of a new drug. In healthcare delivery, it may lead to an incorrect decision regarding the treatment of a patient.

Health informaticians and health data scientists work to achieve semantic interoperability in two ways: by agreeing upon standards for data recording; and by working to standardise data that has already been recorded. Both approaches are needed: agreement is difficult to reach, and the nature and the context of the data being recorded will change over time.

Health data standards exist at two levels: standards for data values, and standards for data models or data sets. The former are exemplified by terminologies such as Systematized Nomenclature of Medicine (SNOMED)-Clinical Terms (CT); the latter by modelling frameworks such as Fast Healthcare Interoperability Resources (FHIR). FHIR (Fast Healthcare Interoperability Resources).

Terminologies

The appeal of a terminological approach is clear: if we have a word, phrase, or number whose meaning is independent of context, then we don't need to worry too much about trying to capture and comprehend contextual information—the rest of the notes, the form, the equipment used, or other aspects of the situation.

Independence from context is difficult to achieve. Fortunately, in practice, it is enough to have a single, shared context—even if this is often left implicit. For example, the terms in the *International Statistical Classification of Diseases and Related Health Problems* terminology, International Classification of Diseases (ICD)-10, are used to record diagnoses, and are not used in any other context. If a particular part of a patient's record contains the ICD–10 code 'E10', then this can be taken to indicate that the patient has been diagnosed with diabetes.

The terms in healthcare terminologies are arranged in hierarchies, with more specialised terms used to capture additional information about the nature of the diagnosis or observation, including information about the context in which the diagnosis is made. For example, in ICD–

10, the code ‘E10.10’ denotes Type 1 diabetes mellitus with ketoacidosis (a harmful metabolic state) without coma, and ‘E10.11’ denotes the same but with coma.

In an attempt to systematise the recording of contextual information, terminologies have been extended beyond diagnoses. For example, the READ terminology used in primary care settings in the UK includes terms for: occupation, social circumstances, ethnicity, religion, clinical signs, symptoms, observations, laboratory tests, test results, diagnostic, therapeutic or surgical procedures performed, and a variety of administrative items.

As more specialised terms are added to a terminology, and as the scope of the subject matter increases, definitions and relationships become more subjective or contentious. For example, the ICD–10 codes ‘W50.0’ and ‘W51’ are used to specify, respectively, diagnoses of “accidental hit or strike by another person” and “accidental striking against or bumped into by another person”. It is hard to imagine that the subtle semantic distinction, if there is one, between these codes is reliably reflected in patient records.

In an attempt to limit the depth of the hierarchies, and to reflect the relationships between different kinds of terms, the developers of the SNOMED-CT terminology introduced the notion of ‘post coordination’, in which two or more existing terms may be combined to describe a new concept—often a specialisation of one of the existing terms. The SNOMED-CT documentation includes an example of this, shown in Figure 2.5.

The example shows how a new term for ‘fracture of tibia’ could be introduced in one of two ways: as a ‘pre coordinated’ term, added to the hierarchy as a new ‘child’ term of ‘injury of tibia’ and ‘fracture of lower limb’; or defined as a ‘post coordinated’ term, having an ‘associated morphology’ of ‘fracture’ and a ‘finding site’ of ‘bone structure tibia’.

This approach has the disadvantage that agreement is now required not only on the ‘is a’ parent–child relationship but also upon the ‘associated morphology’ and ‘finding site’ relationships. All of these relationships are described only at the conceptual level, and may not correspond to the way in which data is recorded or managed in a clinical context.

There remains some prospect of improving semantic interoperability through the use of composite terms, and SNOMED-CT is maintained as a formal ontology using the OWLto facilitate this [82, 73]. However, the vast majority of clinical systems use only pre-coordinated terms, and it seems likely that post-coordination, if it is to be more widely adopted, will be limited to a small number of clear relationships.

A patient record may contain terms from several different terminologies. In the UK, despite the fact that SNOMED-CT has been mandated for use throughout the health service: any primary care data before 2020 will be coded using the READ codes; hospital diagnoses are

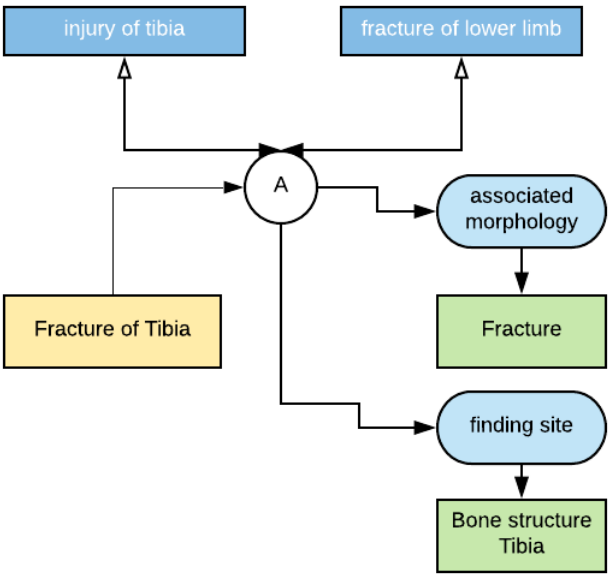

Figure 2.5: An Illustration of Pre- and Post-Coordinated Terms in SNOMED CT

still reported using ICD-10; hospital procedures reported using the OPCS-4 terminology; and laboratory tests are often identified using the Logical Observation Identifiers Names and Codes (Logical Observation Identifiers Names and Codes (LOINC)) terminology. Thus a medical record potentially can end up with 3 or more codes for the same concept. Each code set will be compiled by experts from a particular domain, LOINC is maintained by experts in laboratory reporting, whereas SNOMED is maintained by clinicians and is aimed at more general medical terminology. In most cases a one-to one relationship between concept codes in one domain and other will exist, but in many cases a clinician may want to adopt a different concept definition to a laboratory analyst for what appears to be the same concept.

Furthermore, terminologies will necessarily evolve over time, as new diagnoses, new tests, and new procedures are introduced. Although the intended meaning of an existing term may remain unchanged, the usage of that term may change as other terms are introduced into the terminology. For these reasons, it is essential that any scalable, engineering approach to semantic interoperability includes the facility to manage and map between groups of terms from different terminologies.

Datasets and data models

Standards are defined also at the level of datasets and data models. Although these two terms may be used interchangeably, the former usually denotes a data specification aimed at a

particular reporting requirement, whereas the latter may be more generic—not only in terms of purpose, but also in terms of the data definitions included.

An example of a standard dataset is the 'Cancer Outcomes and Services Dataset' Cancer Outcomes and Services Dataset (COSD), defined by Public Health England. This contains a mixture of contemporaneous and summary data points, giving details of the care delivered to individual patients with cancer. Some of the data points describe the cancer diagnosis, others aspects of the treatments provided.

An example of a standard data model—or a standard data modelling approach—is the OpenEHR standard for electronic health records, formalised by the international standard ISO 13606 [34]. This standard provides a overarching, generic 'reference' model, a set of archetypes or smaller, generic data models aimed at different kinds of patient data, and a set of templates—instantiations of those archetypes to address particular situations.

The 'reference model' of OpenEHR—see Figure 2.6—sets out the core classes and concepts to be used in constructing archetypes, including:

- FOLDER: a grouping to gather data relating to a clinical event
- COMPOSITION: a set of information related to one clinical encounter
- SECTION: a classification of information in a COMPOSITION
- ENTRY: a single action or observation
- CLUSTER: a method of grouping multi-part data structures (such as a time series)
- ELEMENT: a single data attribute, associated with a particular data type

The resulting common structure makes it easier to relate and combine information recorded against different archetypes.

The dominant agency in healthcare data standards has been Health Level 7 (HL7) an organisation that has promoted first messaging standards, and then generic data modelling approaches, over the past three decades. The key contributions from HL7 have been

- HL7 Version 2: a basic specification for medical messages and transactions, still widely used across healthcare
- HL7 Version 3: a more ambitious standard for structured messaging, with a universal 'reference implementation model'

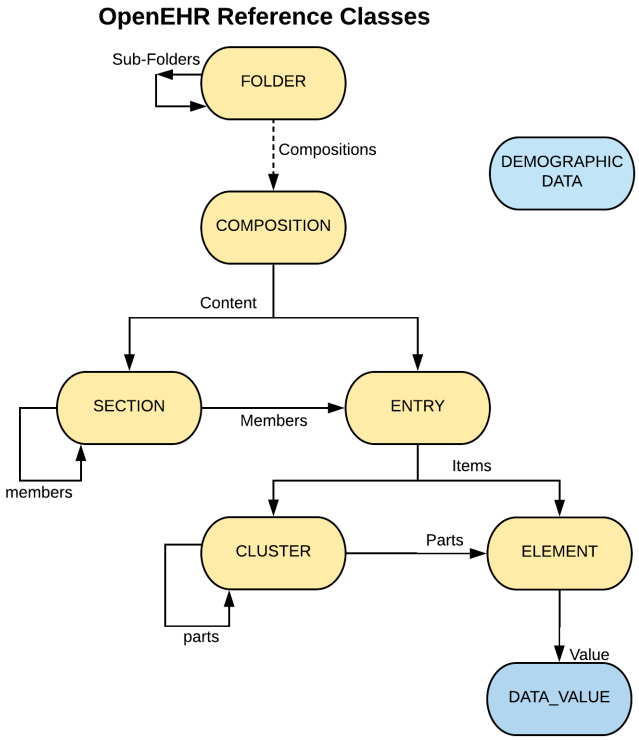

Figure 2.6: Core classes in the OpenEHR Reference Model

- FHIR, or Fast Healthcare Interoperability Resources, an approach to the modular representation of patient records

HL7 Version 2 was criticised for the lack of any consistent, intuitive design, making it difficult to learn or implement. HL7 Version 3 addressed these criticisms through the adoption of a reference model based upon Speech Act Theory [21]: the idea that a statement can be issued in terms of a request, a command, a promise, or a question by changing the way in which the words are spoken, or by changing the context. Every model fragment would be a specialisation of one of four core classes: Act, Entity, Role, and Participation. In practice, HL7 Version 3 proved more problematic, and less popular, than Version 2.

FHIR is a more pragmatic approach, involving the definition of concrete 'resources' for representing the different kinds of data that may appear in a patient record: standard, generic ways of describing entities such as observations, medications, or patients. The concrete nature of the resources lies in the fact that they are defined as working data structures, accessible via applications programming interfaces.

The approach taken in FHIR is similar to that of Open Electronic Health Record (OpenEHR), although the prospects for success are enhanced by the emphasis upon messages exchanged be-

tween systems rather than records existing within a single system. Every piece of information is considered in terms of how it might be shared and interpreted in another context, and the size and selection of the core resources has benefitted from the experience of earlier HL7 developments.

Health Research Data

There are two kinds of health research data. The first kind, which we may see as more traditional, is data collected on questionnaires and case report forms. The questions asked are often quite specific, and may be unique to the individual research project or clinical study for which the data is being collected. The second is data extracted automatically from patient records, whose definition and collection is quite independent of any particular project or study.

The challenge of semantic interoperability arises in both cases, but at different points in the proceedings. In the first case, the original researchers are free to choose their own definitions for source data, within the confines of their own project or study. Interoperability is a secondary concern: something to be addressed when their work is compared with that of others.

In the second case, researchers must work with data that others have collected. They cannot choose the definitions of their source data: they must instead strive to understand what these are; they must also determine if and how the data needs to be transformed to address the scientific questions that they have in mind. This is a familiar problem: data science researchers can expect to spend up to 80% of their time cleaning or 'wrangling' data for analysis [72].

In traditional clinical research, a lack of semantic interoperability, resulting from a failure to coordinate the design of questionnaires or case report forms, limits the extent to which data from different studies can be combined to gain greater statistical power and increase the certainty and clarity of research results. Without this, many studies are inconclusive, and there is considerable duplication of effort.

In big health data research, a lack of semantic interoperability in healthcare systems, across organisations and over time, limits the extent to which data from different patient records can be reliably re-used for research purposes. This has been clearly illustrated in Moritz Lehne and colleagues report *"Why digital medicine depends on interoperability"*, published in Nature [47]. Furthermore, a failure to coordinate the 'wrangling' undertaken in each project will lead to a lack of semantic interoperability between the resulting, transformed datasets, limiting again the extent to which data from different studies can be combined.

Chapter 3

The ISO11179 Standard

3.1 Overview

This chapter gives an overview of the ISO11179 standard for metadata registries, the background section details some essential parts of its history and application. This is followed by a description of the standard itself, looking in detail at the core metamodel as described in part 3 of the standard. Following that we evaluate the metamodel and its use in managing and curating medical datasets and identify particular weaknesses for managing big datasets.

We show areas in which we had difficulty in implementing the standard, and record observations that inform the development of a revised metamodel in later chapters. The fact that only a handful of companies and organisations implement ISO11179 is almost certainly due to experiencing similar difficulties, it is clear that existing sites such as the National Cancer Institute (US) (NCI)'s CaBig site are experiencing problems when it comes to automate and scale the registry [17].

ISO11179 is the ISO standard governing metadata registries, its purpose and intent is to faciliatate semantic interoperability through capturing the *context* of data. It is proposed that by capturing a number of artefacts around the core data elements, data elements can be uniquely described. As a result if data element A, has the same set of metadata artefacts as data element B, which may be taken from a completely different dataset, then it can be accepted that it is the 'same' data element.

The goal of ISO11179 is common usage of data elements across an organisation, and between different organisations and enterprises, in a manner which is commonly searchable and which uses a common semantic description. This standard defines what a metadata registry is, how a data element can be registered, and how the data consisting of data elements can be

semantically described, named, identified, stored, retrieved, and managed. It provides guidance for establishing metadata description registries, as well as registration, authorization, and maintenance itself.

Some of the ideas behind the ISO11179 standard are informed by Formal Concept Analysis (FCA) which a way of developing an ontology or concept hierarchy from a collection of objects and their properties which builds on the mathematical theory of lattices and ordered sets that was developed by Garrett Birkhoff and others [1]. The central idea is based around the idea that all information systems have *data elements* at their core. A data element is viewed as an atomic particle of data, akin to a word in the English language, something that carries meaning and cannot be broken into a smaller sub-division. Each data element will be part of a larger context, and the full *meaning* of the data element cannot be captured without reference to its context.

Context is broken into several other entities, the key ones being

- data element concepts - which capture the conceptual part of a data element
- value domains - which capture the value associated with this data element concept
- data types - which describe the type of the data element

The standard aims to be *implementation-agnostic*, and so reference to programming or database structures are avoided. UML is used as a language to describe the core data description metamodel, and the conceptual metamodel, which are treated in different sections of the standard. The standard itself is not written in UML, although many aspects of the metamodel are described in UML diagrams. If we are to referrence the MOF framework ideas then the metamodel sits at level 2 of the MOF framework, see Figure 3.1

Some of the healthcare datasets we reference are in level 2, others are in level 1, requiring slightly different transformations to map. There are over 20 different healthcare data standards available in the UK, in this work we mainly concentrate on mapping between ISO11179 and SNOMED,COSD and the National Health Service (NHS) Data Dictionary, between them they cover a representative sample of the key aspects of healthcare data standards that are in use today.

3.2 Background

One of the earliest efforts to apply the principles of ISO11179 in practise was the caBIG initiative by the NCI in the USA [45]; they built a software development kit which allows

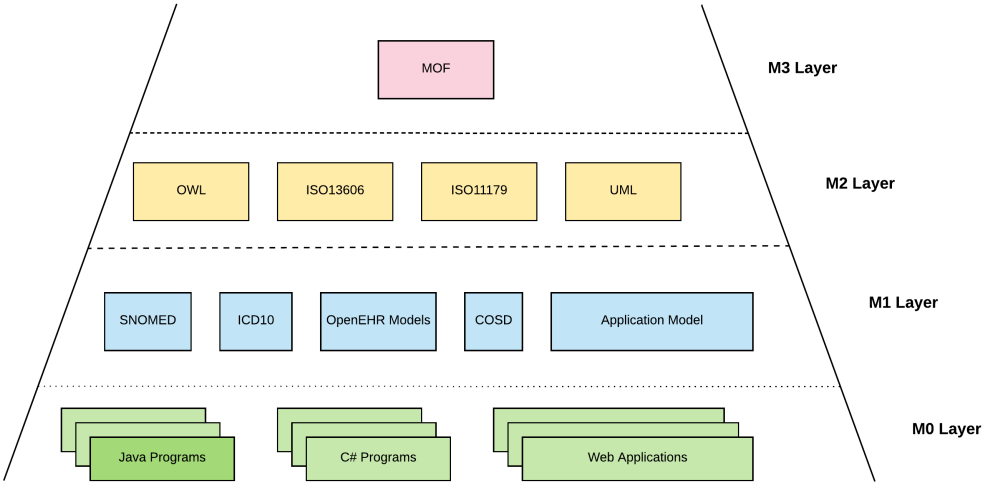

Figure 3.1: ISO11179 in MOF

developers to build web service stubs around data elements, [44], however it does not appear to have been widely adopted. Indeed there are very few examples of ISO11179 metadata registries in practise, one study has used semantic web technology to integrate metadata registries. Sinaci and Erturkmen [79] describe a semantic metadata registry framework where Common Data Element (CDE) are exposed as Linked Open Data resources. CDEs are described using RDF, and thus can be queried using SPARQL Protocol and RDF Query Language (SPARQL) and interlinked with CDEs in other registries using the W3C Simple Knowledge Organization System (SKOS). An ISO11179 ontology has been defined as part of the framework, and the Semantic MDR has been implemented using the Jena framework, although no clear advantages of this implementation have been published.

Metadata Registries, such as those conforming to the ISO11179 standard, can help to solve the problem of data incompatibility, provenance and compliance, as is indicated in studies such as those conducted by Ulrich et al. [30] . In this study a hybrid architecture consisting of an ISO11179-3 conformant metadata registry (MDR) server application for interactively annotating and mediating data elements and the translation of these data elements into FHIR[32] resources was used to manage data for the North German Tumor Bank of Colorectal Cancer.

Tao *et al.* [11] present case studies in representing HL7 Detailed Clinical Models (DCM)s and the ISO11179 model in OWL; a combination of UML diagrams and Excel spreadsheets were used to extract the metamodels for fourteen HL7 DCM constructs. A critical limitation of this approach is that the transformation from metamodels to their ontological representation in OWL is based on a manual encoding. Leroux *et al.* [48] use existing ontologies to

enrich OpenClinica forms. Ngouongo et al. [62] have mapped most of the key datasets standards(SNOMED CT, ICD10, CDISC ODM using the ISO11179-3 metamodel with some degree of success, but have identified a number of shortcomings, which are listed below:

- ISO11179 allows only a single unit of measure, whereas some medical standards allow for multiple measurement units.
- range checks are more sophisticated in standards such as CDISC ODM
- The order of CodeListItem entries in CDISC ODM can only be preserved by using the generic slot extension mechanism
- A CDISC ODM counterpart for data_element_concept, object_class and characteristic could not be found.
- complex data_element_concepts must be recostructed in the logic of an application
- ISO11179 does not support self-referencing associations between instances of the same class on the representation layer
- it remains unclear whether ISO11179 Ed. 3 supports inheritance that is essential in using the RIM-HL7
- Coding standards could not be appropriately transferred to ISO11179 Ed. 3. Remnant issues demand a resolution in the logic of a metadatarepository.
- Some information gets lost each time items or vocabularies are imported from healthcare standards into the metamodel of ISO 11179 Ed. 3
- Mappings for two of LOINC axes: Time Aspect and Method Type were not found. We cannot omit these axes since this would result in structural "equal" data elements having different LOINC codes

Model driven engineering techniques have also been applied to healthcare problems, Schlieter *et al.* [78] record their experience gained from using model-driven engineering to implement an application for path-based stroke care; amongst the lessons learned they recommend using existing ontological models where possible, and being prepared to reconcile a heterogeneity of models from the various stakeholders under a common metamodel. Atanasovski [4] presents a formal meta-model used to specify healthcare process management in an electronic health record system using FHIR and OpenEHR(ISO13606), see [53]. Marcos et al.[51] describes the

implementation of an OpenEHR system to enable interoperability in clinical trial data using OpenEHR archetypes. Archetypes are part of a DSL which in turn is part of the OpenEHR standard, and are described in detail in [53]. The problems with integrating data encoded using different datasets and terminologies are clearly identified by Jian [37], and solutions using OpenEHR technology are put forward in [52], although these solutions do not translate into general solutions applicable over datasets not encoded using OpenEHR syntax. In short whilst OpenEHR provides an effective basic model, like all the other models it is dependent on everybody using it. It lacks an effective meta-model.

3.3 Core Metamodel

Metadata is recorded and registered using a MDR, which is a toolkit that allows definitions of datasets to be stored, curated, managed and linked to data. The problems described here are from one domain: the healthcare data management domain and for most data management purposes the term *metadata* is normally taken to mean data which defines the structure of data. Data about the governance, provenance and other aspects of that data can also be relevant for data management, and can also be included in a metadata registry. By storing the definitions of every data element and all the relationships between data elements in a metadata registry a map of all the data elements and data flows in an organization or domain can be created. Such a *map* can be used to manage, understand and curate the datasets across an organisation rather than just a single application or database.

The ISO standard 11179 is a standard for metadata registries, although this designation is extended in the body of the standard[33] standard. It was issued originally in 2004, and is composed of 6 parts:

- ISO/IEC 11179-1:2015 Framework (referred to as ISO/IEC 11179-1)
- ISO/IEC 11179-2:2005 Classification
- ISO/IEC 11179-3:2013 Registry metamodel and basic attributes
- ISO/IEC 11179-4:2004 Formulation of data definitions
- ISO/IEC 11179-5:2015 Naming and identification principles
- ISO/IEC 11179-6:2015 Registration

The standard is comprehensive, the focus of this work are some key aspects of Part 3 in this research. Part 7 called *Datasets*, has been recently released. This introduces a metamodel for datasets into the standard, however this was not readily available at the time that this research was carried out, and is not considered here. To quote from the standard itself:

> ISO/IEC 11179 addresses the semantics of data, the representation of data and the registration of the descriptions of that data. It is through these descriptions that an accurate understanding of the semantics and a useful depiction of the data are found.

The purpose of ISO/IEC 11179, as described in the standard, is to promote the following:

- standard description of data
- common understanding of data across organizational elements and between organizations
- re-use and standardization of data over time, space, and applications
- harmonization and standardization of data within an organization and across organizations
- management of the components of descriptions of data
- re-use of the components of descriptions of data

Part 3 of the standard provides a registry metamodel, specified using text and UML diagrams, and it was chosen as the starting point for building this implementation. There is a warning at the beginning of Part 3, stating that this part *prescribes a conceptual model, not a physical implementation* This part of ISO/IEC 11179 also prescribes a list of basic attributes (see clause 12) for situations where a full conceptual model is not required or not appropriate. The other 5 parts were used to inform the core metadata registry metamodel as specified in ISO11179: Part 3.

Part 3 - Registry Metamodel and Basic Attributes

The objectives of the metadata registry *metamodel* are defined, in the standard, as:

- providing a unified view of the concepts, terms, value domains and value meanings
- promoting a common understanding of the data described

- providing the specification at a conceptual level to facilitate the sharing and reuse of the contents of the implementations

The standard continues to split the metamodel up into 6 packages, Basic, Registration, Concepts, Binary relations, Data description and Identification, Designation and Definition. In

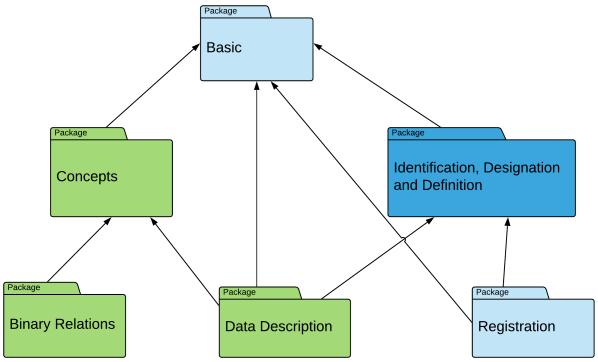

Figure 3.2: ISO11179 Metamodel Packages

section 4 it is pointed out that clauses 7-9 are needed to implement a **Concept Systems Registry**, clause 10 will allow the implementation of an **Extended Concept Systems Registry**, and clause 11 specifies a metadata registry, whereas an **extended metadata registry** will implement all clauses 7-11. Our initial scope was to implement an extended metadata registry as described in clauses 7-11.

Concept Package (Section 9 111179-3)

This package specifies a metamodel for concepts and concept systems, it is closely related to the data description metamodel described in section 11, and many of the classes and artefacts described there are derived from, or are sub-classes of *Concept*. The core model structure is shown in Figure 3.3.

While the inheritance relationship with the Data Description package is illustrated in Figure 3.4, it is also shown in Figure 3.5 which is the core of the data description metamodel. Here we can see that 3 out of the 6 core classes are derived from the *Concept* class. This is important as it provide a means to relate and link different data elements together, as we can link together all classes which are derived from *Concept*. Relation is a class, each instance of which models a relation (ISO11179 - 3.2.119), a sense in which concepts (ISO11179 - 3.2.18) may be connected via constituent relation roles (ISO11179 - 3.2.120). The problem is that when we try and input a set of concepts and relations from a standard medical dataset such as

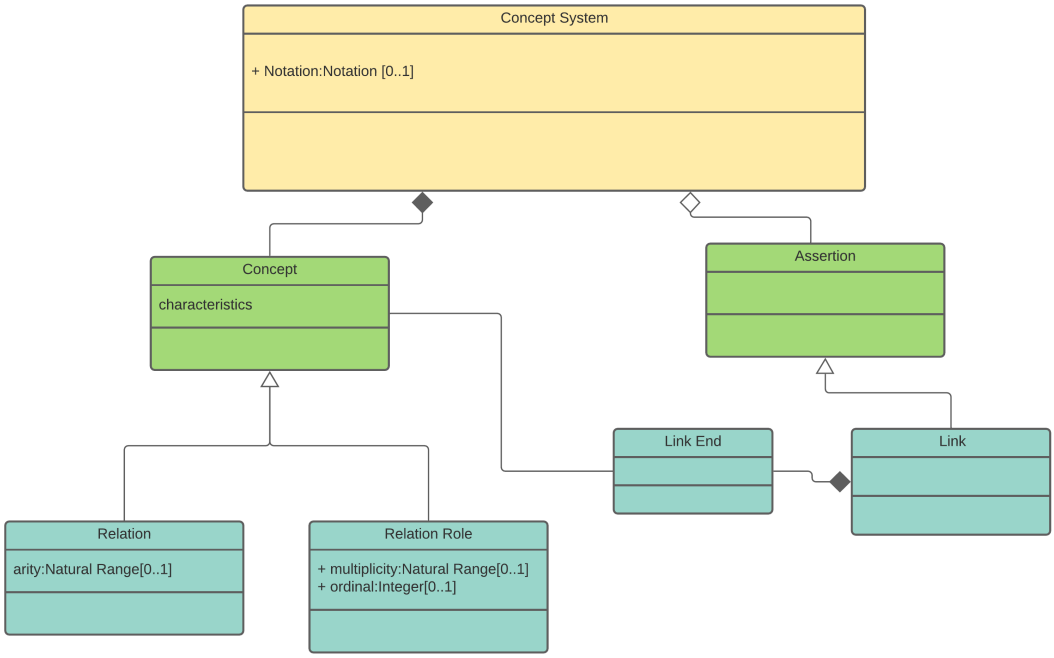

Figure 3.3: ISO 11179 Concept System

COSD it is not obvious what is a relation and what is a concept. When examining a dataset a human reader may distinguish between a *Lesion location* and register it as a concept. She may then interpret a set of numbers 0-98 which describe the location on the body where the lesion occurs as a relation, each code corresponding to a description or definition (e.g. Frontal lobe, temporal lobe, etc). However, there are no obvious rules which would enable a machine to make these kind of semantic differentiations, most of the clues come from the context and this makes the standard difficult to implement without a high degree of expert input.

Data Description Package (Section 11 111179-3)

This package specifies a metamodel for handling *data*, and although it references other packages which are mostly dealing with the more administrative aspects of registering metadata, it primarily puts forward a conceptual metamodel for handling data. Hence for the purposes of data interoperability it is the most relevant part of the standard.

The core model given for data description in ISO11179 is that reproduced in Figure 3.5, it shows the linkage between a Data Element, a Value Domain, a Conceptual Domain and a Data Element Concept. The area above the dotted red line is defined as the *semantic or conceptual* level, whereas the area below the red dotted line is defined as the *representational* level. The

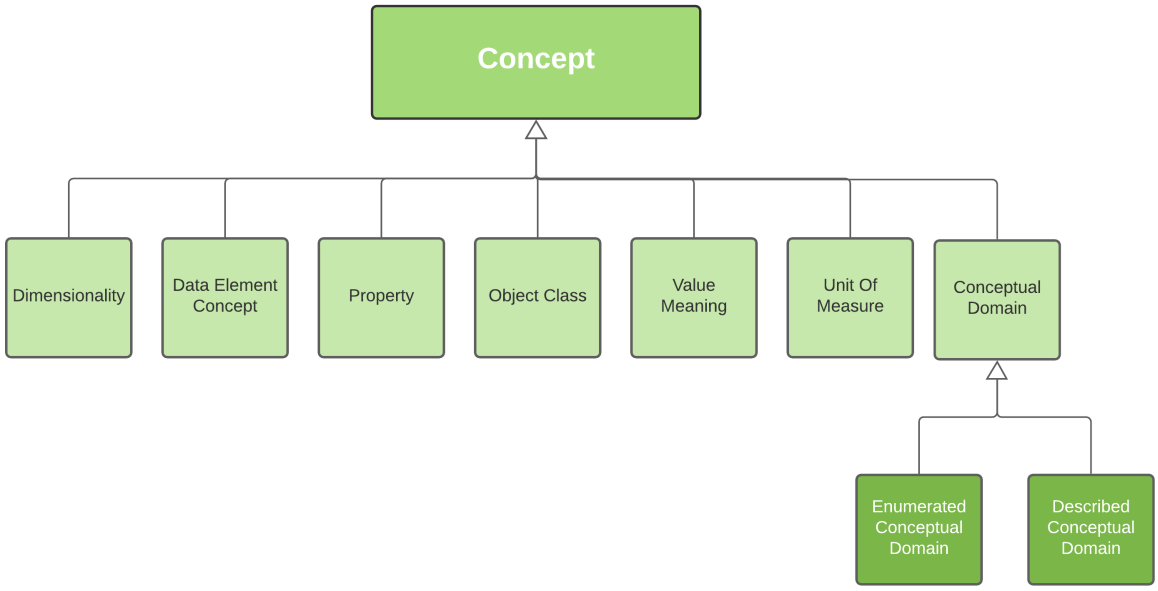

Figure 3.4: ISO 11179 Concept Hierarchy

assumption is that the Data Element and Value Domain are objects which are being registered and classified, as per the processes defined in other parts of the standard.

This arrangement can be illustrated by the idea of a visit to the doctor, we can define a concept called *reason for visit to healthcare centre* and call this a data element concept, and from this we would implement a data element called *reason for attendance* and perhaps represent that with a set of enumerated codes, each representing a different reason. In ISO11179 structuring we would split the data element concept into an object class: Person, and a property: Reason for clinic attendance.

This example shows the intention behind the core ISO11179 metamodel, and it works very well with structures that are defined and created using this model. However what happens if we have a dataset table or tables, and we need to backfill the 6 core entities? Let us suppose that we have a dataset such as that illustrated in Figure 3.6, a table of ethnicity values for instance, then it is easy to fill the representational classes/slots (blue) in the registry. However without access to the original database designer, it is difficult to impossible to work out what metadata should be stored in all the conceptual (green) classes/slots, let alone derive rules for automation. An expert may derive one or two conceptual classes, but getting all four in the manner of the previous example is rare.

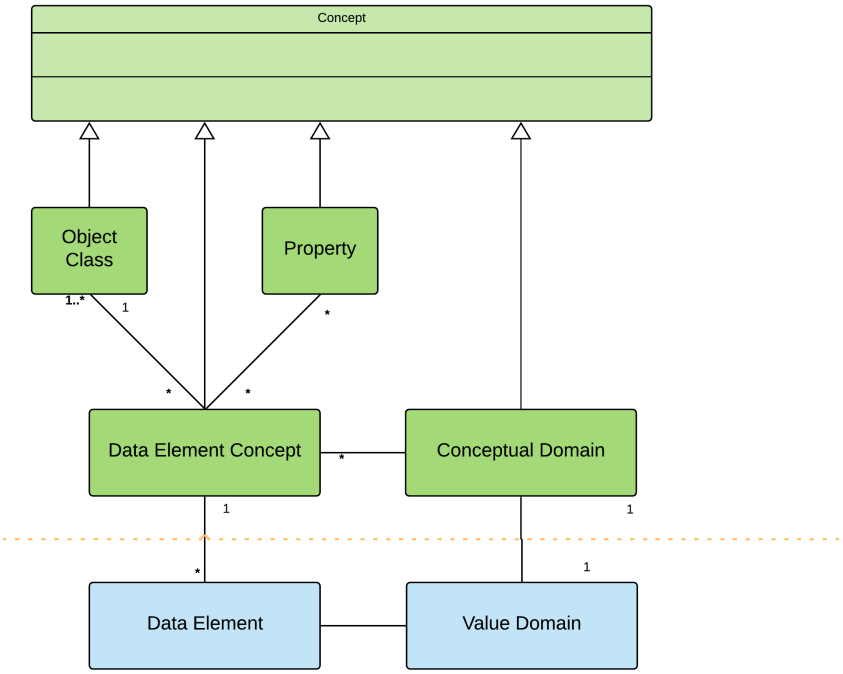

Figure 3.5: ISO 11179 Data Description

SN_Concept	SN_Description	SN_DescriptionType	SN_Term	R2_TermCode	R2_Cod	R2_EffectiveDate	SN_Acti	SNR2_Ma	R2_CodeC	SN_TermC
9238100000010	41291000000111	P	Ethnic category - 2001 census	00	9i...	2004-01-31	1	1	9i	Ethnic category
9239100000010	58341000000117	P	British or mixed British - ethnic category 2001 census	00	9i0..	2004-01-31	1	1	9i0	British or mixed British
4941310000001	1063981000000117	P	White British - ethnic category 2001 census	00	9i00.	2009-10-01	1	1	9i00	White British
9240100000010	41301000000110	P	Irish - ethnic category 2001 census	00	9i1..	2004-01-31	1	1	9i1	Irish
4941610000001	1064041000000111	P	White Irish - ethnic category 2001 census	00	9i10.	2009-10-01	1	1	9i10	White Irish
9241100000010	41311000000112	P	Other White background - ethnic category 2001 census	00	9i2..	2004-01-31	1	1	9i2	Other White background
1107610000001	57281000000117	P	English - ethnic category 2001 census	00	9i20.	2004-01-31	1	1	9i20	English
9254100000010	41431000000111	P	Scottish - ethnic category 2001 census	00	9i21.	2004-01-31	1	1	9i21	Scottish
9255100000010	41441000000119	P	Welsh - ethnic category 2001 census	00	9i22.	2004-01-31	1	1	9i22	Welsh
9257100000010	41461000000118	P	Cornish - ethnic category 2001 census	00	9i23.	2004-01-31	1	1	9i23	Cornish
9256100000010	41451000000116	P	Northern Irish - ethnic category 2001 census	00	9i24.	2004-01-31	1	1	9i24	Northern Irish
9392100000010	42691000000116	P	Ulster Scots - ethnic category 2001 census	00	9i25.	2004-01-31	1	1	9i25	Ulster Scots
9279100000010	41661000000115	P	Cypriot (part not stated) - ethnic category 2001 census	00	9i26.	2004-01-31	1	1	9i26	Cypriot (part not stated)
9393100000010	42701000000116	P	Greek - ethnic category 2001 census	00	9i27.	2004-01-31	1	1	9i27	Greek
9394100000010	42711000000119	P	Greek Cypriot - ethnic category 2001 census	00	9i28.	2004-01-31	1	1	9i28	Greek Cypriot
1104010000001	56921000000110	P	Turkish - ethnic category 2001 census	00	9i29.	2004-01-31	1	1	9i29	Turkish
9395100000010	42721000000113	P	Turkish Cypriot - ethnic category 2001 census	00	9i2A.	2004-01-31	1	1	9i2A	Turkish Cypriot
9396100000010	58481000000115	P	Italian - ethnic category 2001 census	00	9i2B.	2004-01-31	1	1	9i2B	Italian
8891100000010	13817000000114	P	Irish Traveller - ethnic category 2001 census	00	9i2C.	2004-01-31	1	1	9i2C	Irish Traveller
8892100000010	13818000000111	P	Traveller - ethnic category 2001 census	00	9i2D.	2004-01-31	1	1	9i2D	Traveller
8893100000010	13819000000113	P	Gypsy/Romany - ethnic category 2001 census	00	9i2E.	2004-01-31	1	1	9i2E	Gypsy/Romany
8894100000010	13820000000110	P	Polish - ethnic category 2001 census	00	9i2F.	2004-01-31	1	1	9i2F	Polish
8895100000010	13821000000112	P	Baltic States (Estonian or Latvian or Lithuanian) - ethnic c	00	9i2G.	2004-01-31	1	1	9i2G	Baltic States (Estonian or Latvian or Lithuanian)

Figure 3.6: Dataset Input

3.4 ISO Metamodel Evaluation

The next section describes the initial evaluation process for registering 3 standard healthcare models to the ISO11179 registry metamodel artefacts, all three attempts have similar unsatisfactory outcomes. This prompted the redesign and specification of the core metadata registry using the principles identified in chapter 4, and the metamodel described in chapter 5.

SNOMED, NHS Data Dictionary (NDD) and COSD were chosen as commonly used dataset standards in the NHS, the first uses OWL as its core language, the NDD is published using XML Metadata Interchange (XMI), and COSD is published in Excel. These datasets illustrate the diversity in which data standards are specified and provided in the NHS.

COSD

The COSD is typical of the datasets used in the NHS, it is centrally curated by a team of people, and every 3-6 months a new version is published, with recent additions and often a few changes or deprecations. The dataset itself is used to collect and monitor cancer outcomes in the UK, and allows NHS England to monitor different outcomes in different trusts, to ensure a uniform high quality treatment across the glsnhs. The standard is published as an excel spreadsheet, or an eXtensible Markup Language (XML) file, we focus here on the excel version. The COSD dataset we use is specified as an excel spreadsheet which can be downloaded from NHS Digital[63], most of the work carried out in this research used version 6.0. Difference sections are divided by different tabs, and each tab contains columns and rows containing the details. The terms are defined by using headers, so that a particular data element's details, id/code, name, description, etc will be shown as a list of headers stretching horizontally across the top of the page. This is illustrated in the extract in Figure 3.7.

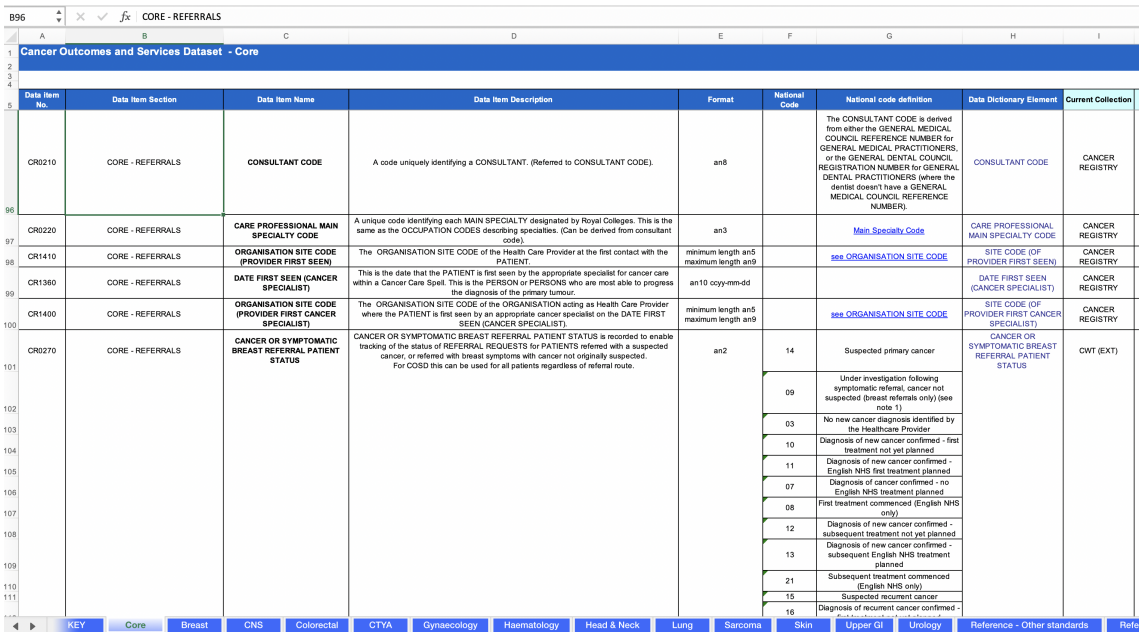

Figure 3.7: Excel Specification

One row is used to identify a data elements, with various aspects indicated in different columns, for instance her we have a constraint on the value domain of the element in the column *format* which uses a code to denote the structure of the value used by that data element. For example **an8** indicates that this item, say a consultant code, needs to be alphanumeric character and have at most 8 characters. The data elements pertaining to a particular class or classification are held in one tab, so we have tabs which are classified by the type of cancer;

Central Nervous System (CNS) ,Breast, Colon, etc. Data elements can be further classified by using the various attributes, captured in the header names, so **Data Item Section** would be an appropriate heading to use as a sub-classification header.

Deciding which terms and data elements to include in a dataset like this, how to group the data elements into hierarchies, what formats and data types to use is a process known as curation, and derives from the term that museums use to curate artefacts. The problem with this kind of data curation is not that it cannot be done, but that it is very easy to get confused between versions, especially when several people in different locations are involved. Spreadsheets are not uniquely identified, nor are cells or tabs within spreadsheets, there is no inate mechanism for versioning between different versions of the dataset or data model. Nor is there any way to differentiate a version of COSD which is being updated. Using a metadata registry to carry out this task makes sense as it ensures that versioning and naming issues can be eliminated.

Column Header	Description
Data Item No.	A reference number for internal use
Data Item Section	The Section of the dataset, generally related to the patient pathway.
Data Item Name	The name of the data item.
Data Item Description	A description of the data item. (Further details will be provided in the User Guidance).
Format	The format in which the data item should be submitted.
National Code	The code to be used when the data item is submitted.
National Code Definition	The definition of National Code i.e. what the code means. **NB:** *Not Known = not recorded or test not done.*
Data Dictionary Element	The name of the data item as recorded in the NHS Data Dictionary.
Current Collection	Current dataset in which this data item is collected. NB: Where data item is collected in more than one dataset, only one may be listed.
Greyed Data Items	Not directly flowed from providers and not included in the schema. Item is already listed elsewhere or is not flowed directly from providers as part of COSD.

Figure 3.8: COSD Key Headers

In COSD the first tab of the spreadsheet defines the columns/headers, and rows in the other tabs, as shown in Figure 3.8. These define the columns present in the other spreadsheet tabs, most of which are grouped by sub-disease classification, e.g. Breast Cancer, Cancer of the Central Nervous System, etc, while some others relate to general topics, e.g. "Core" - items relating to general diagnosis, Morphology, Imaging details, and references to other standards and sources. Each of these headers represents an aspect of the dataset, and thus if we are to store this information model in an ISO11179 registry we need to map the headers to ISO11179 constructs, which we are defined in Appendix A.

The first stage therefore is to define the mapping, the second stage is to input the metadata contained in the spreadsheet and store it in the metadata registry. So for each ISO11179 construct we make a comparison, and attempt a mapping for each construct: Data Element Concept, Data Element (see A.1.3), Data Element Derivation .

Mapping: Data Element Concept

The key definition in ISO11179 (part 3 section 3.3.29, and Appendix A) defines a Data Element Concept as being a *concept that is an association of a property with an object class*, although later on this definition is augmented by the statement that a Data Element Concept is a concept that can be represented as a *data element*, so that a data element can be thought of as a concrete instantiation of the abstract Data Element Concept.

The data element is represented here by a row of data, and arguably we could use the description as a Data Element Concept, however it is not really an abstraction of the data element concept, it is extra information which describes the data element. Consider the data element in Figure 3.9, the data element called *Local Patient Identifier*, the description is not really an abstract or conceptual aspect of the data element, it is a decription of how a patient identifer is used. We can take the name and description as the name and description of the *data element*, but as we have no other reference this also forms the basis of the *data element concept*. Taking the example of *Local Patient Identifier* we could presume that a data element called *Local Patient Identifier* is related to a *data element concept* called *Patient Identier*, or even *Identifier*. However deciding this involves a judgement call by someone familiar with the codeset and its purpose. Such knowledge cannot be encoded into an algorithm or rule that can be used by a program carrying out the metadata analysis. As a result we cannot make an automatic mapping from a COSD data construct to an ISO11179 construct.

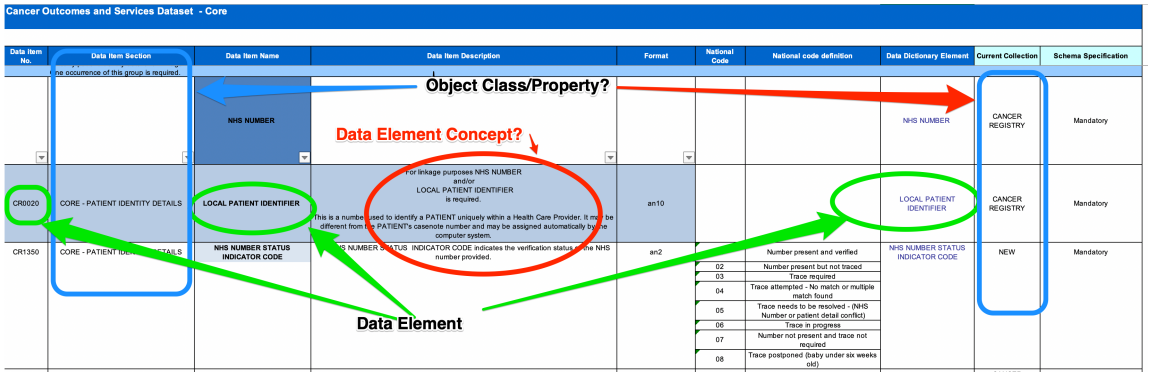

Figure 3.9: Suggested Mappings - Part 1

Mapping: Data Element

In Figure 3.9 we can easily identify the core data element as being "Local Patient Identifier", and would expect that the data item number, Name, description at least be attributes of the data element.

Mapping: Data Element Derivation

A Data Element Derivation is a rule which can be applied to a data element to derive another data element. There is no obvious candidate here

Mapping: Derivation Rule

A Derivation rule is the rule which can be applied to derive a new data element from an existing data element. However there is no obvious candidate here.

Mapping: Object Class

From Appendix (A) we see that an object class is a subclass of Concept, it represents a set of ideas, abstractions or things in the real world that can be identified with explicit boundaries and meaning, and whose properties and behaviour follow the same rules, it may either be a single or a group of associated concepts, abstractions or things.

With COSD we have a rows of "data elements" which have "properties" such as name, descriptions, format etc. However it is hard to make a direct mapping from Object Class to any of the properties. Candidates, from the section shown in Figure 3.9 would be *Data Item Selection* and/or *Current Collection*, as these are clearly aspects which are common to many data elements. We will select *Current Collection* for now as it appears to relate to collections or classifications of data elements which would be nearer the intention of the object class construct.

Mapping: Property

This construct is modelling a property common to a number of data elements, so looking at Figure 3.9 again we are left with *Data Item Selection* which appears to be the more reasonable choice, since the values can be construed as properties of the data elements.

Mapping: Conceptual Domain

From Appendix (A) this is describing a set of value meanings, however there is no obvious way in which this set is defined. Without a clear definition we simply cannot automate. We could use the tabs, to arrive at conceptual domains for each sub-category of disease, however some of the tabs relate to completely different concepts, and although we may be able to map the tabs to different conceptual domains there is no machine-readable way to do this. An initial analysis indicates that we can have 2 maybe 3 conceptual domains here, but training a machine to recognise which tabs belong to which conceptual domain would not be possible without either a very large training dataset, or some kind of higher ontology which could be used as a reference. If there was such an ontology then we would not need to carry out this exercise.

Mapping: Described Conceptual Domain

This is defined as *a conceptual domain that is specified by a description or specification, such as a rule, a procedure, or a range (i.e. interval).* We can see no obvious described conceptual domain in the COSD dataset, there are rules defined for formating and arguably typing the data, but they are not defining a conceptual domain. Instead they simply defining the way the values are structured. The problem here is that if we cannot map to a conceptual domain, then we lose the ability to use the semantic meaning to guide the interoperability between different datasets. If we lose the ability to point at or reference the context then we lose the meaning, and we are unable to link two data elements from heterogeneous models. The standard therefore loses it's utility, we cannot automate. Also, there are two ways in which a hierarchy can be defined using the standard, firstly through the object class and property attributes in the data description models, and secondly by using the conceptual model. The latter is more flexible, however if we cannot map to a conceptual domain we lose one of the ways to map hierarchies of data elements, which are very common in helathcare datasets, again the standard loses its utility if these conceptual elements are lost.

Mapping: Value Domain

The value domain can be taken from the National Codes heading, with the format (an attribute of Value Domain) being used both to describe the format attribute and the description of the value domain.

Mapping: Enumerated Value Domain

In some case the National Code is split into a number of codes in which case this can be represented as an enumerated value domain.

Mapping: Described Value Domain

In the other cases where only a single national code is described this can be represented as a *Described Value Domain*.

Mapping: Enumerated Conceptual Domain

The *Enumerated Conceptual Domain* can be inferred from the *Enumerated Value Domain*, however they are in essence the same thing, one being a sign which designates the meaning of the other. If we are taking the information from a spreadsheet such as this then this is essentially repeating the information held in the Enumerated Value Domain.

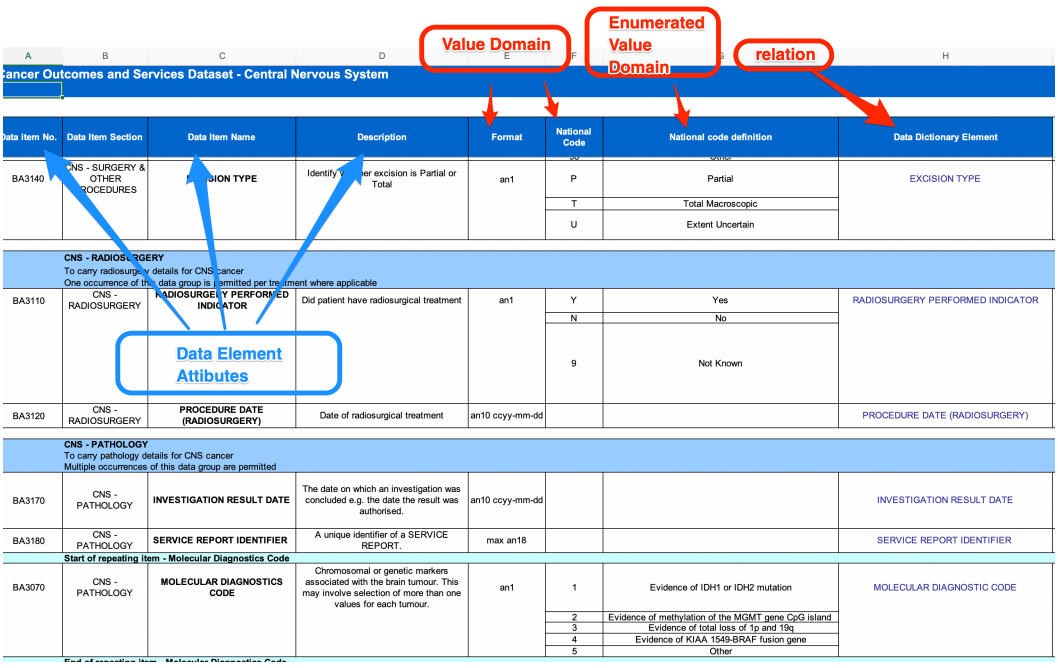

Figure 3.10: Suggested Mappings - part 2

Mapping: Value Meaning

As with the Enumerated Value Domain this is in essence simply repeating information gathered from the Value Domain construct, taken from the National Code header in this example from 3.10

Mapping: Permissible Value

We can argue that the permissible value is the same as the enumerated Value domain, so that where we have a number of codes used to describe the *National Codes* these are permissible values. However in populating a construct in the registry this is essentially the same as describing the enumerated Value domain with each enumeration being a permissible value.

Mapping: Relation

Relations are described in section 10 of ISO11179-3, and any administered item can have a relation with another administered item, the relation can have be transitive, symmetric or reflexive and are binary in nature. So we could identify the heading *Data Dictionary Element* as a relation with another administered item, if we are able to capture the "Data Dictionary" data elements in this registry.

COSD Mapping - Summary

In summary, in our first pass in aligning the ISO11179 metamodel with COSD therefore we have the mappings which are illustrated in Figure 3.1. In the COSD Dataset there are 10 column headers, each one holding metadata about a particular data item. In the direction from COSD to the ISO11179 metamodel, we can map 3 headers (Data Item No, Data Item Name and Data Item Description) onto the Data Element in the ISO model. Part 4 of the ISO11179 standard specifies that a data element has a definition, identication, representation and permissible values specified by means of a set of attributes, so we are able to capture the name, number and description using attributes.

If we limit ourselves to the data definition artefacts in ISO11179-3 then we have no mapping for the Data Item Sections. We were able to use section 9 of the metamodel for classification, which if brought into play with the data definition metamodel (section 11), see Appendix A), solves this problem. By adding a new artefact called classification we can accommodate ways of classifying different data elements without overloading existing artefacts in the way originally suggested. This can also handle the problem with overall naming of the datasets, and the

Table 3.1: ISO11179 - COSD - ISO11179 Mappings

COSD Artefact	**ISO Artefact (v.0.1)**	**Notes**
Spreadsheet Name	Concept System	mapping
Spreadsheet Tab Name	Concept	mapping
Data Item No	Data Element	mapping
Data Item Section	Concept	mapping
Data Item Name	Data Element	mapping
Data Item Description	Data Element	mapping
Format	Permissible Value	mapping
National Code	(Described or Enumerated) Value Domain	mapping
National Code Definition	(Described or Enumerated) Value Domain	mapping
Data Dictionary Element	Relation	mapping
Other Collections	Link	mapping
Schema Specification	Data Element Obligation (attribute)	mapping
...	Object Class	no mapping
...	Property	no mapping
...	Described Conceptual Domain	no mapping
...	Enumerated Conceptual Domain	no mapping
...	Data Element Derivation	no mapping
...	Derivation Rule	no mapping

different tabs in the spreadsheet which refer to different areas of interest. It is also interesting since section 9.2.3.1 of ISO11179 explicitly states that *The exact semantics of the Classification association are not specified by this standards, but will depend in on the way in which this classification scheme is used.*

It is not an exclusive way of defining and managing the data elements, we could find the same data element concepts in other datasets, especially when the tabs are names such as *Blood*. The text item *Blood* will be ingested and a data element concept called *Blood* put into the system, but users tend to want to use a hierarchical classification pertaining only to this one dataset. We can certainly do this manually, however if ingesting the data automatically

it is hard to find an appropriate rule to do this. This system also has problems when the hierarchies are more than 2 levels deep, since the classifications have not been defined as a set of hierarchical data element concepts, and almost inevitably they will not make sense as *Data Element Concepts* in their own right. For now we are able to add in both a *Relation* and a *Concept* to handle the classification element of the COSD dataset.

We can map Format onto Permissible Value, and then the National Code and National Code Description onto Value Domain, either enumerated or described depending on the value. The next two items are links or relationships with other datasets, so these can be handled using the *Relation* artefact, although it is not technically from the Data Definition section of ISO11179-3. Lastly the Schema Specification is metadata about whether the element is optional or mandatory in the dataset, and this can be handled using an attribute of the ISO Data Element artefact.

The other key item which appeared to be missing from the mapping was the ability to add in relationships. Relationships are key to the way in which COSD is used, all the elements relate to elements in the NHS, and so ideally a direct link would be useful from within a metadata registry from a COSD element to an NDD element. At the moment we are not able to easily capture the relationships in the registry using the data description metamodel, and without this we lose interopeability.

Relations are modelled in a separate section of ISO11179-3, section 10, and that model is shown in the Appendix A. By moving this artefact into the Data Description model we can model relations between different data elements, however upgrading the metamodel would make things much more explicit and simple to automate.

It is possible to use ISO11179-3 to manage the dataset, however there are at least 5 key artefacts of the ISO11179-3 data description metamodel which are not used, and several which require non-standard management. These missing artefacts relate to way in which ISO11179 record the *meaning* of the data elements, and hence the core aspects of ISO11179-3 which enable semantic interoperability are not present in the COSD registration. We cannot match up concepts, conceptual domains and data element concepts to arrive at the semantic matching, which is referred to, but not defined within the standard. Another observation is that some of the ISO artefacts are taken from the Concept metadatamodel(section 9), e.g. Relation, whereas others are taken form the data description metamodel (section 11). Relations are illustrated to be between concepts, however we are assuming that this stretched to relationships between data elements, using the data element concept.

The standard's intention is to achieve interoperability by matching conceptual artefacts, and thus enabling interchange. If two data elements belong to the same conceptual domain, then they are related, we can use that relationship to map the instance data. If there are no conceptual artefacts available from the dataset then we cannot relate the data elements to the conceptual domain, and we cannot map the instance data. Thus we cannot achieve semantic interoperability using an ISO 11179 compliant registry by similar inputting the datasets on their own.

NHS Data Dictionary

The NDD [18] is composed using the Darwin Information Typing Architecture system (DITA), and until September 2020 was published using XMI, an excerpt of which is shown in Figure 3.11. Currently the NDD is published using a registry called Mauro, which is largely based on the Metadata Management Language (MDML) metamodel shown here. The work show here predates this event, however since this analysis illustrates the use of UML which is a common modelling languages in use in this sector it seems worth leaving in as the reasoning applies to any UML model. XMI is the XML syntax which is used to transmit and serialise UML models, therefore the NDD is stored and transmitted as a version of UML.

```
<ownedMember xmi:type="uml:Class" xmi:id="_IaTNbwxhEeuXG9qw4_Xuig" name="ACCOMMODATION_STATUS_RECORDED_DATE">
  <ownedComment xmi:id="_IaTNcAxhEeuXG9qw4_Xuig">
    <body>&lt;table border=&quot;0&quot; cellpadding=&quot;1&quot; cellspacing=&quot;1&quot; class=&quot;border&quot; width=&quot;100%&quot;>&lt;tbody>&lt;tr>&lt;td class=&quot;fieldhead&quot; valign=&quot;top&quot; width=
&quot;30%&quot;>Format/Length:&lt;/td>&lt;td valign=&quot;top&quot; width=&quot;70%&quot;>an10 CCYY-MM-DD&lt;/td>&lt;/tr>&lt;tr>&lt;td class=&quot;fieldhead&quot; valign=&quot;top&quot; width=&quot;30%&quot;>National
Codes:&lt;/td>&lt;td valign=&quot;top&quot; width=&quot;70%&quot;>&amp;nbsp;&lt;/td>&lt;/tr>&lt;tr>&lt;td class=&quot;fieldhead&quot; valign=&quot;top&quot; width=&quot;30%&quot;>Default Codes:&lt;/td>&lt;td valign=&quot;top&quot;
 width=&quot;70%&quot;>&amp;nbsp;&lt;/td>&lt;/tr>&lt;/tbody>&lt;/table>&lt;br>&lt;strong>Notes:&lt;/strong>&lt;br>&amp;lt;a
href=&amp;quot;http://www.datadictionary.nhs.uk/data_dictionary/data_field_notes/a/accommodation_status_recorded_date_de.asp&amp;quot;&amp;gt;ACCOMMODATION_STATUS_RECORDED_DATE&amp;lt;/a&amp;gt;&amp;nbsp;is the&amp;nbsp;same as
attribute &amp;lt;a href=&amp;quot;http://www.datadictionary.nhs.uk/data_dictionary/attributes/p/person/person_property_recorded_date_de.asp&amp;quot;&amp;gt;PERSON_PROPERTY_RECORDED_DATE&amp;lt;/a&amp;gt;.&lt;/p>&amp;lt;a
href=&amp;quot;http://www.datadictionary.nhs.uk/data_dictionary/data_field_notes/a/accommodation_status_recorded_date_de.asp&amp;quot;&amp;gt;ACCOMMODATION_STATUS_RECORDED_DATE&amp;lt;/a&amp;gt;&amp;nbsp;is the
date&amp;nbsp;when the &amp;lt;a href=&amp;quot;http://www.datadictionary.nhs.uk/data_dictionary/data_field_notes/a/accommodation_status_code_de.asp&amp;quot;&amp;gt;ACCOMMODATION_STATUS_CODE&amp;lt;/a&amp;gt;&amp;nbsp;was
recorded.&lt;/p></body>
  </ownedComment>
  <ownedAttribute xmi:id="_IaTNcQxhEeuXG9qw4_Xuig" type="_IaKqigxhEeuXG9qw4_Xuig" isUnique="false" association="_IaTNMgxhEeuXG9qw4_Xuig">
    <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_IaTNcgxhEeuXG9qw4_Xuig" value="1"/>
    <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_IaTNcwxhEeuXG9qw4_Xuig" value="1"/>
  </ownedAttribute>
</ownedMember>
</ownedMember>
<ownedMember xmi:type="uml:Package" xmi:id="_IaTNdAxhEeuXG9qw4_Xuig" name="B">
  <ownedMember xmi:type="uml:Package" xmi:id="_IaTNdQxhEeuXG9qw4_Xuig" name="Bay">
    <ownedMember xmi:type="uml:Association" xmi:id="_IaTNdgxhEeuXG9qw4_Xuig" memberEnd="_IaKq1AxhEeuXG9qw4_Xuig _IaTNsAxhEeuXG9qw4_Xuig"/>
    <ownedMember xmi:type="uml:Association" xmi:id="_IaTNdwxhEeuXG9qw4_Xuig" memberEnd="_IaKq1wxhEeuXG9qw4_Xuig _IaTNtQxhEeuXG9qw4_Xuig"/>
    <ownedMember xmi:type="uml:Association" xmi:id="_IaTNeAxhEeuXG9qw4_Xuig" memberEnd="_IaKq2gxhEeuXG9qw4_Xuig _IaTNugxhEeuXG9qw4_Xuig"/>
    <ownedMember xmi:type="uml:Association" xmi:id="_IaTNeQxhEeuXG9qw4_Xuig" memberEnd="_IaKq3QxhEeuXG9qw4_Xuig _IaTNvwxhEeuXG9qw4_Xuig"/>
    <ownedMember xmi:type="uml:Association" xmi:id="_IaTNegxhEeuXG9qw4_Xuig" memberEnd="_IaKq4AxhEeuXG9qw4_Xuig _IaTNxAxhEeuXG9qw4_Xuig"/>
    <ownedMember xmi:type="uml:Association" xmi:id="_IaTNewxhEeuXG9qw4_Xuig" memberEnd="_IaKq4wxhEeuXG9qw4_Xuig _IaTNyQxhEeuXG9qw4_Xuig"/>
    <ownedMember xmi:type="uml:Association" xmi:id="_IaTNfAxhEeuXG9qw4_Xuig" memberEnd="_IaKq5gxhEeuXG9qw4_Xuig _IaTNzgxhEeuXG9qw4_Xuig"/>
    <ownedMember xmi:type="uml:Association" xmi:id="_IaTNfQxhEeuXG9qw4_Xuig" memberEnd="_IaKq6QxhEeuXG9qw4_Xuig _IaTN0wxhEeuXG9qw4_Xuig"/>
    <ownedMember xmi:type="uml:Association" xmi:id="_IaTNfgxhEeuXG9qw4_Xuig" memberEnd="_IaKq7AxhEeuXG9qw4_Xuig _IaTN2AxhEeuXG9qw4_Xuig"/>
    <ownedMember xmi:type="uml:Association" xmi:id="_IaTNfwxhEeuXG9qw4_Xuig" memberEnd="_IaKq7wxhEeuXG9qw4_Xuig _IaTN3QxhEeuXG9qw4_Xuig"/>
```


Figure 3.11: Raw XMI for NHS Data Dictionary

The challenge here was how to represent a UML model, albeit a specific one, in an ISO11179 metadata registry. Some aspects of the XMI schema used were non-standard, however for the purposes of the discussion here we focus on equating UML with the ISO Metamodel.

The key elements that we are looking to match are package, class, association, attribute, enumeration, type and one or two others. This can be seen in Figure 3.12 which shows an excerpt of the same XMI file, but filtered to show the UML types which are used in the NHS Data Dictionary model. It is these particular types which need to be mapped into the ISO metamodel if we are to use the ISO metadata registry effectively. The first item to match is *uml:class*, this is the main construct being used in this model, and it is a construct which is not composed of others, although it is in many cases linked to others. In some cases the linking is hierarchical, which is immediately evident by referring to the online browser version of the dataset.

The next item on the list is *uml:Association*, which in this instance can be easily mapped to a Relation in the ISO metamodel. *uml:Package* is more difficult, but it describes the naming of the domain in which the elements are placed, and thus can be equated to the *Concept Domain*,

```
<ownedMember xmi:type="uml:Class" xmi:id="_laTmqgxhEeuXG9qw4_Xuig" name="LOOKED_AFTER_CHILD_INDICAT...
<upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_laTmrQxhEeuXG9qw4_Xuig" value="1"/>
<lowerValue xmi:type="uml:LiteralInteger" xmi:id="_laTmrgxhEeuXG9qw4_Xuig" value="1"/>
<ownedMember xmi:type="uml:Class" xmi:id="_laTmrwxhEeuXG9qw4_Xuig" name="LOWER_LAYER_SUPER_OUTPUT_A...
<upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_laTmsgxhEeuXG9qw4_Xuig" value="1"/>
<lowerValue xmi:type="uml:LiteralInteger" xmi:id="_laTmswxhEeuXG9qw4_Xuig" value="1"/>
<ownedMember xmi:type="uml:Class" xmi:id="_laTmtAxhEeuXG9qw4_Xuig" name="LUNG_METASTASES_SUB-STAGE_...
<upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_laTmtwxhEeuXG9qw4_Xuig" value="1"/>
<lowerValue xmi:type="uml:LiteralInteger" xmi:id="_laTmuAxhEeuXG9qw4_Xuig" value="1"/>
<ownedMember xmi:type="uml:Association" xmi:id="_laTmuQxhEeuXG9qw4_Xuig" memberEnd="_laBvzgxhEeuXG9...
<ownedMember xmi:type="uml:Association" xmi:id="_laTmugxhEeuXG9qw4_Xuig" memberEnd="_laBungxhEeuXG9...
<ownedMember xmi:type="uml:Association" xmi:id="_laTmuwxhEeuXG9qw4_Xuig" name="ACTIVITY_IDENTIFIER_...
<ownedMember xmi:type="uml:Association" xmi:id="_laTmvAxhEeuXG9qw4_Xuig" name="LABOUR_OR_DELIVERY_O...
<ownedMember xmi:type="uml:Association" xmi:id="_laTmvQxhEeuXG9qw4_Xuig" memberEnd="_laBv2AxhEeuXG9...
<ownedMember xmi:type="uml:Association" xmi:id="_laTmvgxhEeuXG9qw4_Xuig" memberEnd="_laBlBQxhEeuXG9...
<ownedMember xmi:type="uml:Association" xmi:id="_laTmvwxhEeuXG9qw4_Xuig" memberEnd="_laBlCAxhEeuXG9...
<ownedMember xmi:type="uml:Association" xmi:id="_laTmwAxhEeuXG9qw4_Xuig" memberEnd="_laBv3QxhEeuXG9...
<ownedMember xmi:type="uml:Association" xmi:id="_laTmwQxhEeuXG9qw4_Xuig" memberEnd="_laBv4AxhEeuXG9...
<ownedMember xmi:type="uml:Association" xmi:id="_laTmwgxhEeuXG9qw4_Xuig" memberEnd="_laBv5QxhEeuXG9...
<ownedMember xmi:type="uml:Association" xmi:id="_laTmwwxhEeuXG9qw4_Xuig" memberEnd="_laBv6gxhEeuXG9...
<ownedMember xmi:type="uml:Association" xmi:id="_laTmxAxhEeuXG9qw4_Xuig" memberEnd="_laBv7QxhEeuXG9...
<ownedMember xmi:type="uml:Association" xmi:id="_laTmxQxhEeuXG9qw4_Xuig" name="ACTIVITY_DATE" membe...
<ownedMember xmi:type="uml:Association" xmi:id="_laTmxgxhEeuXG9qw4_Xuig" memberEnd="_laBv8gxhEeuXG9...
<ownedMember xmi:type="uml:Association" xmi:id="_laTmxwxhEeuXG9qw4_Xuig" memberEnd="_laKqQQxhEeuXG9...
<ownedMember xmi:type="uml:Association" xmi:id="_laTmyAxhEeuXG9qw4_Xuig" memberEnd="_laKqRAxhEeuXG9...
<ownedMember xmi:type="uml:Association" xmi:id="_laTmyQxhEeuXG9qw4_Xuig" memberEnd="_laKn7gxhEeuXG9...
<ownedMember xmi:type="uml:Association" xmi:id="_laTmygxhEeuXG9qw4_Xuig" memberEnd="_laBv9wxhEeuXG9...
<ownedMember xmi:type="uml:Class" xmi:id="_laTmywxhEeuXG9qw4_Xuig" name="LABORATORY_CODE">
<upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_laTmzgxhEeuXG9qw4_Xuig" value="1"/>
<lowerValue xmi:type="uml:LiteralInteger" xmi:id="_laTmzwxhEeuXG9qw4_Xuig" value="1"/>
<ownedMember xmi:type="uml:Class" xmi:id="_laTm0AxhEeuXG9qw4_Xuig" name="LABORATORY_RESULT_AUTHORIS...
```

Figure 3.12: Key UML Model constructs

although an argument could be made for equating it with object property. The package element is used as a grouping or classification construct and could also be mapped to a Concept, we have chosen to map it to a *Concept System*, since it is being used to give the name of *Data Dictionary* to the collection of classes, and thus is closer to a whole concept system rather than a particular concept.

The item *uml:Collaberation* represents another classification method, in essence the UML construct of collaberation is a classifier, used to relate different entities within a model, although here it is only used once to reference the XMI file being used to transfer the model itself, we therefore have mapped it to the Relation construct. The two number constructs, *uml:LiteralInteger and uml:LiteralUnlimitedNatural* only occur to identify the number of relations in associations, and only the number 1 is used, so it is not obvious why the second construct is used here, however both can be mapped to the ISO Integer (data type).

We can also find some *modelling information* buried in some of the text which is present in the XMI file, for instance in the *body* XMI element we find referencing to the format used, which is in the notation *an2 or n1* where *an* stands for alpha-numeric, and *n* stands for numeric, with the digits representing how many digits are used. This is in effect a constraint on the value domain, and so arguably we can extract the format string and use it as a value domain in the ISO metamodel.

Table 3.2: NHSDD - ISO11179 Mappings

NHSDD construct	**ISO Construct (v.0.1)**	**Notes**
uml:Class	Classification	mapping
uml:Association	Relation	mapping
uml:Package	Concept System	mapping
uml:Collaboration	Relation	no mapping
uml:LiteralInteger	Integer datatype	mapping
uml:LiteralUnlimitedNatural	Integer datatype	mapping
uml:Attribute	Data Element	mapping
uml:DataType	Data Type	mapping
uml:Enumeration	(Described or Enumerated) Value Domain	mapping
-	Data Element Obligation (attribute)	no mapping
...	Object Class	no mapping
...	Property	no mapping
...	Described Conceptual Domain	no mapping
...	Enumerated Conceptual Domain	no mapping
...	Data Element Derivation	no mapping
...	Derivation Rule	no mapping

NHSDD Mapping - Summary

The exercise of mapping the NDD UML model to the ISO11179 metamodel leaves a very similar absence of elements as does the previous mapping, most items can be mapped, however the core ISO constructs are absent from our initial mapping attempt and without further expert input it is unlikely to provide an effective model.

SNOMED CT

SNOMED CTis one of the largest clinical datasets in the world comprising of over 3 million data items, as well as an expression language which can be used to combine or coordinate elements to arrive at new meanings. It is written in OWL, the web ontology language, although the NHS distribution is by way of a set of relational database tables, these can be combined using some (python) scripts to reform an OWL file.

The core elements in SNOMED CTare concepts, descriptions, relationships and stated relationships, and these are published as separate relational tables. For the initial comparison we took these tables, and tried to insert them into the ISO Metamodel, which was only possible by extending into both the Concepts region and the Designation region.

Table 3.3: SNOMED CT - ISO11179 Mappings

SNOMED Construct	**ISO Construct (v.0.1)**	**Notes**
Concept	Concept	mapping
Term	Designation	mapping
Relationship	Link (Link End and Relation Role)	mapping
StatedRelationship	Relation	mapping
Description	Definition	mapping
-	Data Element	no mapping
-	Permissible Value	no mapping
-	(Described or Enumerated) Value Domain	mapping
...	Object Class	no mapping
...	Property	no mapping
...	Described Conceptual Domain	no mapping
...	Enumerated Conceptual Domain	no mapping
...	Data Element Derivation	no mapping
...	Derivation Rule	no mapping

SNOMED CT Mapping - Summary

Again we are left with a surplus of empty constructs if we attempt to map SNOMED into the ISO11179 registry. We have found one published mapping of SNOMED CTto ISO11179 by Ngouongo et al. [62], and they claim to have achieved it with some reservations. Overall SNOMED-CT provides a clear terminology in a formally defined dataset, which can be converted to an ISO11179-3 metamodel, but it is still missing many of the constructs that determine the semantics, and as noted by Ngouongo et al. *"Some information gets lost each time items or vocabu- laries are imported from healthcare standards into the metamodel of ISO 11179 Ed. 3. A reconstruction of the source, e.g. a complete ODM-file, will not be possible."*.

There is no doubt that SNOMED CTcan be transformed into an ISO11179 metadata registry, however there are limitations in addition to the ones noted above, especially around the hierarchical nature of SNOMED and the lack of clear hierarchical representation within ISO11179. A hierarchical relationship can be built up using artefacts within the ISO metamodel, for instance the *"is-a"* relationship between concepts can be introduced using the Link, Link End and Relation Role, but there is no real way of knowing whether this is the best way of making the transformation. For instance we can attempted to also use a Relation and Assertion in this representation, however we were unable to define a set of rules which could be applied automatically, as a result the Relation and Assertion tables in the implementation database were either mostly empty or repeating metadata was stored elsewhere.

3.5 Concerns arising during Implementation

The first major problem was in specifying exactly what metadata should be input into the prototype metadata registry, in particular how to translate or transform existing models or meta-models into the set of constructs defined and discussed in the standard. To illustrate this issue, consider taking a data item from an existing medical dataset, in this case COSD, as shown in Figure 3.13.

Using ISO11179 we take the *date of clinical assessment* as a data element, however it would be specified with the patient details, since conceptually one would need to describe the context of the data item, this results in a data element which has an object property of *patient* as an integral part of the construct. This is illustrated in Table 3.4

However if one is trying to enter the metadata for the *Date of Clinical Assessment* shown in the first row of Figure 3.13, there is an immediate disparity in that no object class is specified,

Cancer Outcomes and Services Dataset - Breast

Data Item No.	Data Item Section	Data Item Name	Description	Format	National Code	National code definition	Data Dictionary Element	Current Collection	Schema Specification
	BREAST - REFERRALS To carry referral details for breast cancer Multiple occurrences of this data group are permitted								
BR4000	BREAST - REFERRALS	DATE OF CLINICAL ASSESSMENT	Date of clinical assessment of the breast for which a cancer is registered. This is based on clinical history and physical examination and will normally be the date of the first outpatient appointment at the breast clinic. If the patient attends more than one breast clinic, the date of each clinical assessment undertaken should be recorded.	an10 ccyy-mm-dd			DATE OF CLINICAL ASSESSMENT	NEW	Mandatory
BR4010	BREAST - REFERRALS	ORGANISATION SITE CODE (OF CLINICAL ASSESSMENT)	Provider code where clinical assessment of the breast for which a cancer is registered was carried out. This is based on clinical history and physical examination and will normally be the site code of the first outpatient appointment at the breast clinic. If the patient attends more than one breast clinic, the site code of each breast clinic where a clinical assessment was undertaken should be recorded.	Minimum length an5, maximum length an9		see ORGANISATION SITE CODE	SITE CODE (OF CLINICAL ASSESSMENT)	NEW	Mandatory
BR4020	BREAST - REFERRALS	CLINICAL ASSESSMENT RESULT (BREAST)	Result of the clinical assessment of the breast for which a cancer is registered. This will normally be the result of an assessment of a patient's clinical history and physical examination undertaken at the first outpatient appointment at the breast clinic. If the patient attends more than one breast clinic, the result of each clinical assessment should be recorded.	an2	P1	Normal	CLINICAL ASSESSMENT RESULT CODE (BREAST CANCER)	NEW	Mandatory
					P2	Benign			
					P3	Uncertain			
					P4	Suspicious			
					P5	Malignant			

Figure 3.13: COSD Dataset Excerpt

Table 3.4: ISO11179 Cancer Referral Representation

ISO Artefact	**Description**
Data Element	Patient and Date of clinical Assessment in form ccyy-mm-dd
Data Element Concept	Patient and Date of clinical Assessment
Value Domain	Date in form ccyy-mm-dd
Object Class	Patient
Property	Diagnosis Date

available or immediately obvious from the spreadsheet. There is mention of the outpatient or patient in the description, however it is not in the view of the analyst preparing the dataset of any significance, and therefore was not included in a separate column. Therefore there is immediately a dilemma, do we enter *outpatient, patient* or simply leave the object property blank? The next issue is the idea of having the data element, if we drop the *Patient* from the name and then call the data element *Date of Clinical Assessment*, what then is the *Data Element Concept*? and how is it different from the description of the *Data Element*? Should the Data Element Concept include the patient? There is no obvious answer, and from a user perspective it appears that a simple set of dataset metadata, i.e. column headings on a spreadsheet, are being transformed into something more complex in order to manage them, but that management can only be carried out by experts versed in ISO/IEC11179.

3.6 Discussion

The discussions so far have established that the ISO11179 metadata registry, whilst having a structure to enable semantic interoperability in theory, is difficult to use without a lot of expert human intervention and guidance. For 3 datasets it may be worthwhile setting up an ISO11179 metadata registry using the ISO11179-3 metamodel, however the differences between the representations and mappings do not achieve the goal of having a single registry which is able to easily map between data elements from different dataset standards. The work carried out with clinicians did not allow new datasets to emerge based on different parts of the existing datasets, and therefore we looked to develop the metamodel to see if it could be enhanced to overcome these drawbacks.

During the course of implementing the ISO11179 metadata registry the problems with loading existing dataset standards have been documented here. One of the outcomes of the kind of ambiguity indicated was that the same text was entered in many different fields of the database, so that a data element, data element concept, data type and value domain would end up with the same name - resulting in searches which pull back 4 items which was confusing for clinicians and data curators using the system.

To illustrate this, suppose we have a database table called *visit summary* which is recording the visits a diabetes patient has to a hospital in the course of a year. The table will look something like Figure 3.14

Visit Summary	
event_id	Integer
patient_id	Integer
date_of_visit	Date
professional_seen	Integer

Figure 3.14: Diabetes - Summary of Visits Table

We see a field recording the patient identity, the date of the visit and the type of health professional seen. In practise the table will probably include many more fields, but for now just consider we have a table like this, and we need to generate an entry in metadata registry to capture the metadata surrounding this event. It maybe we need to map this dataset or database to a particular dataset standard, or simply that we need to record the metadata surrounding the dataset. This process will generally involve importing the information that is available

through the database, most databases such as Oracle 9i, MySQL 5/8 and Postgres will have jdbc connectors allowing one to obtained the structure and linkage of tables programmatically. A mapping exercise will normally involve importing a table such as this one into a metadata registry by extracting details such as the names of fields, the datatypes of field and the foreign key linkage of tables to obtain enumerations from simple data entries.

Let us assume such a table is being imported into an ISO11179 compliant metadata registry, the task will be to infer from the two tables concerned the contents of 6 related tables in the metadata registry, as shown in Figure 3.15.

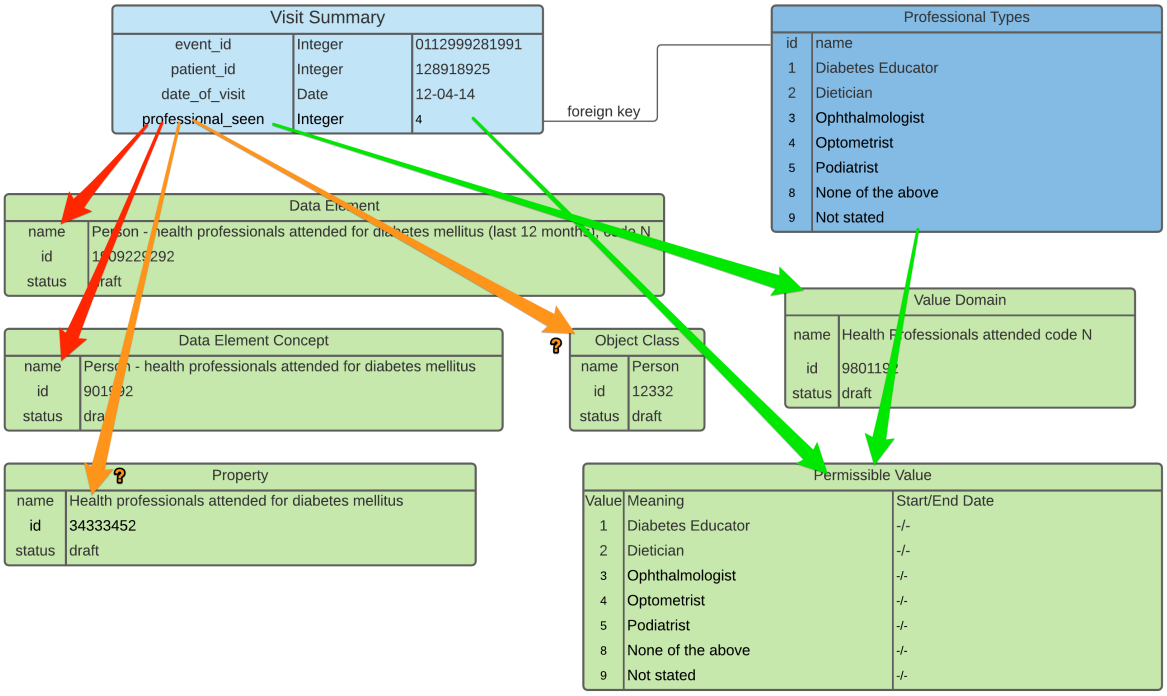

Figure 3.15: Diabetes - Ideal Mapping to ISO11179

From the ISO11179-3 registry an expert will be able to generate a set of table entries consisting of metadata matching the diagram shown in Figure 3.15, however consider what is possible without an expert present. Firstly we only have the descriptions of field that are present in the database, we can identify the name of the field *professional_seen* as the name of a data element, but all we have are the terms in the name of the field. If we compare this with data elements in a publically available medical metadata registry (in this case we have taken the example data from the online Aristotle ISO11179 metadata instance [81]), we see that a typical entry would be called *"Person - health professional attended for diabetes mellitus*

*last*12*months code N"*, and likewise this would be linked to a data element concept, a value domain, an object class and property and a set of permissible values as illustrated.

The problem a clinician or data manager has when importing such a table from a working database is getting any more information than is available from the headers, this results in a set of metadata registry entries like those in 3.16. For this example we are able to populate the Data Element and Data Element concept with the name from the header, *professional_seen*, which can be changed using simple text string manipulation to *Professional seen*.

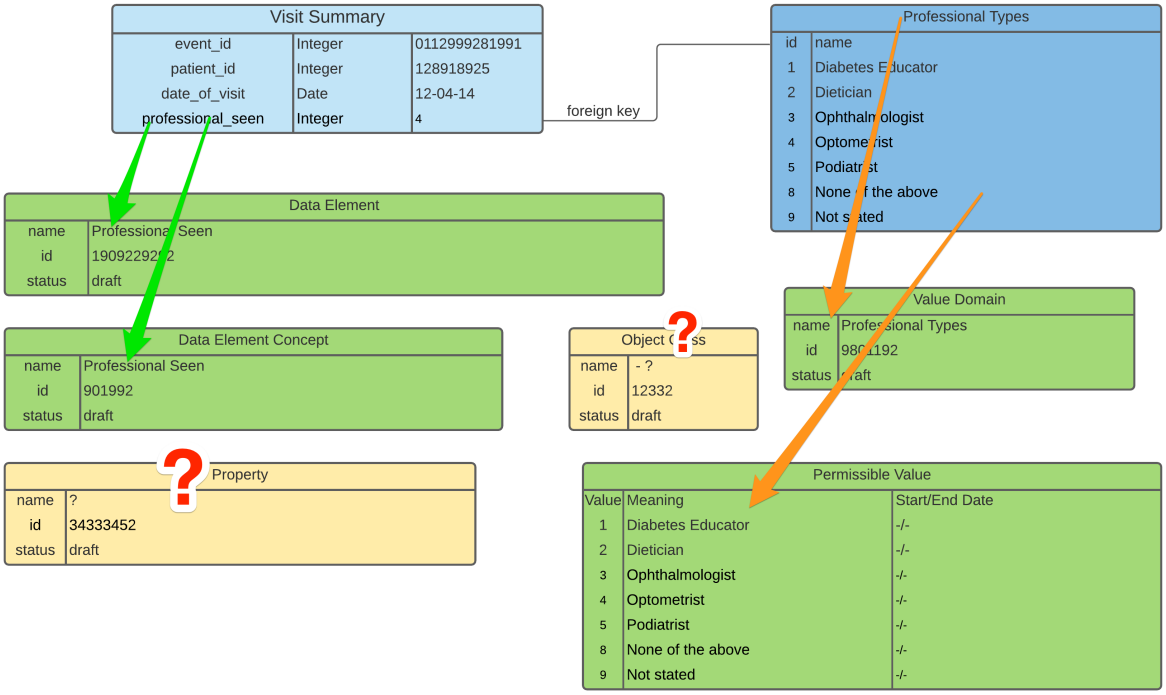

Figure 3.16: Diabetes - Practical Mapping to ISO11179

The Value Domain, or Enumerated Value domain class name can be obtained by using the table name of the foreign key of this field, although this is not very reliable approach since the database in question could put all the lookups in one big table called *lookups*. More common would be simply adopting the field name *Professional Seen* for the Value Domain as well, resulting in 3 key artefacts having identical names.

The next problem is with the Object Class and Property, here there is no obvious rule that allows an importing process to identify the subject as a person, as perhaps a human expert would be able to do, and these two tables will go empty in most cases. Hopefully this example is sufficient to illustrate why the full ISO11179 metamodel can in some circumstances, in particular those involving automatic ingestion of metadata, result in a bloated registry, full

of identical terms or names for data element concepts, data elements and value domains, and lacking in entries for object class and property. Navigating such a registry then becomes difficult and confusing for those without a good understanding of the standard.

In reviewing the experience of building an ISO conformant registry we made the following observations:

Observation 1 *The UML models provide a great deal of detail for each sub-section of a metadata registry, however no system diagram or clear description is provided, it is therefore difficult to build a working system based on the UML diagrams alone, considerable interpretation is required, which detracts from the specification provided.*

Observation 2 *The examples provided are very concept specific, for instance, the example of country codes works for concepts which are used as a list, but many concepts used in healthcare are not used in such a straightforward fashion. Furthermore, the standard supports only a single ontology for the organisation of 'data element concepts', meaning that all users of a particular registry must agree upon the same set of relationships between these concepts, storing 2 separate* OWL *ontologies, such as SNOMED and HPO is not possible.*

Observation 3 *The relation and linking rules are adaquate, but do not easily allow for clinicians to curate local datasets based on dataset standard. A better way of linking and modifying data elements is required, currently data elements (attributes in a object oriented program or fields in a database) cannot be directly linked to other data elements.*

Observation 4 *Data Elements can only be grouped or placed in hierarchical form with the help of concepts, this means that capturing the metadata around a class hierarchy. In short there is no notion of composition relating to data elements, to capture composition one needs to overload the relationships in the concept metamodel. It also means that inferring context from structural relationships is not possible, thus limiting the ability to capture structural metadata.*

Observation 5 *Basic types used in the* metamodel *include types which in most computer science contexts would be viewed as derived types (e.g. Address - see A), this makes implementation needlessly complex.*

Observation 6 *The publishing, administration and versioning rules which are applied to Data Element in* ISO*11179 are very useful, however the way in which they are applied to individual data elements does not reflect the way in which medical authorities manage datasets in our experience.*

Observation 7 ISO*11179 addresses technical metadata, however there is no obvious way to manage descriptive metadata, issues such as ownership and sensitivity.*

The key problem in this approach is that the gathering all the artefacts that form the metadata around a data element is a manual process that can only be undertaken by domain experts who also understand the ISO Standard. It is an expensive and time-consuming process, with un-proven results. Later on in this thesis we look at the core problems which are faced by researchers trying achieve some degree of semantic interoperability.

The standard is declared as being a standard for metadata registries, and although it contains a lot of disparate ideas on the subject of interoperability which can be applied to a metadata registry, no core set of definitions, core language or metamodel was found that could be used as a measure for conformance. Standards by definition should be conformed to, and whilst conformance is mentioned, it is lacking a clear set of definitions which can be used as a measurement for conformity.

Therefore the goal of building a fully ISO11179 *conformant* metadata registry was abandoned early on, when it became apparent that clear objective conformance criteria were not present in the standard, despite the subject of conformance being discussed. There is much in the standard document which can be usefully incorporated into the design of a metadata registry, and development continued with a view to improve those aspects of the standards which could be shown to be beneficial to the construction of a metadata registry.

The work carried out by Hannes Ulrich and colleagues [30] with the North German Tumour Centre, mentioned earlier, and although their solution is very different, it is clear that they encountered similar problems, which they resolved by pairing the ISO conformant metadata registry with a FHIR based repository and a separate database. Our solution takes this further in that we proposed an abstract model, built around the core ISO metamodel.

3.7 Summary

We have shown in this chapter the core workings of the ISO11179 standard for metadata registries, and why it can contribute to the development of interoperable data and information systems.

We have also highlighted some flaws and difficulties which become apparent when the standard is used to manage large datasets, and where expert human curation is simply not available to the degree needed to ensure the successful operation of the standard.

The key problems identified here are summarised as being:

- an inability to classify and group data elements
- lack of consistency and thus repeatable results)
- an ambiguous set of artefacts in the metamodel systems

After attempting to re-interprete the ISO11179 to meet these requirement we then attempted to revise the specification using the Z specification language with the intention of resolving the ambiguity present in the specification. We managed to re-work the metamodel described in ISO11179-3 parts 9, 10 and 11, and this is described in the next chapter.

In the next chapter we take these problems and try to resolve them by using them to specify key requirements for a domain specific language to handle data models, a language which will then form the basis of our implementation of a metadata registry.

Chapter 4

Specification

This chapter provides a formal reference for a standard which is enhanced to include many of the relationships between data structures. The work here builds on the lessons learnt in the last chapter, which show weaknesses in the standard when it is used for medical dataset curation and data quality use cases. There were 7 observations made, and these are addressed as follows:

1. This is addressed by building a formal specification for the new registry metamodel.
2. The design of the registry must consider not just data elements but also groups and collections of data which we are calling models.
3. The different data structures need to have the ability to link and relate to one another.
4. The key aspects of structure relating to data items need to be recorded.
5. We need to include a simple type system that relates to commonly used objected oriented languages.
6. We need to retain the administrative and publishing aspects of ISO11179
7. We need to have a mechanism to describe descriptive metadata, which will vary between different users and organizations.

In approaching this revision of the core metamodel the first step is to consider a metadata registry populated by *models* rather than by individual data elements. We will adopt a notion of model consistent with UML, even if that language does not make such a notion explicit.

For our purposes, a model will be a collection of classes, typically linked by associations. We will represent associations, inheritance, and composition as reference-valued attributes of

classes. As our principal concern is data description, we will remain agnostic with regard to behavioural aspects of inheritance.

Our language of models has two purposes. The first is to group and structure definitions so that when we wish to say something about all of the attributes in a class, or in a hierarchy of classes (under inheritance or composition), we need say it only once, at the level of the enclosing class, superclass, or model.

The second is to make a connection between a structured collection of data definitions—such as those pertaining to a form for data entry, or a database schema—and the software artefact in question. If the artefact exists, then much of the metadata for registration can be created automatically; if not, then part or all of the artefact may be generated automatically from the model.

Our first step in achieving a scalable approach to metadata registration and semantic interoperability is to consider a metadata registry populated by *models* rather than by individual data elements. We will adopt a notion of model consistent with UML, even if that language does not make such a notion explicit.

Data Elements, Classes, and Models

We will use the term attribute interchangably with Data Element. The ISO standard is focussed around data elements, and so in the resultant language we will still use that term, however in this analysis we use the term attribute as a synonym of data element. Each individual data element as an *dataElement*, corresponding to an individual field, variable, column, or property in a software artefact, or to an individual item in a data standard. Each dataElement definition will include a name, a textual explanation, and a reference to an enumeration or data type, representing the set of values that the dataElement may take.

To facilitate the capture of contextual information from, and the generation of, software artefacts we will include also multiplicity and logical constraint information. The multiplicity of a dataElement is described in the usual way, as a pair of numbers or symbols, representing the lower and upper bounds upon the number of values associated with a single instance.

A class definition will include the definitions of its attributes, which we register as dataElements, together with a logical constraint and a textual explanation. A fully-qualified class name may be used also as the type of a dataElement. In this case, the dataElement represents a navigable association between two classes. As in object-oriented design, the constraint and the textual explanation of one class may make reference to the attributes of another, associated

class. For this reason, it is advisable to manage the definitions of associated classes, and the constraints they impose, together.

We will use the notion of a *model* to describe a collection of class definitions. We will use models as our 'units of context and versioning': attribute and class definitions will be developed, released, and updated only as components of models.

Types, Units, and Enumerations

The type of an attribute may be a primitive type, corresponding to a string or numeric type in an implementation. The description of a numeric type may include also an indication of the units employed, such as kilograms or metres per second. Alternatively, the type may be an enumeration, a terminology, or a reference to a class.

An *enumeration* contains a list of enumerated items or terms that may appear as values for attributes. It may contain also explanatory text, providing additional information regarding the interpretation of one or more of the enumerated items it contains. Each individual item may contain further textual explanation, together with a set of links to other items. We will manage enumerations as components of models, just as we manage classes and attributes.

A *terminology* is a set of linked items or terms. The difference is that we expect a terminology to be used across many models, and perhaps to contain other items that cannot be used as values. Furthermore, a terminology will typically be maintained as an artefact in its own right, and not as a part of the design of a software artefact or data standard.

Example

The class diagram shown in Figure 4.1 presented using the class diagram notation of the Unified Modeling Language (UML) [68]. The model contains five classes: **Person**, **NHS Organisation**, **Hospital Trust**, **Laboratory** and **Department**. The class **Person** is shown with two attributes: 'age' and 'gender'. The type of 'gender' is given by an enumeration included in the same model; there are four possible values for 'gender', each of which comes with a textual explanation.

There is an association between **Person** and **NHS Organisation**: this may be seen as a reference-valued attribute of **Person**, called 'employedBy'. This association is many-to-many, as indicated by the asterisks at either end. There is a constraint, written in the Object Constraint Language (OCL) designed for use with UML. This constraint is defined in the context

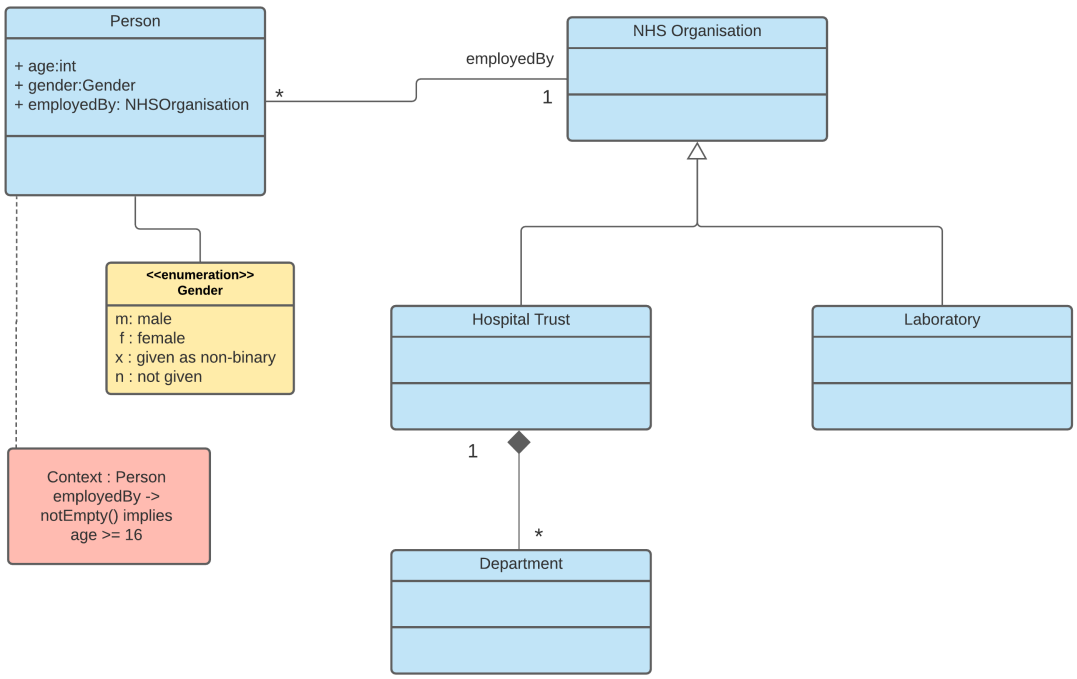

Figure 4.1: A class diagram for the model 'Employment'

of the class **Person**, and insists that whenever the reference-valued attribute 'employedBy' is non-empty, the value of the attribute 'age' must be greater than or equal to 16.

There are two subclasses of **NHS Organisation**, as indicated by the lines leading to the hollow arrowhead: this is the notation for inheritance in the UML. An instance of one of the subclasses, **Hospital Trust**, can 'contain', or take sole responsibility for, a number of instances of **Department**. This is indicated by a line with a solid diamond: this is the notation for composition in the UML. If this association were annotated with 'role names' at either end, then these would be treated as reference-valued attributes within the classes, in the same way as 'employedBy' within **Person**.

4.1 Extending Definitions

Our second step in achieving scalability is to introduce a more flexible, distributed approach to the assertion of semantic interoperability within a registry. Rather than relying on a single object-class-property hierarchy of data element concepts, or assertions that two data elements share the same data element concept, for semantic interoperability, we will introduce an explicit notion of semantic interoperability, or substitutability, between data elements.

We will introduce a link between definitions to indicate that one definition **extends** or **refines** another: that is, it says everything that the other definition says, and possibly more. This is a notion which appears as the **Open-Closed** principle [46, 7, 56] in Software Engineering, but which has a poor semantics in UML as pointed out by Laguna et al. [46]. Here we are applying the same principle to Data and giving it a formal semantics. This notion of semantic relationships reflects our intention to use linked collections of data models to record or determine that data collected in one context can be safely re-used in another context. For this to be the case, the defining properties of data in the first context must be enough to guarantee the properties of data in the second context. For the rest of this chapter we will use the term **refines** in order to avoid confusion with the term **extends**.

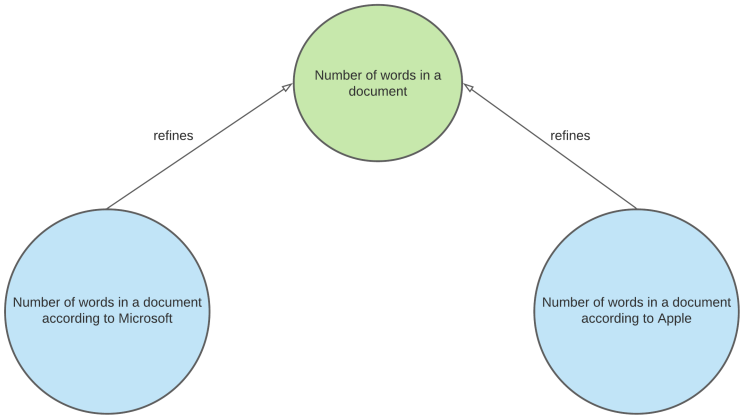

Figure 4.2: Two refinements of a definition

As an example, consider the three definitions given in Figure 4.2. Here, we are assuming that the name of the attribute—e.g. 'word count'—and the datatype—e.g. positive integer—is the same in each case, and we are focussing simply upon the explanatory text. The two refinement arrows represent assertions that the definitions to the left and right are both refinements of the definition at the centre.

Such assertions cannot be derived automatically from the explanatory text. The explanations may be subjective, and will certainly give only a partial account of the context. However, the application of natural language processing techniques to the explanatory text, together with the structural, contextual information provided by the enclosing models, will enable a degree of automation in the curation and linking of definitions.

The *distributed* nature of our approach comes from the fact that we will require that each of these assertions is made locally, in the context of a particular model: that is, they are not global statements or facts for a registry, in the way that the object-class-property hierarchy is

for the ISO/IEC 11179 standard. We will allow global statements, as classification schemes, but for now that is an orthogonal concern.

We will require that the model in which the assertion is made contains at least one of the two definitions in question. So that we may make an assertion of refinement from the perspective of either definition, we will introduce also the opposite notion, saying that one definition *abstracts* another if and only if that other definition refines it.

As our models may represent incomplete information about the artefacts they represent, and the semantic relationships between them, we cannot infer that no refinement exists simply because there is no *refines* or *abstracts* link present. To record the assertion or conclusion that no refinement exists, we have two other forms of link: *does not refine* links and *does not abstract* links.

As a further example, consider the three attribute definitions given in Figure 4.3. Here, we have a more specific definition at the centre of diagram: 'number of words in a document ignoring hyphens'. We have also an assertion that this refines the definition with the explanatory text 'number of words in a document'. If we accept this assertion, then any data collected against the new definition can be used in any situation in which the original definition was accepted.

We have also an assertion that the definition 'number of words in a document according to Microsoft' is not a refinement of this new, more specific definition. The 'word count' feature in Microsoft's Word application treats hyphenated phrases as single words: it does not ignore hyphens. In contrast, the same feature in Apple's Pages application does ignore hyphens, treating them as if they were spaces. For example, Word will count the phrase 'strongly-connected' as one word, whereas Pages will count it as two words.

Attribute definitions may be linked to indicate that one refines, abstracts, does not refine, or does not abstract the other. A link indicating refinement or abstraction may make reference to a data transformation, indicating how the names and values of attributes in one context may be related to those of corresponding attributes in the other, linked context.

4.2 Consistency

Allowing refinement to be asserted separately and independently in each model eliminates the dependence upon a single hierarchy. However, it raises the question of consistency: it is perfectly possible to make inconsistent assertions about the same data elements. How are we to ensure

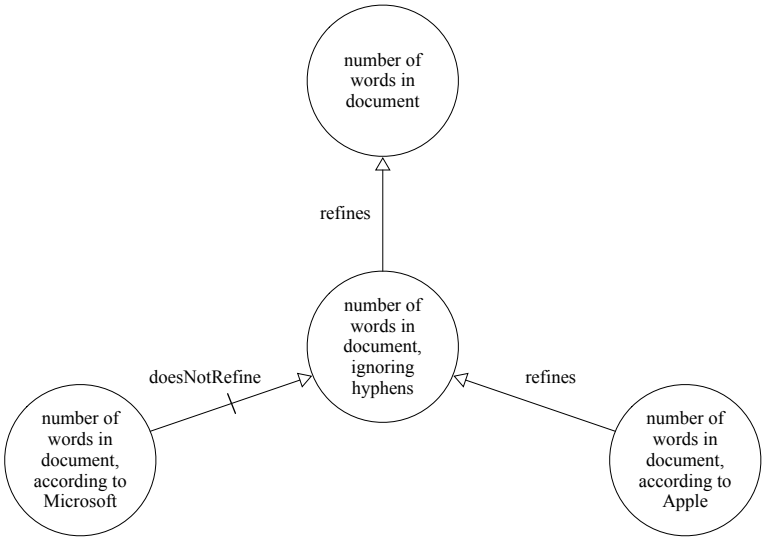

Figure 4.3: A definition that is not a refinement

that a metadata registry consisting of independent models, rather than a coordinated collection of data elements, will provide consistent answers to questions of semantic interoperability?

We now propose a simple protocol for model registration, and prove that it is sufficient to guarantee consistency. We wish to allow differences of opinion: for example, the author of one model might feel that one existing data definition refines another *for the particular purpose that they have in mind* whereas the author of another model, with a different purpose, might feel that this is not the case.

Here, the notion of 'purpose' equates to 'the application context', and it may be described by a particular model, together with all of the other models that this model refers to or depends upon. It should be perfectly acceptable to have a pair of assertions that would contradict each other, provided that they are made in different application contexts.

What should not be acceptable is the registration—or, at least, publication—of a model that combines these two contexts. If a user of the metadata registry were to refer to this model in deciding whether the two data elements were semantically compatible, they would obtain an inconsistent response. We might view the registration of such a model as a loss of integrity for the registry.

An additional concern, given the role of a metadata registry as a lasting source of reference information regarding data, is that the answers provided by the registry should be consistent over time. For a given purpose, described by a particular model, any assertion made or derived regarding semantic interoperability should remain true from that point onwards. Whilst models will change during their initial creation phase, once a model is complete it is finalised or put into

an immutable state, and thus consistency is established. The full publication cycle is describe later on, but it is worth mentioning that in this formal treatment the models are finalised or superseded, both states being considered the same for this treatment.

A Formal Semantics

To derive an appropriate protocol, we will first construct a formal, abstract account of a metadata registry containing models written in our data modelling language. We will use the formal notation Z introduced in chapter 2.2.

We will write *Name* to denote the set of all names for data elements, classes, types, and models, *Text* to denote the set of all possible natural language descriptions, and *Constraint* to denote the set of all possible logical constraints. To support our description of numeric data types, we will introduce also the concepts of units and sign:

$$[Name, Text, Constraint, Units, Sign]$$

A path, or fully-qualified name, is a non-empty sequence of names:

$$Path == \mathrm{seq}_1\, Name$$

Each semantic link—each assertion of refinement or abstraction, or their negation—is associated with a path, leading to the classifier or term that is the subject of the assertion.

$$\begin{aligned} Link ::= {} & refines\langle\langle Path\rangle\rangle \mid doesNotRefine\langle\langle Path\rangle\rangle \\ \mid {} & abstracts\langle\langle Path\rangle\rangle \mid doesNotAbstract\langle\langle Path\rangle\rangle \end{aligned}$$

Each item in a registry will have a name, a path, and a textual explanation; the name of the item will be the last name appearing in the path.

Item

$$name : Name;\ path : Path;\ text : Text$$

$$name = last\ path$$

For example, we could define a data element called *bloodType* within our registry, and it would have a unique path such as *https://theDevonRegistry/1234/bloodType* the first component being a unique path, defined by a dns name (theDevonRegistry) followed by a unique identifier (1234) followed by a name.

If the item in the registry is a classifier, it will have two further properties: a logical constraint and a set of links.

CItem

$$Item;\ constraint : Constraint;\ links : \mathbb{P}\ Link$$

If the classifier is a data attribute, representing a data element, it will have another property: a path leading to a type definition.

Attribute

$$CItem;\ type : Path$$

If the classifier is a data class, it will have three other properties: a set of paths leading to classes that the current class extends or inherits from; a set of paths leading to classes that the current class notionally contains; and a set of attributes. We are able to reason on the *name* which is available from including the *CItem* definition, which in turn includes the *Item* definition.

Class

$$CItem;\ extends, contains : \mathbb{P}\ Path;\ attributes : \mathbb{P}\ Attribute$$

$$\forall\ a : attributes \bullet a.path = path \frown \langle a.name \rangle$$

An attribute can belong to only one class: the attribute definitions are contained within the class definition. For each attribute, the attribute path is obtained by extending the class path with the name of the attribute.

An enumerated item has a set of links,

Enum

$$Item;\ links : \mathbb{P}\ Link$$

and an enumeration contains a set of enumerated items:

Enumeration

$$Item;\ enums : \mathbb{P}\ Enum$$

$$\forall\ e : enums \bullet e.path = path \frown \langle e.name \rangle$$

A model is itself a classifier, bringing additional constraint information and its own set of links. Models are 'top level' items within a registry: a model path contains only the name of the model in question. A model contains classes and enumerations: in each case, the path to the contained item consists of the name of the model followed by the name of the item.

Model

$$CItem;\ classes : \mathbb{P}\ Class;\ enumerations : \mathbb{P}\ Enumeration$$

$$path = \langle name \rangle$$
$$\forall\, c : classes \bullet c.path = \langle name, c.name \rangle$$
$$\forall\, e : enumerations \bullet e.path = \langle name, e.name \rangle$$

Note that it is not necessary, for the purposes of consistency, to insist that inheritance relationships can exist only between classes in the same model.

To allow us to cover the full range of information considered in the ISO/IEC 11179 standard, we need to allow for different representations of the same data element concept. It will be sufficient to introduce a notion of transformations upon data elements, as functions upon sets of attribute-value pairs:

Transformation

$$Item;\ transform : (Path \nrightarrow Path) \nrightarrow (Path \nrightarrow Path)$$

$$path = \langle name \rangle$$

Each attribute or value is identified by a path. Each set of attribute-value pairs will be functional, in the sense that each attribute will have at most one value. We will regard transformations as another kind of 'top level' item, alongside models.

We will extend our notion of semantic links to allow positive assertions of refinement, in either direction, under a particular transformation:

$$Link ::= \ldots \mid refinesUsing\langle\langle Path \times Path \rangle\rangle \mid abstractsUsing\langle\langle Path \times Path \rangle\rangle$$

A mapping is a top-level item relating values to values,

Mapping

$$Item;\ relation : Path \leftrightarrow Path$$

Note that such a relationship need not be functional.

A type is another top-level item with a range of values from some primitive or native datatype and—for numeric types—a description of the units employed, and a type conversion is a top-level item with a function acting on primitive datatypes and—for numeric types—source and target units. For the purposes of this thesis, it is enough to know that these items are maintained within the registry, alongside the data models, and that they can be referenced using fully-qualified names or paths.

Type

$$Item;\ values : \mathbb{P}\ Primitive;\ units : Units$$

Conversion

$$Item;\ function : Primitive \nrightarrow Primitive;\ source, target : Units$$

In the same way that transformations allow us to record that two data elements correspond to the same data element concept, we can use type conversions to record that two value domains correspond to the conceptual domain.

A registry is simply a collection of these top-level items:

Registry

$$models : \mathbb{P}\ Model$$
$$types : \mathbb{P}\ Type$$
$$mappings : \mathbb{P}\ Mapping$$
$$transformations : \mathbb{P}\ Transformations$$
$$conversions : \mathbb{P}\ Conversion$$

For the purposes of deriving a protocol for consistency, it will be enough focus upon the registration of models. The other components of the registry support the definitions of data elements contained within the models, and the assertions regarding relationships between them.

As an example of the 'abstract syntax' for models, we will consider how the 'Employment' model of Figure 4.1 might be represented.

Before doing so, we will introduce a second model, 'HMRC', in order that we may demonstrate also semantic linking. This second model, shown in Figure 4.4, has two classes, **Person** and **Organisation**, and an association between them that corresponds to a reference-valued attribute of **Person** named 'employeeOf'. The model might contain also constraints, setting

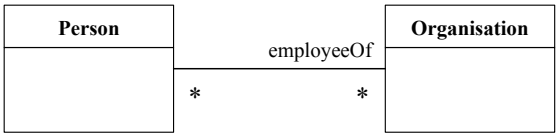

Figure 4.4: The model ‘HMRC’

out the conditions under which Her Majesty’s Revenue and Customs (HMRC) would consider a person to be an employee of an organisation.

The representation of the ‘Employment’ model is then shown in Figure 4.5. In this example, only the values of ‘Gender’ have any explanatory text.

$\langle\!|$ *name* $\rightsquigarrow$ Employment, *path* $\rightsquigarrow$ ⟨Employment⟩, *text* $\rightsquigarrow$ `""`, *links* $\rightsquigarrow$ {},

classes $\rightsquigarrow$ {$\langle\!|$*name* $\rightsquigarrow$ **Person**, *path* $\rightsquigarrow$ ⟨Employment, **Person**⟩, *text* $\rightsquigarrow$ `""`,
links $\rightsquigarrow$ {*refines* ⟨HMRC, *Person*⟩}, *extends* $\rightsquigarrow$ {}, *contains* $\rightsquigarrow$ {},
constraint $\rightsquigarrow$ `employedBy->notEmpty() implies age >= 16`
attributes $\rightsquigarrow$
{$\langle\!|$*name* $\rightsquigarrow$ age, *text* $\rightsquigarrow$ `""`, *links* $\rightsquigarrow$ {},
path $\rightsquigarrow$ ⟨Employment, **Person**, age⟩,
multiplicity $\rightsquigarrow$ {1}, *type* $\rightsquigarrow$ Age$|\!\rangle$,
$\langle\!|$*name* $\rightsquigarrow$ gender, *text* $\rightsquigarrow$ `""`, *links* $\rightsquigarrow$ {},
path $\rightsquigarrow$ ⟨Employment, **Person**, gender⟩,
multiplicity $\rightsquigarrow$ {1}, *type* $\rightsquigarrow$ ⟨Employment, Gender⟩$|\!\rangle$,
$\langle\!|$*name* $\rightsquigarrow$ employedBy, *text* $\rightsquigarrow$ `""`,
links $\rightsquigarrow$ {*refines* ⟨HMRC, **Person**, employeeOf⟩},
path $\rightsquigarrow$ ⟨Employment, **Person**, employedBy⟩,
multiplicity $\rightsquigarrow$ ℕ, *type* $\rightsquigarrow$ ⟨Employment, **Organisation**⟩$|\!\rangle$}$|\!\rangle$,

$\langle\!|$*name* $\rightsquigarrow$ **Organisation**, *path* $\rightsquigarrow$ ⟨Employment, **Organisation**⟩,
text $\rightsquigarrow$ `""`, *links* $\rightsquigarrow$ {*refines* ⟨HMRC, **Organisation**⟩},
extends $\rightsquigarrow$ {}, *contains* $\rightsquigarrow$ {}, *attributes* $\rightsquigarrow$ {}$|\!\rangle$,

$\langle\!|$*name* $\rightsquigarrow$ **GovDept**, *path* $\rightsquigarrow$ ⟨Employment, **GovDept**⟩,
links $\rightsquigarrow$ {}, *extends* $\rightsquigarrow$ {⟨Employment, **Organisation**⟩},
contains $\rightsquigarrow$ {}, *attributes* $\rightsquigarrow$ {}$|\!\rangle$,

$\langle\!|$*name* $\rightsquigarrow$ **Company**, *path* $\rightsquigarrow$ ⟨Employment, **Company**⟩,
links $\rightsquigarrow$ {}, *extends* $\rightsquigarrow$ {⟨Employment, **Organisation**⟩},
contains $\rightsquigarrow$ {⟨Employment, **Subsidiary**⟩}, *attributes* $\rightsquigarrow$ {}$|\!\rangle$,

$\langle\!|$*name* $\rightsquigarrow$ **Subsidiary**, *path* $\rightsquigarrow$ ⟨Employment, **Subsidiary**⟩,
links $\rightsquigarrow$ {}, *extends* $\rightsquigarrow$ {}, *contains* $\rightsquigarrow$ {}, *attributes* $\rightsquigarrow$ {}$|\!\rangle$

enumerations $\rightsquigarrow$ {$\langle\!|$*name* $\rightsquigarrow$ Gender, *path* $\rightsquigarrow$ ⟨Employment, Gender⟩,
enums $\rightsquigarrow$ {$\langle\!|$*name* $\rightsquigarrow$ m, *path* $\rightsquigarrow$ ⟨Employment, Gender, m⟩,
text $\rightsquigarrow$ "male", *links* $\rightsquigarrow$ {}$|\!\rangle$,
$\langle\!|$*name* $\rightsquigarrow$ f, *path* $\rightsquigarrow$ ⟨Employment, Gender, f⟩,
text $\rightsquigarrow$ "female", *links* $\rightsquigarrow$ {}$|\!\rangle$,
$\langle\!|$*name* $\rightsquigarrow$ x, *path* $\rightsquigarrow$ ⟨Employment, Gender, x⟩,
text $\rightsquigarrow$ "given as non binary", *links* $\rightsquigarrow$ {}$|\!\rangle$,
$\langle\!|$*name* $\rightsquigarrow$ n, *path* $\rightsquigarrow$ ⟨Employment, Gender, n⟩,
text $\rightsquigarrow$ "not given", *links* $\rightsquigarrow$ {}$|\!\rangle$ } $|\!\rangle$

Figure 4.5: Formal, abstract representation of the ‘Employment’ model

Registry Properties

To simplify the presentation of our analysis, without compromising our ability to type-check or reason about the specification, we will introduce the notion of a generic registry item:

$$
\begin{aligned}
&\mathit{RegistryItem} \mathrel{\widehat=} \\
&\quad \mathit{Model} \lor \mathit{Class} \lor \mathit{Attribute} \lor \mathit{Enumeration} \lor \mathit{Enum} \lor \mathit{Type} \lor \\
&\quad \mathit{Transformation} \lor \mathit{Term} \lor \mathit{Terminology} \lor \mathit{Mapping} \lor \mathit{Conversion}
\end{aligned}
$$

together with a lookup function *item* mapping paths to items, and use these to shadow the contents of the registry:

RegistryItems

$$
\begin{aligned}
&\mathit{Registry} \\
&\mathit{item} : \mathit{Path} \pfun \mathit{RegistryItem}
\end{aligned}
$$

We will introduce a specialisation of this lookup function for each different kind of item, and insist that the properties of a shadow, generic item have the same values as those of the corresponding, actual item:

RegistryObjects

$$
\begin{aligned}
&\mathit{RegistryItems} \\
&\mathit{model} : \mathit{Path} \pfun \mathit{Model} \\
&\mathit{class} : \mathit{Path} \pfun \mathit{Class} \\
&\mathit{attribute} : \mathit{Path} \pfun \mathit{Attribute} \\
&\ldots \\
\hline
&\mathit{model} = \{m : \mathit{models} \bullet m.\mathit{path} \mapsto m\} \\
&\ldots \\
&\forall\, p : \operatorname{dom} \mathit{model} \bullet (\mathit{item}\, p).\mathit{constraint} = (\mathit{model}\, p).\mathit{constraint} \\
&\ldots
\end{aligned}
$$

We will introduce also the set of all currently-valid paths for a registry, as the union of the domains of these lookup functions.

RegistryPaths

$$
\begin{aligned}
&RegistryObjects \\
&paths : \mathbb{P}\, Path \\
\hline
&paths = \bigcup \{ \mathrm{dom}\, model, \mathrm{dom}\, class, \mathrm{dom}\, attribute, \mathrm{dom}\, enumeration, \\
&\quad \mathrm{dom}\, enum, \mathrm{dom}\, transformation, \mathrm{dom}\, term, \mathrm{dom}\, terminology, \\
&\quad \mathrm{dom}\, mapping, \mathrm{dom}\, type, \mathrm{dom}\, conversion \}
\end{aligned}
$$

The definition of one item may re-use and include the definition of another by way of a *refines* link or—in the case of a class—in terms of extension.

RegistryUses

$$
\begin{aligned}
&RegistryPaths \\
&uses : Path \leftrightarrow Path \\
\hline
&uses = \{\, p1, p2 : paths \mid \exists\, t : paths \bullet \\
&\quad refines\, p2 \in (item\, p1).links \lor \\
&\quad refinesUsing\, (t, p2) \in (item\, p1).links \lor \\
&\quad p2 \in (class\, p1).extends \}^*
\end{aligned}
$$

The relation *uses* is defined as the reflexive transitive closure of the relation corresponding to refinement or extension links.

One registry item may contain another in terms of its path or—in the case of a class—through a class composition association:

RegistryContains

$$
\begin{aligned}
&RegistryUses \\
&contains : Path \nrightarrow Path \\
\hline
&contains = \\
&\quad (\_ \preceq \_) \rhd paths \cup \{ p, q : \mathrm{dom}\, class \mid q \in (class\, p).contains \}
\end{aligned}
$$

where $\preceq$ denotes sequence prefix and $\rhd$ denotes relational range restriction.

We can use the reflexive transitive closure of the union of *uses* and $contains^{\sim}$, the inverse of this relation, to identify the context in which any item in the registry is defined:

RegistryContext

$$RegistryContains$$
$$context : Path \pfun \mathbb{P}\, Path$$
$$\forall\, p : paths \bullet context\, p = (uses \cup contains^{\sim})^{*}(\!|\, \{p\} \,|\!)$$

Again, we do this in terms of paths: the *context* of a path to an item is the set of paths to items providing context for its definition.

This final schema is the only schema that concerns us going forward: it contains all of the constraint information that we need. Each of the schema declarations in *RegistryObjects*, *RegistryPaths*, *RegistryUses*, *RegistryContains*, and—finally—*RegistryContext* has included a reference to the previous schema.

Purpose

Whether we are able to establish that one item refines another may depend upon the chosen purpose or context of application. Such a context may be described by a particular model, together with the models that it depends upon. In general, we may characterise a 'purpose' as a set of models, represented as a set of path names in the registry:

RegistryPurpose

$$RegistryContext \ldots$$
$$purpose : \mathbb{P}\, Path$$
$$(uses \cup contains^{\sim} \cup contains)(\!|\, purpose \,|\!) \subseteq purpose$$

This set should already be closed under *uses*, *contains*, and $contains^{\sim}$.

If our purpose relies upon the definition of an item, then it must rely also upon the definitions of other items that this definition refers to, the definitions that form part of its context, and the definitions of any items it contains.

We will write '$p1$ refines $p2$' to indicate that the item with path $p1$ has a link asserting that it refines the item with path $p2$. The definition of a class $p1$ may include the statement that it extends the definition of another class $p2$. Where this is the case, we will write '$p1$ extends $p2$'.

RegistryAssertions

$$
\begin{array}{l}
\textit{RegistryPurpose} \\
\_ \text{ refines } \_, \_ \text{ extends } \_ : \textit{Path} \leftrightarrow \textit{Path} \\
\hline
(\_ \text{ refines } \_) = \{\, p1, p2 : \textit{paths} \mid \exists\, t : \operatorname{dom} \textit{transformation} \bullet \\
\quad \textit{refines}\, p2 \in (\textit{item}\, p1).\textit{links} \lor \\
\quad \textit{refinesUsing}\, (t, p2) \in (\textit{item}\, p1).\textit{links} \,\} \\
(\_ \text{ extends } \_) = \{\, p1, p2 : \textit{paths} \mid p2 \in (\textit{item}\, p1).\textit{extends} \,\}
\end{array}
$$

We may now formalise the concept of a 'purpose closure' of a given set of paths. If P is a set of paths, we write $\overline{P}$ to denote its purpose closure, defined by

PurposeClosure

$$
\begin{array}{l}
\textit{RegistryAssertions} \\
\overline{} : \mathbb{P}\, \textit{Path} \nrightarrow \mathbb{P}\, \textit{Path} \\
\hline
\forall\, P : \mathbb{P}\, \textit{Path} \bullet \\
\quad \overline{P} = (\ (\_ \preceq \_) \cup (\_ \preceq \_)^{\sim} \cup (\_ \text{ refines } \_) \cup (\_ \text{ extends } \_)\ )^{*} (\!|\ P\ |\!)
\end{array}
$$

The purpose closure of P is the smallest set of paths whose constraints we are required to accept—the scope of the constraint information—for the purpose described by the set of paths P. It is the set of all paths that can be reached from paths in P by following refinement and extension links and/or selecting items in the same model, repeatedly.

We do not need to require consistency of a model that is still being edited or created, and which has yet to be finished, published, or used. With this in mind, we will introduce a notion of 'finalisation', with the intention that a model that has been 'finalised' is one that may no longer be edited or changed, and which is available for use within a registry. We can view initial models as mutable or *draft* so we can say:

$$\textit{Status} ::= \textit{draft} \mid \textit{finalised}$$

With this notion, we are able to set out a simple protocol for consistency which can be applied to finalised models:

1. once a model has been finalised, its contents are fixed or 'immutable'
2. *outgoing* refinement and extension links may be created only if the target item lies within a finalised model;

3. a model may be finalised only if its 'purpose closure' is consistent, containing no contradictions

More precisely, the purpose closure $\overline{M}$ of a model M is consistent if and only if it yields no logical contradictions—arising from structural or constraint properties—and no conflicting refinement assertions.

An inductive argument, using the properties of the registry language, can be used to establish that these constraints are enough to guarantee that:

– for any finalised model M, and the definition of any item within M will remain constant—even if other models, and other links, are added to the registry;

– for any purpose set B containing only paths to items in finalised models, the registry will yield only consistent information regarding refinement.

Containment

We may now define a notion of 'semantic refinement' between classifiers in the registry, again identified by their paths, based upon values, as opposed to the assertions of refinement included as links in the registry, which we might now think of us 'syntactic refinement'.

A classifier $p1$ is a semantic refinement of another classifier $p2$ if the definition of $p1$ is consistent with that of $p2$. In the case where no transformation, mapping, or conversion is applied, this means that:

– every value that $p1$ can take is also a possible value of $p2$

– for every value of $p1$, the explanation provided in the context of $p1$ is consistent with, or expands upon, the explanation of the same value provided in the context of $p2$

We will write $p1 \sqsupseteq p2$ to indicate that the item with path $p1$ is a semantic refinement of the item with path $p2$, defined as follows:

RegistryRefinement

$$
\begin{array}{l}
\mathit{RegistryDefinition} \\
\_ \sqsupseteq \_ : \mathit{Path} \leftrightarrow \mathit{Path} \\
\hline
\forall\, p1, p2 : \mathit{Path} \bullet p1 \sqsupseteq p2 \Leftrightarrow \\
\quad \mathrm{dom}(\mathit{definition}\, p2) \subseteq \mathrm{dom}(\mathit{definition}\, p1) \land \\
\quad \forall\, v : \mathrm{dom}(\mathit{definition}\, p1) \bullet \\
\qquad \mathit{definition}\, p1\, v \supset \mathit{definition}\, p2\, v
\end{array}
$$

We write '$\supset$' to denote logical implication between textual explanations. In the absence of any contradictions between the additional explanation $t3$ and either of the explanations $t1$ and $t2$, it should be the case that

$$(t1 \uplus t3) \supset (t2 \uplus t3) \;\Leftrightarrow\; (t1 \supset t2)$$

In our definition of $\sqsupseteq$, this corresponds to the assumption that the explanation of each of the possible values does not conflict with the explanation of either classifier. Where this is the case, the defining constraint can be simplified:

$$\forall\, p1, p2 : \mathit{Path} \bullet p1 \sqsupseteq p2 \Leftrightarrow$$
$$\mathit{values}\, p2 \subseteq \mathit{values}\, p1 \wedge \mathit{completeText}\, p1 \supset \mathit{completeText}\, p2$$

In practice, of course, refinement between classifiers will simply be asserted by authors rather than derived from a formal analysis of the constraints and textual explanations involved. A user will add a link to the registry to indicate that, in their opinion, one classifier is a refinement of another. Naturally, we would hope that the assertions made would be consistent with our ideal 'semantic' interpretation of the text, but we could not hope to guarantee this.

Our purpose here is somewhat simpler. If this is the underlying, semantic notion, that we hope users will reflect in their addition of assertions to their models, then we can usefully compare it to our interpretation of class extension.

The registry language does not include any explicit representation of behaviour or functionality. Even where a corresponds to an abstraction of a program or a workflow, the classes of the model will have no explicit behavioural properties: no methods, functions, or operations will be described. Our notion of inheritance is therefore based entirely upon the textual explanations and constraints present in class definitions.

The semantics that we have given to inheritance in our modelling language takes into account the fact that some of the constraint information and textual explanation pertaining to a particular class may not appear in the properties of the class. It may instead appear in the definition of any item that forms part of the context for the class in question.

If $p1$ and $p2$ are paths leading to classes in a model, then

$p1$ extends $p2$

$$\Rightarrow p2 \in (\mathit{item}\, p1).\mathit{extends} \qquad [\mathit{RegistryStatements}]$$
$$\Rightarrow p2 \in (\mathit{class}\, p1).\mathit{extends} \qquad [\mathit{RegistryItems}]$$
$$\Rightarrow p1 \mapsto p2 \in \mathit{uses} \qquad [\mathit{RegistryUses}]$$

$$\Rightarrow p2 \in \mathit{context}\ p1 \qquad [\mathit{RegistryContext}]$$

$$\Rightarrow \mathit{values}\ p1 \subseteq \mathit{values}\ p2 \land \qquad [\mathit{RegistryValues}]$$

$$\mathit{completeText}\ p1 \supset \mathit{completeText}\ p2 \qquad [\mathit{RegistryText}]$$

$$\Rightarrow \mathrm{dom}(\mathit{definition}\ p2) \subseteq \mathrm{dom}(\mathit{definition}\ p1) \land$$

$$\forall\, v : \mathrm{dom}(\mathit{definition}\ p1) \bullet$$

$$\mathit{definition}\ p1\ v \supset \mathit{definition}\ p2\ v$$

$$[\mathit{RegistryDefinition}]$$

$$\Rightarrow p1 \sqsupseteq p2 \qquad [\mathit{RegistryRefinement}]$$

The implication for the step with labels *RegistryValues* and *RegistryText* relies upon three pieces of information:

1. *context* is defined in terms of a reflexive transitive closure, which it is;
2. $\bigwedge$ is monotonic with respect to logical implication on formal constraints, which is again the case;
3. $\uplus$ is monotonic with respect to our notion of logical implication on textual explanations.

The last of these is something that might reasonably be expected of our textual explanations and their interpretation, but which clearly cannot be guaranteed in practice. However this now forms a basis for us to define a number of rules regarding the specification model that we have built up.

4.3 Inference rules

Provided that our textual explanations are reasonable or sensible enough in this respect, the proof above is enough to show that:

Rule 1 (Extension) For paths $p1$ and $p2$ and purpose set B:

$$\frac{p1 \text{ extends } p2 \ \land\ p1 \in B}{B \ \vdash\ p1 \multimap p2}$$

As suggested by our use of the hollow-headed arrow symbol, our underlying 'semantic' notion of refinement, based upon values and interpretations, contains our notion of inheritance.

For this reason, we might sensibly re-use the extension symbol, and perhaps also the name, for our 'definition refinement' assertions. We can also state that then it is enough to believe the definition in question to conclude that the refinement holds:

Rule 2 (Extension) For paths $p1$ and $p2$ and purpose set B:

$$\frac{p1 \sqsupseteq p2 \ \land \ p1 \in B \ \land \ p2 \in B}{B \ \vdash \ p1 \rightarrowtriangle p2}$$

The relationship between purpose and semantic interoperability can be seen more clearly if we adopt a simple calculus-style presentation. We begin by defining the mapping between a particular purpose set *purpose* and the resulting refinement relation between definitions:

Refinement

$$\begin{array}{|l}
\textit{RegistryStatements} \\
_ \rightarrowtriangle _ : \textit{Path} \leftrightarrow \textit{Path} \\
\hline
_ \rightarrowtriangle _ = (\ (\sqsubseteq \cup \text{refines} \cup \text{abstracts}^{\sim}) \cap (\textit{purpose} \times \textit{purpose})\ )^{*} \\
\end{array}$$

The relation $\rightarrowtriangle$ is formed as the reflexive transitive closure of the intersection of *purpose* $\times$ *purpose* with the union of two relations: 'refines' and the inverse of 'abstracts'. Two paths are related under this closure if and only if there is a sequence of steps from one to the other, using only pairs for which both components of the pair lie in the purpose set.

For any paths $p1$ and $p2$, and for any purpose set B, we will write

$$B \ \vdash \ p1 \rightarrowtriangle p2$$

to indicate that, within the current registry, $p1$ is a refinement of $p2$, if the definitions with paths in B are accepted. In terms of the defining schema, the above statement is equivalent to

$$\textit{Refinement}[B/\textit{purpose}] \Rightarrow p1 \rightarrowtriangle p2$$

A fully-formal definition, in the Z notation, can be obtained by introducing the notion of a fact or sequent for the right-hand argument of the turnstile entailment relation $\vdash$; here, the above semi-formal notion of equivalence should suffice.

If the definition of an item includes the assertion that it refines another item, then it is enough to accept the definition in question to conclude that the refinement holds:

Rule 3 (Refinement by assertion) For paths $p1$ and $p2$ and purpose set B:

$$\frac{p1 \text{ refines } p2 \ \wedge \ p1 \in B}{B \ \vdash \ p1 \rightharpoonup p2}$$

It is not necessary to require also that $p2 \in B$: if $p1 \in B$ and $p1$ refers to $p2$, then $p2 \in B$, as purpose sets are closed under definition re-use.

If the definition of an item includes the assertion that it abstracts another item, and we are to accept the definition of that other item, then we may conclude that refinement holds in the other direction:

Rule 4 (Abstraction by assertion) For paths $p1$ and $p2$ and purpose set B:

$$\frac{p1 \text{ abstracts } p2 \ \wedge \ p1 \in B \ \wedge \ p2 \in B}{B \ \vdash \ p2 \rightharpoonup p1}$$

The fact that $p2$ needs to be included in the purpose set for the conclusion to hold, reflects the fact that although purpose sets are closed under 'refines', they do not need to be closed under 'abstracts'.

Closure of purpose sets

Our formal definition for a registry includes the requirement, in the constraint of schema *RegistryPurpose*, that any purpose set should be closed under the re-use of definitions, whether this is achieved through refinement or through extension. From this, we may derive the following rules:

Rule 5 (Closure under re-use) For paths $p1$ and $p2$ and purpose set B,

$$\frac{p1 \in B \ \wedge \ p1 \text{ uses } p2}{p2 \in B}$$

where 'uses' is either of 'refines' or 'extends'.

If path $p1$ is a prefix of path $p2$, then the item that $p2$ points to is a subcomponent of the item that $p1$ points to. It would be inconsistent to accept the whole of a definition, and yet not

accept some part of it. The same closure property given above allows us to derive the following rule:

Rule 6 (Downward closure) For paths $p1$ and $p2$ and purpose set B,

$$\frac{p1 \in B \ \wedge \ p1 \preceq p2}{p2 \in B}$$

Another requirement is that purpose should be closed in the opposite direction. This reflects the idea that classes, attributes, and enumerations should not be defined and managed independently, but only as part of coherent data models:

Rule 7 (Upward closure) For $p1$ and $p2$ and purpose set B,

$$\frac{p1 \in B \ \wedge \ p2 \preceq p1}{p2 \in B}$$

Any of the items that contain the item with path $p1$ may include textual explanations or logical constraints that refer directly to item $p1$.

In combination, these three rules confirm that the context for a given definition includes the definition of every item in the same model, and every item in every model that these definitions refer to, directly or indirectly, with refinement or extension links.

Where a purpose set entails several requirements, we may choose to present a single entailment in which those requirements appear as a common-separated list, rather than write out a conjunction of entailments: for example,

$$B \ \vdash \ r1, r2, \ldots \quad \Leftrightarrow \quad B \ \vdash \ r1 \ \wedge \ B \ \vdash \ r2 \ \wedge \ \ldots$$

We will use this convention in our next rule, which is again a consequence of the definitions of $\multimap$ and $\vdash$:

Rule 8 (Combination) For paths $p1$, $p2$, $p3$, and $p4$, and purpose sets B and C,

$$\frac{B \ \vdash \ p1 \multimap p2 \ \wedge \ C \ \vdash \ p3 \multimap p4}{B \cup C \ \vdash \ p1 \multimap p2, \ p3 \multimap p4}$$

As $\multimap$ is defined as a reflexive, transitive closure, we have also the following rule:

Rule 9 (Transitivity) For paths $p1$, $p2$, and $p3$, and purpose set B,

$$\frac{B \ \vdash \ p1 \multimap\!\triangleright \, p2, \ p2 \multimap\!\triangleright \, p3}{B \ \vdash \ p1 \multimap\!\triangleright \, p3}$$

Together, these rules confirm that we can add paths—and hence new content—to a registry without detracting from the information that it already contains, and that by adding new content that includes refinement or abstraction links, we can provide more information about any item that is already in the registry. We may express this as a further, derived rule: one that is an immediate consequence of Rules 8 and 9.

Rule 10 (Composition) For paths $p1$, $p2$, and $p3$, and purpose sets B and C,

$$\frac{B \ \vdash \ p1 \multimap\!\triangleright \, p2 \ \wedge \ C \ \vdash \ p2 \multimap\!\triangleright \, p3}{B \cup C \ \vdash \ p1 \multimap\!\triangleright \, p3}$$

Graphical illustration

Although every 'outgoing' semantic link forms an inherent part of the definition of the item in question, the same is not true of all 'incoming' semantic links—links that refer to this item, but are properties of some other item. If the definition of that other item sits in another model, and that model is not part of our purpose set, we can choose whether or not to accept it.

To illustrate this and other situations, we will use decorated forms of our refinement symbol $\multimap\!\triangleright$ to represent the various forms of direct semantic link that may be introduced. We will add a solid disc to indicate an outgoing semantic link, corresponding the inclusion of a *refines* link in the *links* property of the source item. In text, the resulting symbol could equally serve as a synonym for 'refines': for any paths $p1$ and $p2$, we have that

$$p1 \bullet\!\multimap\!\triangleright \, p2 \ \Leftrightarrow \ p1 \text{ refines } p2 \qquad (\Leftrightarrow \ \mathit{refines}\, p2 \in (\mathit{item}\, p1).\mathit{links}\ )$$

We will add a hollow disc to indicate an incoming link: one in which the assertion is a property of the target item. In text, the resulting symbol could equally serve as a synonym for *the inverse of* 'abstracts': for any paths $p1$ and $p2$,

$$p1 \circ\!\multimap\!\triangleright \, p2 \ \Leftrightarrow \ p2 \text{ abstracts } p1 \qquad (\Leftrightarrow \ \mathit{abstracts}\, p1 \in (\mathit{item}\, p2).\mathit{links}\ )$$

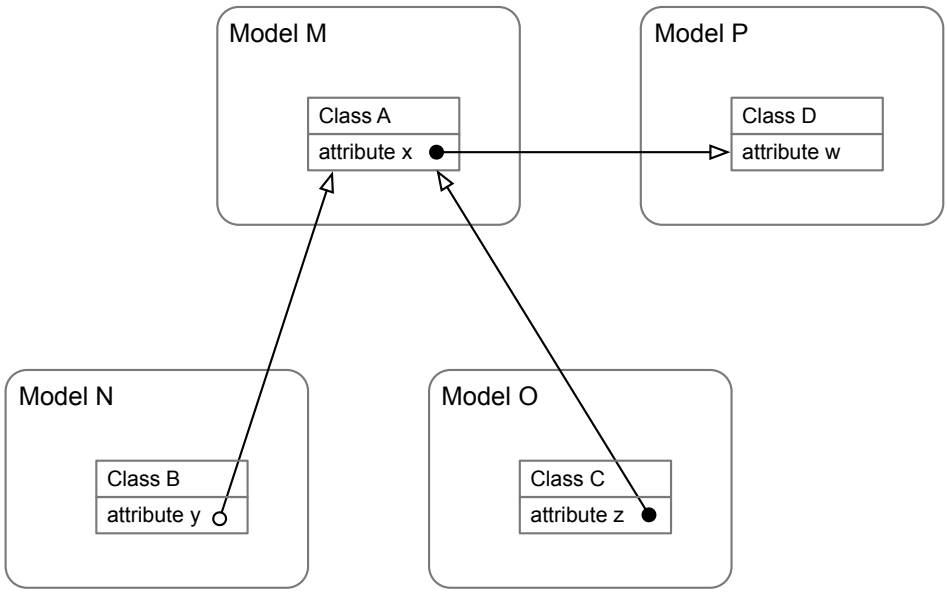

Figure 4.6: Picturing refinement

Figure 4.6 illustrates a scenario in which the definition of attribute M.A.x includes an abstraction link to attribute N.B.y and a refinement link to attribute P.D.w—indicated by the arrow on the right with a hollow disc, and the horizontal arrow on the left with a solid disc—and the definition of another attribute, O.C.z, includes a refinement link to M.A.x. For economy, we will refer to these attributes below using the attribute part of the name: that is, we will write x for M.A.x.

The definition of attribute x re-uses that of attribute w, in terms of the outgoing refinement link. If we wish to make use of the definition of x, we must include in our purpose set not only the model M, in which x is defined, but also the model P, in which w is defined. A registry with these contents would tell us that

$$\mathsf{M} \cup \mathsf{P} \vdash \mathsf{x} \twoheadrightarrow \mathsf{w}$$

The other models provide no additional context for the definition of x: we can use the definition of x whether or not we include model N or model O within the scope of our purpose set.

In the case of attribute y, the refinement link is incoming: the statement that the definition of y refines that of x is made in model M. There are no outgoing links from N to O, corresponding to definition usage or class extension, and so we are free to accept N without also accepting M. Simply accepting N would give us merely the definition of y, with no additional refinement information, whereas accepting also M and P would give us

$$\mathsf{N} \cup \mathsf{M} \cup \mathsf{P} \vdash \mathsf{y} \twoheadrightarrow \mathsf{x},\ \ \mathsf{x} \twoheadrightarrow \mathsf{w}$$

and hence also that $\mathsf{y} \multimap \mathsf{w}$.

In the case of attribute z, the refinement link to x is outgoing, as is the onward refinement link to w. We may use the definition of x without needing to accept model O, just as we have above:

$$\mathsf{M} \cup \mathsf{P} \vdash \mathsf{x} \multimap \mathsf{w}$$

If we are content to accept model O, however, we may infer also that

$$\mathsf{O} \cup \mathsf{M} \cup \mathsf{P} \vdash \mathsf{z} \multimap \mathsf{x}, \ \mathsf{x} \multimap \mathsf{w}$$

and hence also that $\mathsf{z} \multimap \mathsf{w}$.

In this scenario, we might imagine that models N and O corresponds to artefacts used for data collection, such as forms, schemas, or databases, that model M corresponds to a data standard, and that model P corresponds to some usage or application of data. If we include all four of the models within our purpose, the registry tells us that:

- data collected against the definition of x within model M will also meet the definition of w within P;
- data collected in N, against the definition of y, will also satisfy the definition of x within M;
- data collected by O, against attribute z, also meets the requirements of x within M;

and hence that data collected by either N or O, against y or z, respectively, can be combined and/or used as values of w within P.

The registry has support for negative assertions: statements that refinement, or abstraction, does not hold. We may define a negative counterpart $\not\multimap$ to our refinement relation $\multimap$, and write

$$B \vdash p1 \not\multimap p2$$

to indicate that the current registry tells us that, given purpose set B, the item with $p1$ is not a refinement of the item with path $p2$. Formally,

Negation

RegistryStatements

$\_ \nrightarrow \_ : Path \leftrightarrow Path$

$$\_ \nrightarrow \_ = (((purpose \times purpose) \setminus \sqsupseteq) \cup \text{doesNotRefine} \cup \text{doesNotAbstract}^{\sim}) \cap (purpose \times purpose)$$

Note that this relation is defined simply as the union of three relations, constrained to apply only to paths appearing in *purpose*: unlike the definition given in *Refinement*, it does not make use of reflexive transitive closure *.

Using this definition, in the context of the properties set out in the schema *RegistryStatements*, we may derive a set of rules broadly similar to those defined above for $\nrightarrow$. As the form of the rules is now familiar, we may save space by omitting the information that $p1$, $p2$, $p3$, and $p4$ denote paths and that B and C denote purpose sets.

Rule 11 (Negation definition)

$$\frac{\neg\ (p1 \sqsupseteq p2)\ \land\ p1 \in B\ \land\ p2 \in B}{B\ \vdash\ p1 \nrightarrow p2}$$

Rule 12 (Negation by assertion)

$$\frac{p1\ \text{doesNotRefine}\ p2\ \land\ p1 \in B\ \land\ p2 \in B}{B\ \vdash\ p1 \nrightarrow p2}$$

Rule 13 (Negation by inverse assertion)

$$\frac{p1\ \text{doesNotAbstract}\ p2\ \land\ p1 \in B\ \land\ p2 \in B}{B\ \vdash\ p2 \nrightarrow p1}$$

As we might expect, this relation is not transitive. It does, however, satisfy the following pair of rules:

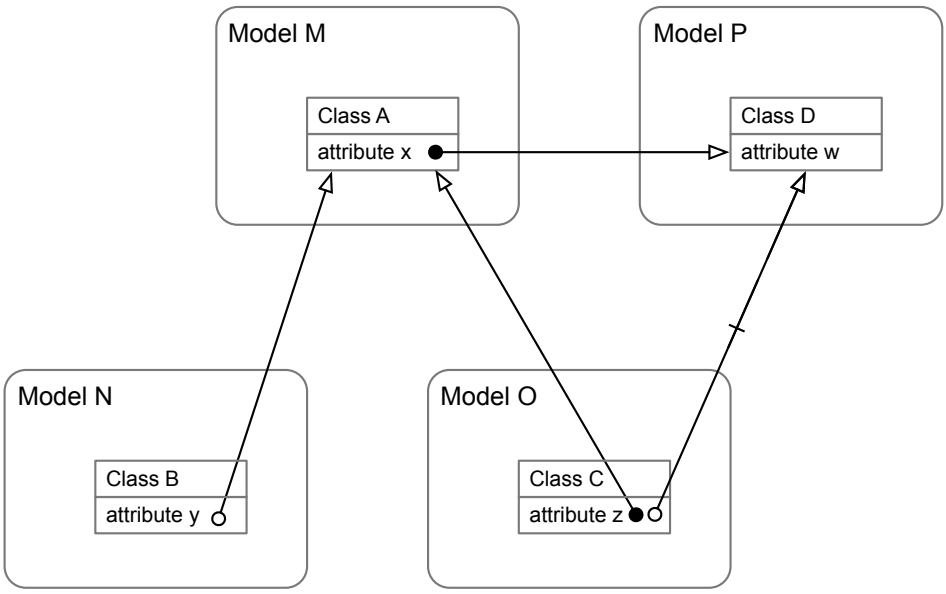

Figure 4.7: Conflicting assertions

Rule 14 (Combination of negation)

$$\frac{B \vdash p1 \nrightarrow p2 \ \wedge \ C \vdash p3 \nrightarrow p4}{B \cup C \vdash p1 \nrightarrow p2,\ p3 \nrightarrow p4}$$

Rule 15 (Combination of refinement and negation)

$$\frac{B \vdash p1 \rightarrow p2 \ \wedge \ C \vdash p3 \nrightarrow p4}{B \cup C \vdash p1 \rightarrow p2,\ p3 \nrightarrow p4}$$

The second rule admits an immediate corollary, showing that it is perfectly feasible to add conflicting information about existing items by adding new paths to the purpose set.

Rule 16 (Introduction of conflict)

$$\frac{B \vdash p1 \rightarrow p2 \ \wedge \ C \vdash p1 \nrightarrow p2}{B \cup C \vdash p1 \rightarrow p2,\ p1 \nrightarrow p2}$$

As an illustration of how this might arise in practice, consider the scenario shown in Figure 4.7. In this scenario, in addition to the links shown in Figure 4.6, there is a negative

abstraction link from P.D.w to O.C.z. We will decorate the 'does not refine' arrow with a hollow disc $\circ\!\!\dashv\!\triangleright$ and a solid disk $\bullet\!\!\dashv\!\triangleright$ to represent incoming and outgoing assertions, respectively.

If we accept O, then we must accept also M and P, as purpose sets are closed under outgoing refinement assertions. This leads us to the following entailment

$$\mathsf{O} \cup \mathsf{M} \cup \mathsf{P} \vdash \mathsf{z} \multimap \mathsf{w}, \quad \mathsf{z} \dashv\!\triangleright \mathsf{w}$$

Here, the registry is presenting us with conflicting information on refinement between attributes z and w.

This does not prevent us from obtaining consistent information from the registry regarding the relationships between other attributes. The negative abstraction link is an incoming assertion, owned by model P; it does not come into play unless we accept also model O. If we accept the other three models, then we may reliably establish that data from y can be re-used in w:

$$\mathsf{N} \cup \mathsf{M} \cup \mathsf{P} \vdash \mathsf{y} \multimap \mathsf{w}$$

Recall that, from the previous section, that a model may be finalised only if its 'purpose closure' is consistent, containing no contradictions. With this in mind, we should consider again the scenario of Figure 4.7. Model P must be finalised before model M, as there is an outgoing refinement link from x to w. Model N can be finalised without reference to any of the other models, provided that the constraint properties of its items contain no logical inconsistencies.

Model O, however, is not self-consistent. The outgoing assertion from z to x places M and hence P in the purpose closure:

$$\overline{\mathsf{O}} = \mathsf{O} \cup \mathsf{M} \cup \mathsf{P}$$

and the outgoing semantic assertion conflicts with the negative assertion made in P:

$$\overline{\mathsf{O}} \vdash \mathsf{z} \multimap \mathsf{w}, \quad \mathsf{z} \dashv\!\triangleright \mathsf{w}$$

Model O cannot then be finalised without violating Constraint 4 above. The imposition of these constraints ensures that such a scenario cannot occur with all four models finalised.

4.4 Extension and refinement

As well as providing a basis for reasoning about the consistency of information within a model-based registry, our formal semantics will allow us to demonstrate that definition refinement is consistent with class extension or inheritance as has been documented in [14]. To do this, we consider how each definition can be characterised in terms of a mapping between allowed values and textual explanations, representing the 'real world' semantic content.

Values and textual explanations

A value may be: an element of a primitive type, such as numbers or strings; a term, or an item in an enumeration; or an instance of a class. We will refer to the last of these as an 'object value'.

$$\begin{aligned} Value ::= {} & primitiveValue\langle\langle Type\rangle\rangle \mid termValue\langle\langle Term\rangle\rangle \\ \mid {} & enumValue\langle\langle Enum\rangle\rangle \mid objectValue\langle\langle Name \times Value\rangle\rangle \end{aligned}$$

The logical constraint defining the set of possible values for a classifier, as a subset of this type, will be formed as the conjunction of all of the logical constraints in the context of its definition.

A registry provides definitive descriptions for a set of classifiers by associating each classifier with a set of possible values, subject to a logical constraint, and a textual explanation. For convenience, we will introduce a function *constraint* to indicate the constraint information associated with each classifier:

RegistryConstraint

$$\begin{aligned} & RegistryContext \\ & constraint : Path \pfun Constraint \\ \hline & constraint = \\ & \quad \{p : \mathrm{dom}\, model \cap \mathrm{dom}\, class \cap \mathrm{dom}\, attribute \bullet \\ & \qquad p \mapsto (item\, p).constraint\} \end{aligned}$$

and a function *values* to indicate the set of possible values:

RegistryValues

$$
\begin{array}{l}
RegistryConstraint \\
values : Path \pfun \mathbb{P}\ Value \\
\hline
\forall\, p : \mathrm{dom}\ attribute \bullet (\exists\, t : Path \mid t = (attribute\ p).type \bullet \\
\quad t \in \mathrm{dom}\ enumeration \Rightarrow \\
\quad\quad values\ p = enumValue(\!|\ (enumeration\ t).enums\ |\!) \cap \\
\quad\quad\quad extent\ (\bigwedge(constraint(\!|\ context\ p\ |\!))) \wedge \\
\quad \ldots \\
\quad t \in \mathrm{dom}\ class \Rightarrow \\
\quad\quad values\ p = values\ t \cap extent\ (\bigwedge(constraint(\!|\ context\ p\ |\!)))) \\
\forall\, p : \mathrm{dom}\ class \bullet values\ p = \\
\quad objectValue(\!|\ \{n : Name;\ v : Value \mid \exists\, a : (class\ p).attributes \bullet \\
\quad\quad n = a.name \wedge v \in values\ a.path\}\ |\!) \cap \\
\quad\quad\quad extent\ (\bigwedge(constraint(\!|\ context\ p\ |\!))) \\
\ldots
\end{array}
$$

Here we show only the values for classes, and for enumeration- and class-valued attributes; the values for models, and for term- and primitive-valued attributes, are defined in an entirely similar fashion.

We will introduce also a function *text* to indicate the textual explanation directly associated with each item, and a related function *completeText* to indicate the 'complete' textual explanation for that item, being that obtained by including also the explanations for the enclosing items that provide additional context:

RegistryText

$$
\begin{array}{l}
RegistryValues \\
text, completeText : Path \pfun Text \\
\hline
text = \{p : paths \bullet p \mapsto (item\ p).text\} \\
completeText = \{p : paths \bullet p \mapsto \uplus\,(text(\!|\ context\ p\ |\!))\}
\end{array}
$$

Here, $\uplus$ is used to indicate the combination of a set of textual explanations.

The path associated with a particular value is given by

RegistryValuePath

$$
\begin{array}{l}
RegistryText \\
valuePath : Value \pfun Path \\
\hline
\forall\, v : Value \bullet \\
\quad v \in \operatorname{ran} primitiveValue \Rightarrow \\
\quad\quad valuePath\ v = (primitiveValue^{\sim}\ v).path \land \\
\quad v \in \operatorname{ran} termValue \Rightarrow \\
\quad\quad valuePath\ v = (termValue^{\sim}\ v).path \\
\quad \ldots
\end{array}
$$

We may then characterise the description of a classifier, its definition in the context of a registry, as a function from possible values to textual explanations.

RegistryDefinition

$$
\begin{array}{l}
RegistryValuePath \\
definition : Path \pfun (Value \pfun Text) \\
\hline
\forall\, p : \operatorname{dom} model \cap \operatorname{dom} class \cap \operatorname{dom} attribute \bullet \\
\quad definition\ p = \{v : values\ p \bullet \\
\quad\quad v \mapsto completeText\ p \uplus completeText\ (valuePath\ v)\}
\end{array}
$$

The logical constraint that defines *values p* depends upon the context of the classifier, and the associated text depends also upon the context of the value.

Example

The definition of the association *employedBy* within the model *Employment* has the following context within the registry:

$$
\begin{aligned}
&context\ \langle Employment, \mathbf{Person}, employedBy\rangle \\
&\quad = (uses \cup contains^{\sim})^{*}(\!|\ \{\langle Employment, \mathbf{Person}, employedBy\rangle\}\ |\!) \\
&\quad = \{\langle Employment\rangle, \langle Employment, \mathbf{Person}\rangle, \\
&\qquad \langle HMRC, \mathbf{Person}, employeeOf\rangle, \langle HMRC\rangle, \langle HMRC, \mathbf{Person}\rangle\}
\end{aligned}
$$

The reflexive transitive closure of the union of the usage and containment relations—the latter reversed here—maps this path to a set of paths within the *Employment* and *HMRC* models.

This set consists of the paths of items whose definitions have been re-used, together with the paths of any other items that contain them.

This is the context in which the definition of *employedBy* is made, according to the registry. We can obtain all of the relevant text and all of the relevant constraint information as the relational image of this set under the functions *text* and *constraint*, respectively. The latter will give us the set of values that may be associated with *employedBy*: in this case, a set of object values satisfying the constraints of *Employment*.**Organisation** and *HMRC*.**Organisation**.

Chapter 5

Metadata Modelling Language (MDML)

This chapter develops a metamodel using standard model driven engineering techniques, based on the work in chapter 4, which will used in the later experimental work documented in chapters 6 and 7.

5.1 Overview

In the next stage of this work a grammar was developed using XText which was called the MDML and from this the most recent metadata registry was developed. This grammar is the basis of a software application which implements the ideas from chapter 4 as a working java program. The software application is then used in chapter 6 to model some medical datasets and perform matching between hetreogeneous datasets.

The XText domain model included all the constructs from column 3 of Table 6.1, and provided a grammar which was then used to generate a set of domain objects, which in turn were able to represent the various healthcare datasets being used. The core enabled the automatic generation of grails domain classes using the code generation capabilities of XText. For illustrative purposes we show the key metadata registry structure in Figure 5.1.

The structure illustrated here allows the metadata registry to capture data elements, a feature that is at the core of the ISO11179 standard, and one that is at the core of the theoretical specification. It also allows each element to have a status, which allows it to conform to the publishing aspects of the standard. By making each item in the registry a class inheriting from

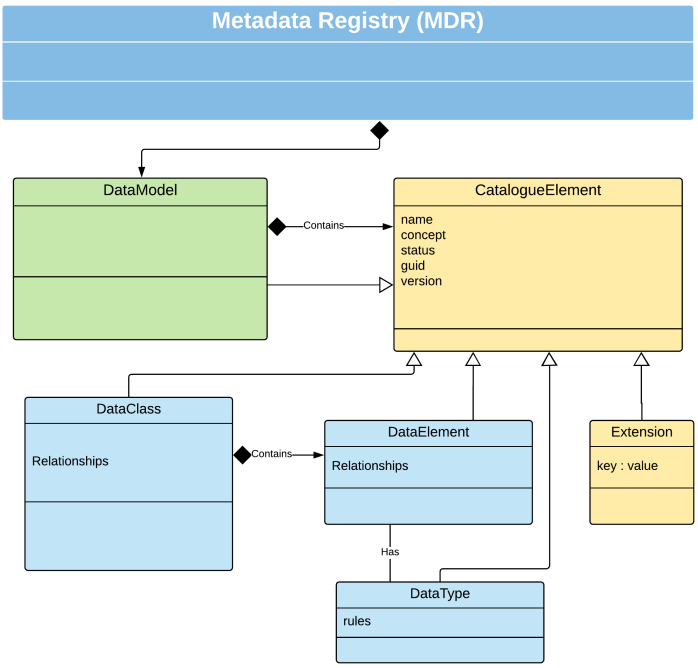

Figure 5.1: Core MDR (Ecore) Metamodel

an *abstract Catalogue Element* each data element is captured by a unique identifier and a status relating to its publishing status (i.e. *draft* or *finalized*).

In the ISO standard metamodel the data element is the primary artefact under consideration, however there is no obvious direction on how groups of data elements should be handled. In practice data elements appear in groups, whether spreadsheets, database tables or CSV files. In the data analysis business, groups of data need to be brought together and to do this datasets need to be mapped and transformed. In our first and second models it was easy to generated lists of data elements, however they were nearly always grouped using several layers of hierarchy into different entities, which made mapping and transformation difficult.

Further a DataModel is able to add in any number of extensions, which can be used to augment particular data elements for instance to mark them as user interface items so that user interfaces can be automatically generated. In addition imports can be added from other DataModels, so that new datasets can be created using data elements from several different datasets. This metamodel evolved over a period time in order to solve the problems of integrating data from a wide variety of heterogeneous sources, it is closely based around UML/Ecore, but is a domain specific language in its own right. It can also be thought of as a metamodel at level 2 of the MOF hierarchy, the same level as the metamodel specified in part 3 of ISO11179.

5.2 Core Elements of MDML

The main constructs of DataClasses, DataElements and DataTypes are common to most technical spaces, and can be used to group and manage data elements in way that is intuitively easy for data analysts and developers to grasp. This contrasts with concepts such as *Conceptual Data Element* and *Enumerated Value Domain*, which are specialist terms Metadata can then be attached to any of these core elements, so that security classifications, and governance directives, found in other parts of the ISO11179 standard can easily be added to particular DataElements or DataClasses.

As well as implementing the notions of consistency identified in the previous chapter the design of MDML allows metadata to be captured in any form by using the *Extension* model whereby ad-hoc metadata can be added to any catalogue element as key-value pairs.

The core metamodel allows datasets to be grouped and classified in a flexible and manageable way, so that one registry could hold several common healthcare metamodels, say FHIR, Observational Medical Outcomes Partnership (OMOP) and LOINC, and build local models based on all three. There is a clear identification system built into the language which means that each version is separately identified, so that any mappings between FHIR and OMOP for instance will be detailed to particular versions of the datasets.

There is a clear publishing cycle, identified in the previous chapter, built into the metamodel, which allows change for *draft* models, but not for *finalized* models, therefore data elements in finalized models can be imported into other (draft) datamodels, but no new elements can be imported in currently finalized models. This is an important management feature, derived from the ISO standard which has been built into the language.

MDML - Implementation

In defining the meta-language the notion of a *dataElement* is taken as one of the core entities in the new meta-language, it is intended to correspond with a *data element* in ISO11179 and with an *element* in the UML meta-model. It is an atomic data item, capturing one single element that cannot be sub-divided. Within a *Model* it is the *atomic* piece of data.

The word or stem *abstract* is used in several different ways in this discussion, and the following brief discussion highlights the different usages.

Firstly, there can be an abstract entity which is the implementation of an EClass at the M3 level, that is to say a representation of in MDML of an EClass which cannot be implemented at the M2 level. It can be *sub-typed* by another EClass at the M2 level and this *sub-class*

can be implemented. This mechanism is used to specify both AbstractItem and DataItem as *abstract* entities in the language. Secondly, each layer is referred to as being an *abstraction* of the previous layer, and therefore having less concrete description, which is the notion built into the idea of MOF layers.

XText Implementation

XText is a language workbench, built around Ecore and the Eclipse environment; it uses the ANother Tool for Language Recognition (ANTLR) [70] parser, which is a widely used language toolkit. On its own ANTLR would be a good choice to develop MDML, it allows a grammar to be built and a parser to be generated from the grammar. There are a number of other artefacts which are needed to develop and these can be provided by using

The XText language development environment, includes ANTLR as a component. XText also provides the ability to integrate with ECore, it generates an ECore model automatically from the abstract syntax tree, in a form that enables it to integrate with the eclipse modelling toolkit.

The full Ecore model is derived from the following grammar, which is listed below in this section. In the following section the Ecore model is examined in detail.

In MDML, the core artefact is the DataModel, something roughly akin to a package in java, or a schema in a relational database model; we are allowing many DataModels within our domain - which we are calling a DomainRegistry, and we represent this in XText as follows:

```
DomainRegistry :
(registeredItems += DataModel)*
;
```

This definition is simply defining the item of a DomainRegistry as being something which can contain zero to many registered items, each of which must be a DataModel. We then move on to define a DataModel:

```
DataModel:
'DataModel' name = QualifiedName 'status:' status = Status 'domainid:'
guid = GUID  '@' '(' Version ')' refines=(REFINE) '{'
(elements += DataItem)*
(relations += Relationship)*
(constraint += Constraint)*
```

```
(predicate += Predicate)*
'}'
;
```

A DataModel is defined by a qualified name and a 'modelid' composed of a unique id and a version. The QualifiedName needs to be an identifier, the id needs to be an integer and the version is a semantic versioning [67] identifier (e.g. 0.1.2 or X.Y.Z) . Semantic versioning uses the convention that the first digit indicates a new version, the use of which will break previous codebases, the second digit indicates new functionality added in a backward compatible manner, and the third digit indicates a backward compatible update to the model. The refines attribute is in fact optional, if used it provides a reference either to the GUID of a dataitem which *refines* this datamodel, or one which does not refine this datamodel. The versioning system has its origins in software development, but it provides a convenient formally referenced system, which combined with the unique identifier gives users an exact reference for a particular data item definition. Constraints and Predicates can also be added into a datamodel, the idea being that constraints can be composed of references to one or more dataitems and a predicate. Constraints can then be build up so that, for instance one data must occur another date using a combination of references and predicates.

```
REFINE:
(REFINES | DOESNOTREFINE)?
;

REFINES:
'refines' ':' Path
;

DOESNOTREFINE:
'doesnotrefine' ':' Path
;
```

The rule for *REFINE* states that the attribute can be either *REFINES* or *DOESNOTREFINE*, both being named paths, in essence a reference to a URI of a particular finalised data item in the registry.

```
DataItem:
```

```
dataitem = (DataModel | DataClass | DataElement | DataType
| Import | ExtensionItem  )
;
```

We take the notion of an DataItem from the previous section (section 4.3.1.1), but we are reducing the scope of DataItem to just 5 entities: DataModel, DataClass, DataType, Import and ExtensionItem. The ExtensionItem is the entity which is modelling the tag entity referred to at the start of that section, several of the other entities,which from the schema might be expected here are defined later on in this grammar.

```
QualifiedName:
ID('.' ID)*
;

Path:
QualifiedName('/' QualifiedName)*
;


GUID:
ID(('.' ID)|('/'))*
;
```

Here we are relying on an imported definition, instead of defining our own basic identifier we can use those previously defined in a 'Terminals' grammar which is available as part of the XText workbench. The item here is simply specifying that an identifier can be composed of a simple identifer, or it can consist of several identifiers joined up using a dot notation.

```
Version:
INT '.' INT '.' INT
;
```

This is the version, for which we are applying the formal definition of a semantic version as discussed earlier, basically this part of the grammar is saying that this version should be defined by 3 integers (as defined in out Terminal's grammar and imported into this grammar) separated by dots.

```
Reference:
"ref" name=ID ":" type=[AbstractItem]
;
```

Here we define a reference, by introducing it using the nomenclature *'ref'*, following this by a name and a type, which can be any other AbstractItem type.

The next 3 lines in the grammar need to be looked at together, we are defining a way to express relations, which were discussed in section 4.3.1.2. A relationship consists of 3 parts, a source dataitem, a destination dataitem and the relationship dataitem itself. We are defining the roles specified in section 4.3.1.2 here as source and destination, which indicates that they are directional in nature. By defining the relationship as a separate dataitem we are able to tag it, or rather add a number of extensionItems to it, each of which provides a key-value pair of data. This allows us to provide for unspecified relationships between dataitems, which may be needed if we are transforming data from one format or dataset into this MDML structure.

```
RelationSource:
DataModel | DataClass | DataElement | DataType
;
```

A relationship in this metamodelling language can be defined between any two data-items (DataModel, DataClass or DataType) and to do so we define the relation as a one-way relationship, having a source(above) and a destination(below).

```
RelationDestination:
DataModel | DataClass | DataElement | DataType
;

Relationship:
'relationship' (src = RelationSource ':' dest = RelationDestination)
;
```

The next item is the ExtensionItem which allows us to add random items of data to any other dataitem in the model, and it is this very simple feature which becomes very useful in the transformation aspects of the language. If we have registry of models we can then query and order it using the key-value pairs stored in the ExtensionItem, thus we can capture information whichwould otherwise be lost during the transformation process.

One of the core features of the language is to capture secondary contextual information from data stored in one format and use it in the transformation to another format. To do this an extension mechanism is required that will allow such metadata to be captured and managed separate to the core data model.

As an example XML Schema has the notion of a *union*, data can be captured but the type of the instance data may be one of several difference options. The following snippet is taken from the *basicTypes.xsd* from the GML core XML Schema definitions:

```
<simpleType name="NullType">
<annotation>
<documentation>Utility type for null elements.
The value may be selected from one of the
enumerated tokens, or may be a URI in which
case this should identify a resource which
describes the reason for the null.
</documentation>
</annotation>
<union memberTypes="gml:NullEnumeration anyURI"/>
</simpleType>
```

This allows null values to be specified by using the NullEnumeration or using a URI. There is no obvious counterpart when defining a type using Ecore or for that matter MDML, therefore this information can be captured using the extension mechanism, enabling two-way transformations to occur.

```
ExtensionItem:
'extension' key = ID '=' value = STRING';'
;
```

The next two items need to be considered together, together they define a notation for importing models into one another.

```
Import:
'import' importedNamespace = QualifiedNameWithWildcard
;
```

```
QualifiedNameWithWildcard:
QualifiedName '.*'?
;
```

A DataType, as discussed in section 4.3.1.3 is either an enumeration (*EnumeratedType*) or a BasicType (which can be a STRING or an INT type from our import Terminal file) or a ReferenceType

```
DataType:
PrimitiveType | ReferenceType | EnumeratedType
;
```

An EnumeratedType consists of an array of key-value pairs of integers, which can be arranged as appropriate to declare a particular type.

```
EnumeratedType:
'enumtype' 'status:' status = Status 'domainid:'
guid = GUID  '@' '(' Version ')' refines=(REFINE)
name = QualifiedName ':' type=PrimitiveType(array ?='[' (length=INT)? ']')?
;
```

A reference type will point at an existing DataClass within that model for a definition of its type.

```
ReferenceType:
'reftype' 'status:' status = Status 'domainid:'
guid = GUID  '@' '(' Version ')' refines=(REFINE)
name = QualifiedName ':' type=[DataClass]
;
```

A PrimitiveType can consist of a String or an Integer.

```
PrimitiveType:
{PrimitiveType} ('datatype' 'status:' status = Status 'domainid:'
guid = GUID  '@' '(' Version ')' refines=(REFINE)
name = QualifiedName ':' type=Basic (rule=[Constraint])?)
;
Basic:
```

```
typename=('STRING'|'INT'|'BOOLEAN')
;

BOOLEAN:
('0'|'1')
;
```

A ContainerElement is introduced as a grammar construct so that a DataClass can contain with a DataElement, another DataClass or a Reference.

```
ContainerElement:
DataClass| DataElement | Reference
;
```

A DataClass can have a degree of specialization, since this is a structural DSL it is limited to including the structure of its parent DataClass. This is expressed using the *extends* keyword.

```
DataClass:
'DataClass' 'status:' status = Status 'domainid:'
guid = GUID'@' '(' Version ')' refines=(REFINE) name = QualifiedName  '{'
(dataelements += ContainerElement)*
(extension += ExtensionItem)*
'}'
;
```

A DataElement must have a DataType associated with it, and like other data-items it also needs to identified using the identifier and version.

```
DataElement:
'DataElement' 'status:' status = Status 'domainid:'
guid = GUID  '@' '(' Version ')' refines=(REFINE)
name = QualifiedName ':' type = [DataType|QualifiedName]
;
```

A Constraint can be added to a DataModel (as in Section 4.2), whereby a constraint consists of two dataitems joined by a predicate.

```
Constraint:
{Constraint} 'constraint' name = QualifiedName '=' (( src = DataElement ':')?
pred = [Predicate]  (':' dest = DataElement)?)
;
```

Here a predicate is simply captured as a text item.

```
Predicate:
'predicate' predicate = 'string'
;
```

5.3 ECore Model

The previous specification allows us to define a basic metamodel in ECore to form the core of the DSL, this is displayed in the diagram 5.2. The diagram is difficult to read in this form, so the next 4 sections reproduce different aspects of the metamodel and explain the way in which the core requirements are implemented in the Domain Registry application. The Xtext base is flexible enough to generate a set of skeleton domain classes from which the core java application is built. This implementation however is language neutral, java is the easiest to integrate with Ecore, but the XText and Xtend(the language used by XText) is flexible enough to generate the core domain classes in a variety of application languages.

Core Domain Registry

The purpose of this DSL is to abstract, manage and transfer datasets, so any implementation will include some kind of registry, called here a *Domain Registry*. The registry registers models, in this case *DataModels*, which are subject to a publishing cycle shown in Figure 5.3

Publishing Cycle

In order to capture a dataset a draft model is developed, it is given a draft status, which means that changes can occur to it. When work on a datamodel is complete the authority can then fix that model and declare it as finalized, no changed can then be applied to that particular model. Any further changes will be to a new *draft* model which will have a new version number. For any datamodel there can only be one finalized version, if a new datamodel is given the status of *finalized* then the old datamodel acquires the status of *superseded* at the same time. A

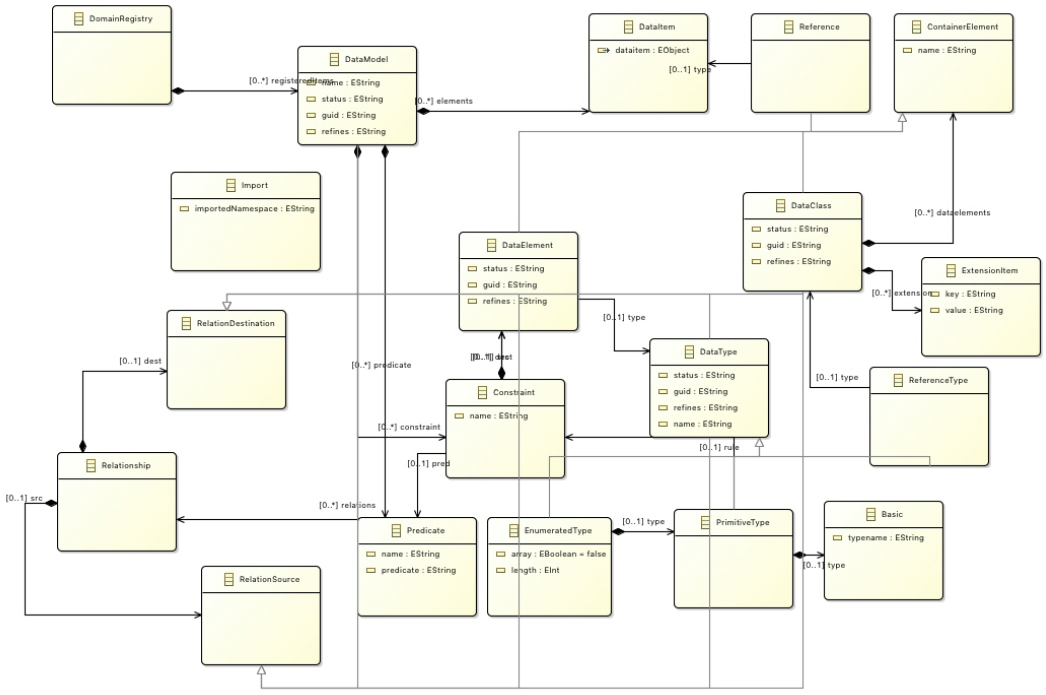

Figure 5.2: MDML Xtext-generated Ecore Metamodel

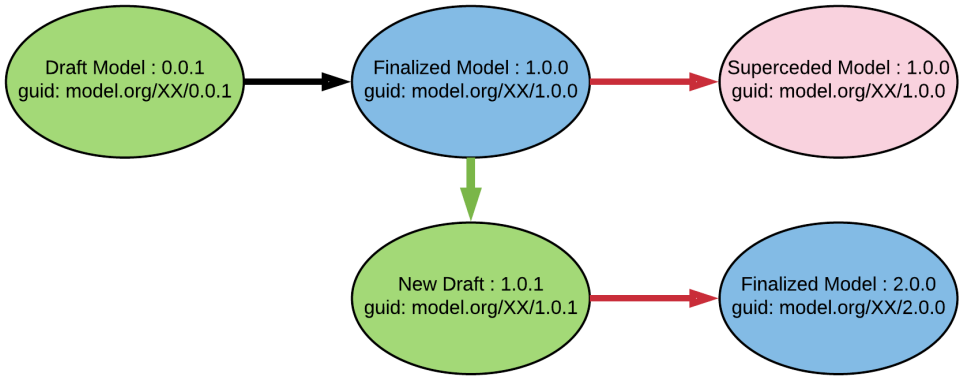

Figure 5.3: MDML Registry Publishing Cycle

superseded datamodel can also not be changed, it may have many references to it, which will still be valid and useful, and it maybe that superseded datamodels continue to be used and remain more popular that the current finalized datamodel. In essence a superseded model is still a finalised or immutable model as per the specification, the term *superceeded* is introduced to help users with the managing a registry full of a variety of similarly named models. In the Xtext grammar this is represented thus:

```
Status:
```

```
('draft'|'finalized'|'superseded')
;
```

When a datamodel is finalized all the dataitems within that datamodel are also finalized. For the purposes of the formal treatment in chapter 4, the models considered are immutable, therefore either finalised or superseded.

The versioning system adopted is based on a system common in software development, one called *semantic versioning*, as described earlier whereby each release of a model is given a 3 part version number: Major:Minor:Revision.

A Major version is one which is then fixed and *Finalized*. The data items within that model cannot be changed. All the references to other dataitems in the registry can only be to other *finalized* data items.

These rules of operation are not built into the language, however the language is built to accomodate them. In practise they are essential to the working of a registry, a hospital making a report needs to be able to reference a dataitem, and know that it is a dataitem that is not going to change during the reporting cycle.

Since **dataItems** and **datamodels** do change, the mechanism has to allow for continual change. A cycle which moves all dataitems from *draft* or mutable to *finalized* or fixed provides this ability. A datamodel once finalized, and all dataitems contined within that datamodel, remains fixed and unchanged.

Since these dataitems will be referenced by multiple parties they must have unique reference system that also will not change, every dataitem must be referenced using a globally unique identifier, such as a URI.

It makes good sense to ensure that all finalized datamodels have a major versioning number of the type *X.0.0*, but this is not essential to the functioning of the registry in the same way that using *guids* for each finalized dataitem is. In essence an item registered in a registry, a **dataItem** is a definition of that **dataItem**, this definition is then used for validation and for creating datasets, if it changes when in use then the purpose of the metadata registry is defeated. The whole point of this exercise is to avoid the confusion caused by many differing naming conventions which are used on an application by application basis. Model driven development without globally unique identifiers is possible, but applying these principles to data interoperability problems cannot be done without a global system of identifiers identifying the structure of an identified data item.

Registry Overview

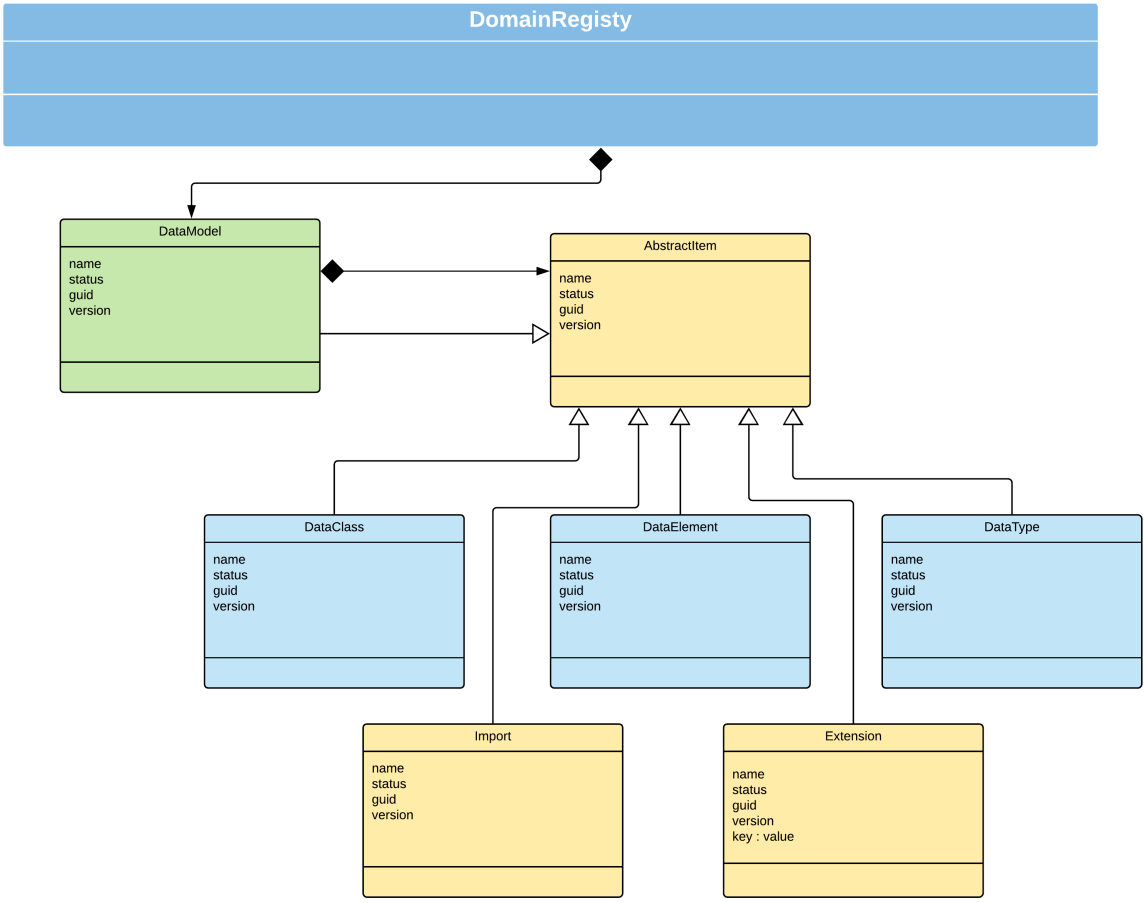

Figure 5.4: MDML Core Registry Metamodel

The core metamodel consists of a *DomainRegistry* which contains many *registered data-items*, primarily these data-items are defined as extending a core *AbstractItem*, which includes the core attributes of *name, guid, status and version.* These attributes define the data-item and are required on all items registered in the registry, this includes imports.

The import mechanism allows finalized datamodels from this registry or in a federated registry from other registries, to be loaded into a model. This feature allows for a sharing od data elements in a domain, for instance there are a lot of core data items which are used regularly in a single domain. In the UK NHS the *National Health Number* a 10-digit number, is used in almost all patient-centric applications, it makes sense for any model needing to identify a patient to import the current definition from a central authorative source, such as the NHS Data Dictionary, rather than re-define the NHS Number locally.

Data Types

DataTypes are needed both to enable existing datasets to be captured and to model new dataitems. The model follows the grammar rules defined earlier, whereby each Data Element *has-a* DataType, which can be a *Primitive DataType* or a *Reference DataType* or it can be an *Enumeration*. Primitive DataTypes, in this implementation, consist of Strings, Integers or Booleans, however it is possible to use a wider set of primitive types if needed.

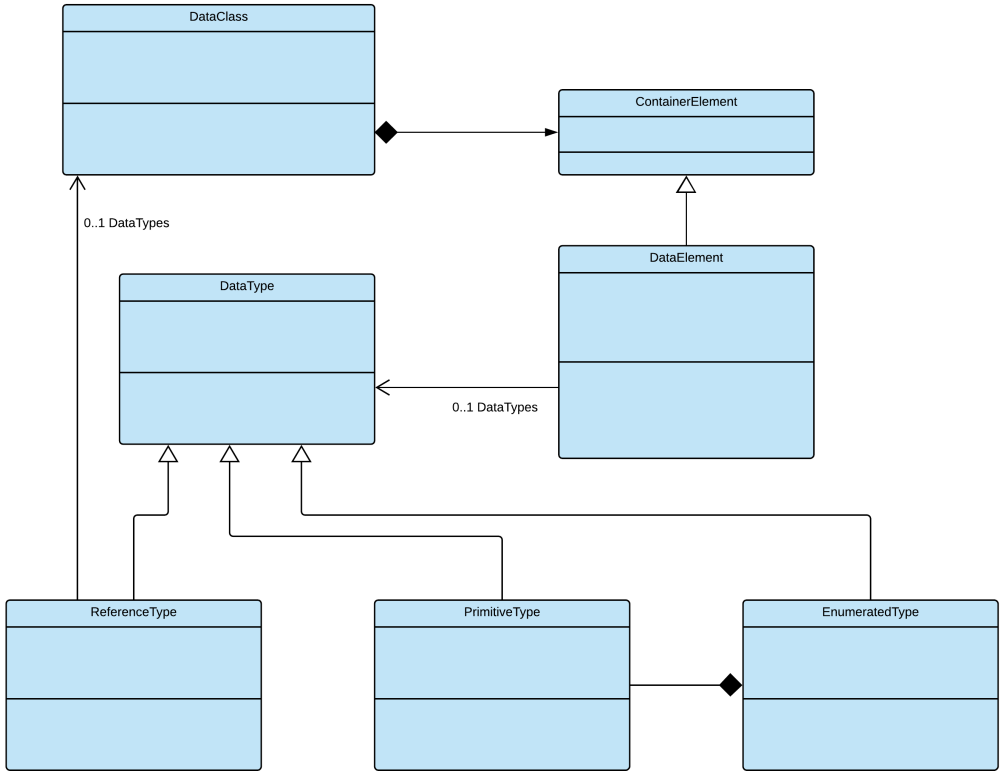

Figure 5.5: MDML Data Types

A DataClass can contain many DataElements (via the ContainerElement, a class which is used purely to capture the containment capability), and DataElements in turn are related to one or zero DataTypes.

A DataType has 3 sub-types, a PrimitiveType, a ReferenceType and an Enumeration. The PrimitiveType is the basic type, and a collection of PrimitiveTypes can be used to create an Enumeration or EnumeraredType. These are key in being able to capture enumerations, which are very often used to capture results of form-based research questionaires.

The ReferenceType exists as a mechanism to turn an existing DataClass into a DataType, this enables DataElements to have the type of an existing DataClass.

Relationships

Relationships need to be extendible, the idea is to capture any possible relationships in any object oriented, or relation database application, and therefore a mechanism to capture different relationships is defined.

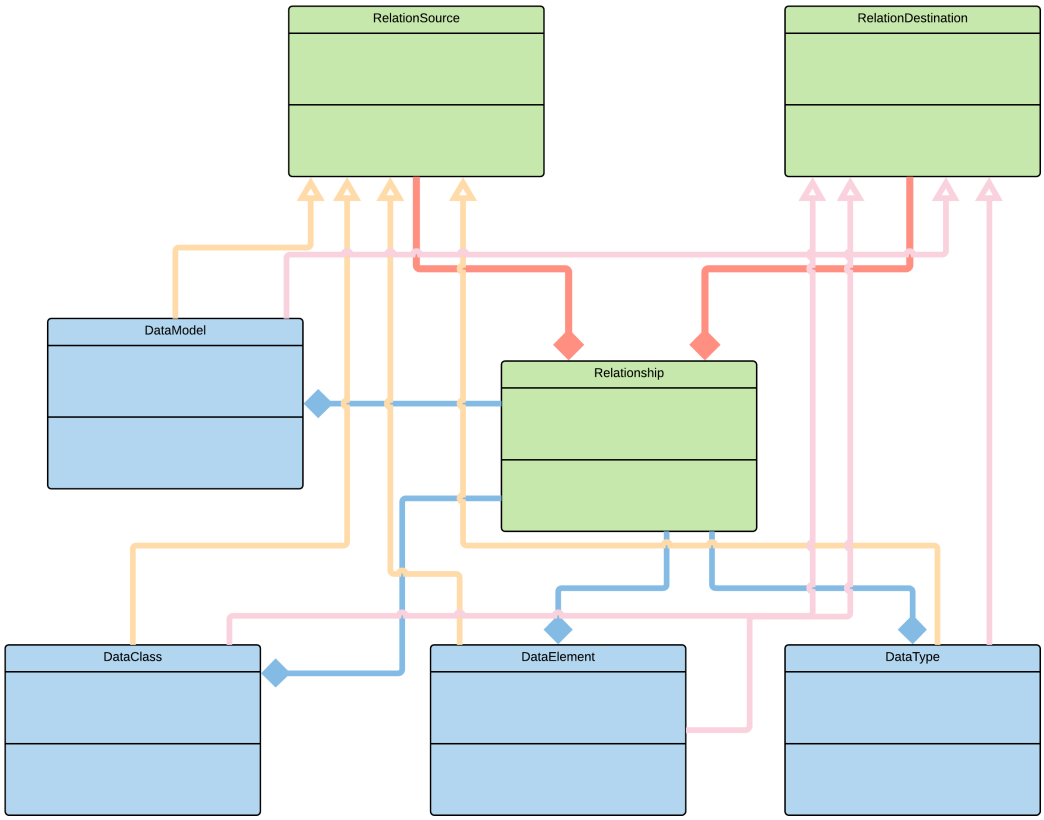

Figure 5.6: MDML Relationships

Any core DataItem can be related to any other code dataitem within the registry, the mechanism to do so is by defining a one way relationship, the relationship itself can have an extension added to it, and so the relationship can carry metadata. The Relationship has 2 ends, a RelationSource and a RelationDestination, for two-way relationships an inverse relationship can be defined using the same mechanism. Each dataitem used in a relationship inherits from both RelationSource and RelationDestination, and thus can be used as part of the Relationship.

This mechanism allows relationships such as *derived from* or *based on* to be defined; for instance a DataType for *subjectCode* might need to be composed of an organisation code,

perhaps representing a hospital and an NHS Number. By using a registry-defined *based on* relationship between two DataTypes a new DataType called *subjectCode* could be devised adding perhaps a 3-digit organization code to a previously defined NHSNumber DataType. This allows DataTypes and other dataitems to be built up using existing definition.

Constraints

Constraints as mentioned earlier are not part of the language, although it would be possible to define an expression language in XText which could be used within MDML to express constraints. Constraints are DataModel-wide,

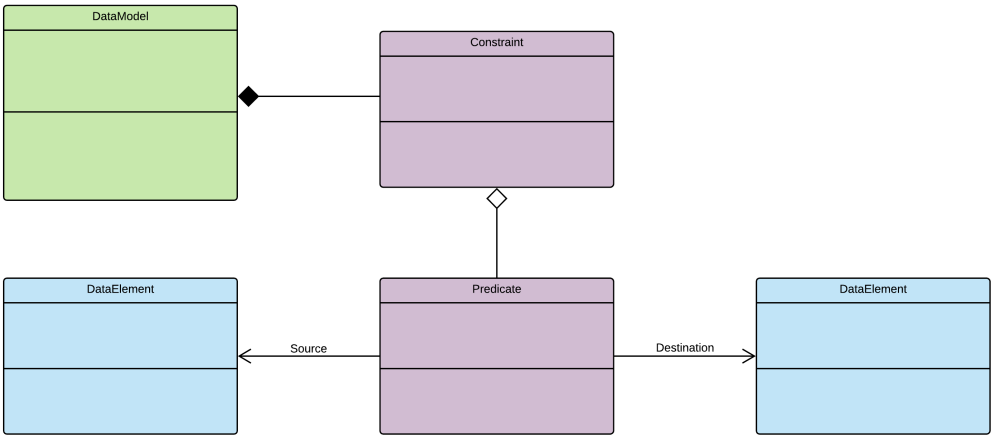

Figure 5.7: MDML Constraints

5.4 Conclusion

The work of this chapter provides a formal grammar which can be used as a basis for a software system, the core grammar can be translated directly to provide java skeleton code, and this in trun has been used to provide a metadata registry which directly conforms to the grammar. This work in this chapter is inspired by the specification work carried out in chapter 4 and is directly related to it. Implementing a first order logic is not trivial, but using a formal EBNF-based grammar takes much of the uncertainty away, that said we haven't been able to prove that MDML is fully conformant to the specification in chapter 4.

Chapter 6

MDML Evaluation

6.1 Overview

In this chapter we show how the MDML metamodel improves on the existing ISO11179 metamodel by solving the key problems that were identified earlier on in chapter 3. We examine how the same datasets that were examined chapter 3 are modelled using MDML and provide an evaluation of the improvements to the metamodel provided by MDML.

Some work had already gone into applying the ideas in ISO/IEC 11179, the ISO standard for metadata registries, and so this was adopted as a way forward in the research program. We built and tested the interchange of several datasets using an ISO11179 compliant metadata registry with mixed results, we examined the problems which arose, and then built a revised metadata registry built on model driven engineering principles. For the most part we are concerning ourselves with structural metadata, information that describes the structure of the data rather than descriptive metadata, which can consist of any labels or metadata that references a data element. We touch on the differences and what they mean for a metadata registry later on in the chapter.

6.2 Key Problems

In chapter 3 we identified 3 key problems associated with applying the ISO11179 to healthcare datasets:

- Inability to classify and group data elements
- Lack of consistency and thus repeatable results

- An ambiguous set of artefacts in the metamodel systems

These key problems are solved by using the MDML language as a base for the registry development. MDML has in turn been developed in parallel with the formal specification work carried out in chapter 4, and is largely informed by this work. It can be observed that if every dataset is fully normalised it is likely that some of these problems may be lessened, however in our view standard for metadata needs flexibility to cover other cases besides fully normalised relational databases, and this is what we are adding in through this new formal metamodel.

Classification

The MDML metamodel is based around data elements, and in this regard can be thought of as in keeping with ISO11179 . It differs in that it also has a hierarchical, containing structure called a **DataClass**, which behaves very much as a Class behaves in UML. This provides a simple, intuitive mechanism to map hierarchies of data elements to. There is a further mechanism of **Tags** which allows all elements derived from the **DataItem** to be classified, allowing multiple alternative hierarchies to be managed in parallel, within the same registry.

Consistency

The lack of consistency in the ISO metamodel is a result of the conceptual aspects, which are open to interpretation when building an implementation. Unlike OWL for instance, it lacks any formal specification which can be referred back to resolve ambiguity between the text and UML models contained in the specification documents. With MDML there is a clear formal specification at the core of the metamodel, which has been translated into a formal grammar before generating the core domain model in code. This allows Class files, DDL files, XML Schema and JSON Schema files to be generated directly.

Ambiguity

The MDML metamodel is closely aligned to UML, and simple intuitive mappings exist between UML, XSD and OWL, which have been detailed in the previous section. Thus we are able to eliminate the ambiguity associated with the initial implementation of the ISO11179 metamodel.

6.3 Research Findings

This research started out to discover better ways of integrating datasets for medical research and clinical interoperability. In particular we are interested to see if automated techniques using metadata could be applied to managing, wrangling and cleaning data used in clinical data analytics.

In the first few weeks of trying to enter standard existing healthcare datasets into the prototype metadata registry many objections were encountered from users experience with existing healthcare datasets, of the kind documented in chapter 3. Metadata was seen to be entered twice or three times needlessly, the difference between the description of a *Data Element* and a *Data Element Concept* was not understood. Likewise the difference between a *Value Domain* and a *Data Type* whilst apparent in theory, was in practise not apparent, since the models being generated were not implementation specific. Therefore a representation of a set of numerical values would in nearly all cases be represented by the same data type, for instance the *date of clinical assessment* would have a value domain of *date* and a data type of *date*, which would then be implemented in a particular system as appropriate, e.g. text string, org.joda.time.format.DateTimeFormat as appropriate by the system concerned.

During the course of implementation of the fully conformant ISO11179 registry we upgraded the structure of the metamodel in order to ease the implementation process. After a few weeks it was decided to update the metamodel, at first, the number of ISO11179 elements was reduced, however this still did not gain any traction with users, who found the system confusing, non-intuituve, time-consuming and needing a lot of extra work to understand the new language constructs introduced by the standard. As a result the development went through a two iterations to arrive at the current model, the initial prototype we refer to as version 0.x, the second as 1.x and the third as 2.x. There is only one minor change from 2 to 3, the main change is from 1 to 2. After the introduction of 1.x and subsequent problems,it was decided to write a formal specification of the relevant parts of ISO11179, which resulted in the development of the Z specification for the metadata registry and the MDML metamodel.

The third iteration has been fairly successful and is currently being used in 2 implementations at over 35 hospital trusts and healthcare research centres in the UK. The changes over time in core constructs are shown in Table 6.1.

The new metamodels leave out the conceptual constructs, these have been dropped from the iteration 2 and 3. Firstly they cannot easily be inferred from an in-place dataset and thus slow down any automation taking place, secondly in practise they were omitted. In the case

of Value Domain, in practise the information entered into this field was identical to the Data Type, therefore it was not adding anything to the expresiveness of the metamodel. The Object Class and Property artefact are dropped, mostly because they were hardly ever completed, it is possible to capture hierarchies of data using these two artefact, however it is not only unintuitive, but the rules are almost impossible to automate.

The key improvement in expressivity comes from the addition of a DataClass, which can be inherited, allowing hierarchical collections of data to be modelled directly in the metamodel. The other key imporvement in expressivity comes with adding RelationshipMetadata and RelationshipType to the metamodel, this allows a wide variety of relationships to be captured and managed easily. By specifying the type of relationship rules can easily be added into the automation process to allow data in two separate columns to be identified by a type of relationship, rather than having all relations lumped together as with the previous metamodel. In addition artefacts have been added to capture Assets, AssetFiles and Extension Values. The later allows free form descriptive metadata to be added on an ad-hoc basis, so that data models can have descriptive metadata sets which can then conform to standards such as DCAT, which is specified using RDF. Automating capture of such metadata in the original metamodel is a laborious process, in MDML it is simply a matter of configuration.

In chapter 3 we illustrated the problems associated with importing 3 dataset standards into the ISO11179-based metadata registry. In this section we demonstrate the same mappings using the MDML metamodel, and show how the revised simpler model allows for a less ambigous mapping process. This in turn allows for semi-automated mappings to be programmed for large datasets which would not be possible using the original ISO11179-3 metamodel. For an ISO11179 registry an expert is required to load in each dataset, and to make decisions about each item; in contrast with MDML once the core mappings are decided no further consultation is needed, the data items themselves are loaded according to the mapping.

Table 6.1: Metamodel Domain Constructs

ISO Artefact (v.0.x)	**Iteration 1 (v.1.x)**	**Iteration 2 (v.2.x)**
Described Conceptual Domain	-	-
Object Class	Model	DataClass
Property	-	-
Data Element Concept	-	-
Data Element	DataElement	DataElement
DataType	DataType	DataType
Value Domain	Value Domain	-
Described Value Domain	-	-
Enumerated Value Domain	EnumeratedType	EnumeratedType
Permissible Value	Enum	Enum
Enumerated Conceptual Domain	-	-
Relations	Relationship	Relationship
-	RelationshipMetadata	RelationshipMetadata
-	RelationshipType	RelationshipType
Classification	Classifier	Tag
Concept System	-	-
Concept	-	-
Classifiable Item	CatalogueElement	CatalogueElement
Measurement Unit	MeasurementUnit	MeasurementUnit
Measure Class	-	-
Dimensionality	-	-
-	Asset	Asset
-	AssetFile	AssetFile
-	ExtensionValue	ExtensionValue

COSD

COSD is grouped using tabs, and sections, both of which can be eaily converted to top-level DataClass's, nesting within a single DataModel. The Data Item id, name and description can all be mapped into a DataElement, allowing the format to be represented as a rule, translated into a regular expression, within a DataType. The DataType in turn will be an enumeration, which can easily be mapped to the National Code definitions. The Data Dictionary link, Other Collections and Schema Specification can all be inserted as relationships to these models, which are held separately in the registry.

Table 6.2: ISO11179 - COSD - ISO11179 Mappings

COSD Artefact	**MDML Artefact**	**Notes**
Spreadsheet Name	Data Model	mapping
Spreadsheet Tab Name	Data Class	mapping
Data Item No	Data Element (id)	mapping
Data Item Section	Data Class	mapping
Data Item Name	Data Element (name)	mapping
Data Item Description	Data Element (description)	mapping
Format	Data Type (Rule)	mapping
National Code	Enum (id)	mapping
National Code Definition	Enum (name)	mapping
Data Dictionary Element	Relationship	mapping
Other Collections	Relationship	mapping
Schema Specification	Relationship	mapping

The core MDML metamodel is used in its entirety for representing this dataset, there are some artefacts such as extension values which are not immediately needed, however for the core artefacts there is 100% coverage.

SNOMED CT

SNOMED CT is formulated as an OWL model, and thus the metamodel can be used as a means of transforming the elements, although we have done this here in the mapping shown in Table 6.3. In this case concepts have been mapped to DataClasses, since there is a clear concept hierarchy within the SNOMED design. DataElement are able to represent terms, although to

mark the *preferred term* it is necessary to add in an extension key-value pair on the appropriate DataElement. Since SNOMED is a conceptual model there is no need for DataTypes, although Tabs are needed to manage *Refsets* which are pre-grouped groups of concepts.

Table 6.3: SNOMED CT - ISO11179 Mappings

SNOMED Construct	**ISO Construct (v.0.1)**	**Notes**
Concept	Data Class	mapping
Term	Data Element	mapping
Relationship	Relationship	mapping
StatedRelationship	Relationship	mapping
Description	Data Element (Description)	mapping

NHS Data Dictionary

The NHS Data Dictionary is partially described using UML, and so the mapping to MDML is strightforward, as is illustrated in Table 6.4. All the artefacts present in the NHS Data Dictionary can be easily incorporated within the MDML model, although Relationships and Collaberations may require more examination. For the most part there are direct mappings between dirrerent artefacts, as mentioned in the case of a unl collaberation the mapping is indirect, as illustrated in table 6.4. By direct mappings we mean that the artefact can be moved from one model to the other, for indirect mapping there may be some aspects of the artefact which are lost on transforming it.

For all three reference datasets the coverage when represented in an MDML-based metadata registry is complete, there are a few items such as the Preferred Term for SNOMED concepts which require extra extension values, however the transformations are relatively straightforward and can be carried out by technically-minded clinicans with ease.

6.4 MDML: A DSL for Metadata Management

The XText domain model provides a grammar which is then used to generate a set of domain objects, these in turn are able to represent the various healthcare datasets being used. The core classes enabled the automatic generation of grails domain classes using the code generation capabilities of XText. For illustrative purposes we show the key metadata registry structure

Table 6.4: NHSDD - ISO11179 Mappings

NHSDD construct	**ISO Construct (v.0.1)**	**Notes**
uml:Class	DataClass	direct mapping
uml:Association	Relationship	direct mapping
uml:Package	DataModel	direct mapping
uml:Collaboration	Relationship	indirect mapping
uml:LiteralInteger	DataType (Int)	direct mapping
uml:LiteralUnlimitedNatural	DataType (Int)	direct mapping
uml:Attribute	Data Element	direct mapping
uml:DataType	Data Type	direct mapping
uml:Enumeration	Enum	direct mapping

in Figure 6.1. In addition a metamodel was generated using ECore, which has 20 classes in it. For illustrative purposes we show the key metadata-structure in Figure 6.1

The structure illustrated here allows the metadata registry to capture data elements, which is at the core of the standard. It also allows each element to have a status, which allows it to conform to the publishing aspects of the standard. By making each item in the registry a class inheriting from an *abstract Catalogue Element* each data element is captured by a unique identifier and a status relating to its publishing status.

In the ISO standard metamodel the data element is the primary artefact under consideration, however there is no obvious direction on how groups of data elements should be handled.

In practise data elements appear in groups, whether spreadsheets, database tables or CSV files. In the data analysis business, groups of data need to be brought together and to do this datasets need to be mapped and transformed. In our first and second models it was easy to generated lists of data elements, however they were nearly always grouped using several layers of hierarchy into different entities, which made mapping and transformation difficult.

Further a DataModel is able to add in any number of extensions, which can be used to augment particular data elements for instance to mark them as user interface items so that user interfaces can be automatically generated. In addition imports can be added from other DataModels, so that new datasets can be created using data elements from several different datasets. This metamodel evolved over a period time in order to solve the problems of integrat-

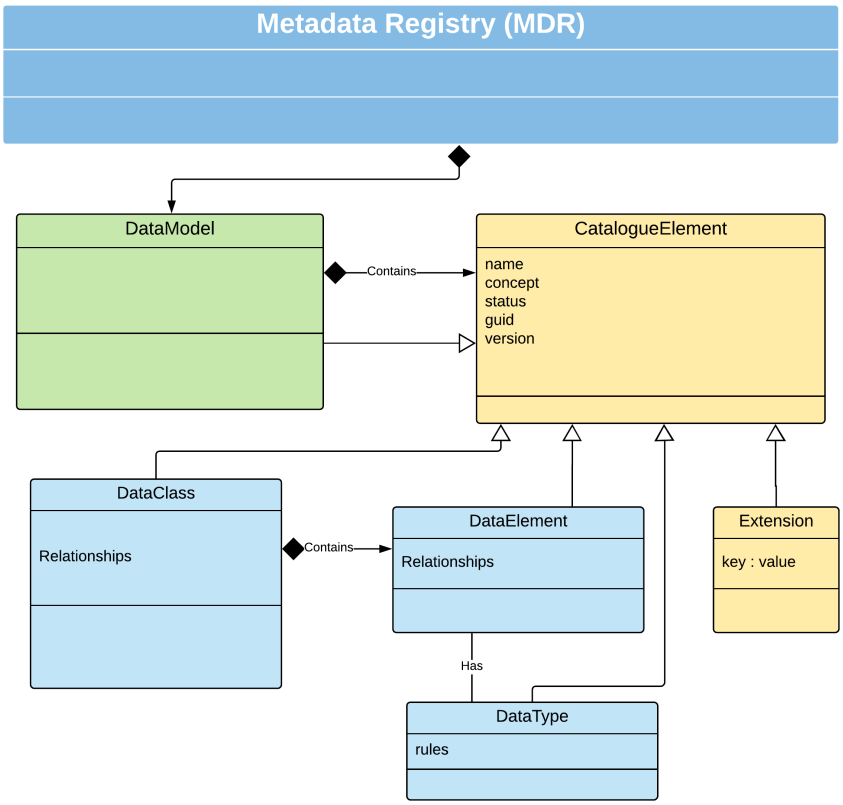

Figure 6.1: Core MDR (Ecore) Metamodel

ing data from a wide variety of heterogeneous sources, it is closely based around UML/Ecore, but is in essence a domain specific language or metamodel for data integration.

The main constructs of DataClasses, DataElements and DataTypes are common to most technical spaces, and can be used to group and manage data elements in way that is easy for data analysts and developers to grasp. Metadata can then be attached to any of these core elements, so that security classifications, and governance directives, found in other parts of the ISO11179 standard can easily be added to particular DataElements or DataClasses.

AT core MDML provides a metamodel, in essence a language for data structures, which allows additional metadata to be added on through a simple mechanism of extension values and tags. It allows datasets to be grouped and classified in a flexible and manageable way, so that one registry could hold several common healthcare metamodels, say FHIR, OMOP and LOINC, and build local models based on all three. There is a clear identification system built into the language which means that each version is separately identified, so that any mappings between FHIR and OMPOP for instance will be detailed to particular versions of the datasets.

There is a clear publishing cycle built into the metamodel, which allows change for *draft* models, but not for *finalized* models, therefore data elements in finalized models can be imported into other (draft) DataModels, but no new elements can be imported in currently finalized models. This is an important management feature, derived from the ISO standard which has been built into the language.

It is relatively simple to transform these datasets into the MDML format, as illustrated in Figure 6.2, which shows how a data class in the UK NHS Data Dictionary called *Diagnostic Test Request* is represented in MDML as a text-based serialisation.

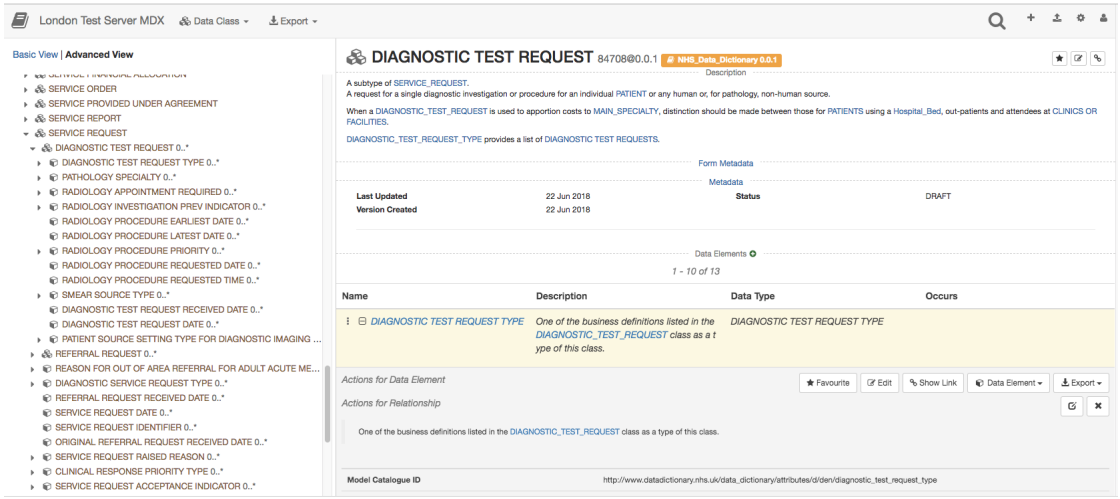

Figure 6.2: MDR UI showing Diagnostic Test Result

6.5 Discussion

Industrial development of interoperability technologies is proceeding at a rapid pace, and middleware toolkits based around the Service Oriented Architecture concept are being developed and refined, however, they are bogged down in syntactic complexity – something that many software consultancies and middle-ware developers have little incentive to reduce. As the amount of data at play in the world increases there is a huge demand for heterogeneous software systems to interoperate automatically or at least with minimal configuration effort.

Metadata

Metadata is a rather vague term, it is often defined as *data about data*, however this definition is rather too fuzzy for our purposes. It is a term which used in data management, library science,

healthcare data management, museum curation but also in telecommunications, engineering and computer science as well as being often referred to in the media and entertainment. As a term the meaning changes with the context. The Daily Telegraph in Australia identifies the importance of metadata in crime prevention [3],as have many other newspapers and publications. The Guardian in the UK has made many references to metadata in the past few years [35], and in 2013, to throw light on "email metadata", they published a quote from Julian Sanchez of the Cato Institute:

> "The calls you make can reveal a lot, but now that so much of our lives are mediated by the internet, your IP [internet protocol] logs are really a real-time map of your brain: what are you reading about, what are you curious about, what personal ad are you responding to (with a dedicated email linked to that specific ad), what online discussions are you participating in, and how often?"

Metadata has been used by librarians and museum curators for hundreds, if not thousands of years, Richard Gartner devotes a chapter of his book on Metadata [26] to historical usages of metadata, starting with the earliest known library in Ebla (Syria) dataing from 2500 BCE. A collection of tablets was positioned so that its first line was left visible when the tablet was shelved, in addition a number of classification lists were also found allowing the system to be accessed using a hierarchival system. In this book Gartner emphasises the fact that metadata is a human construct, or filter on data which is constructed dependent on the context and purpose behind its acquistition. For instance a map can be viewed as metadata, it is data about a journey, data that a human needs to effectively make a journey, in the case of a car journey the map will show roads, possibly some building and features such as bridges, whereas in the case of a metro journey the map will simply show stations and different railway lines will show up in different colours. The data for a city such as London remains the same in both the case of a roadmap and an underground map, however the two may not be well synchronized in the case of someone making a journey involving both road and underground journeys. Two stations on the underground may appear far apart and in fact be a few hundred yards distant from each other.

Metadata can be thought of as a linguistic construct, by which we mean that it can be considered a set of signs which acquire meaning according to the context. Saussure, in a General Course in Linguistics [16], identified a sign as a 2 part construct, made up of the signified (or concept), and the signifier (or sound-image), as illustrated in Figure 6.3, which is reproduced from the original textbook.

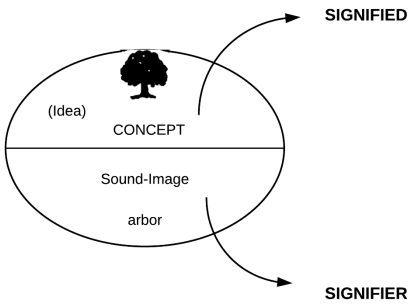

Figure 6.3: Signified and Signifier

Why is this of interest? Because there are other metadata standards, such as DCAT [88], which handle descriptive and administrative metadata, a screenshot is shown in Figure 6.4. Such standards can easily sit alongside MDML **DataModels** as the extension mechanism allows key value pairs to build up metadata sets which conform to these descriptive metadata standards.

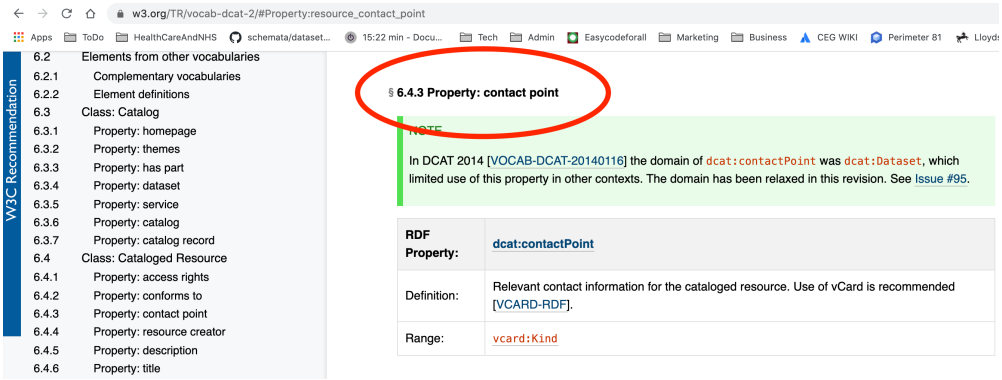

Figure 6.4: DCAT 2 - Data Catalog Vocabulary

DCAT is a data catalog standard, authored by the W3C using RDF, available for usage by organisations engaged in building data catalogues. It is open and being available as RDF can be accessed and managed by machines directly. This means that if it is incorporated into an MDML model, the software is able to access the original specification and validate the metadata against that standard.

This allows not only key aspects of the structure to be captured in legacy systems, and managed automatically, but the structural metadata can be linked to both administrative and descriptive metadata in a flexible manner as shown in screenshot 6.5. This means that data can be viewed, aligned and managed separately from the applications that it resides in, for researchers it allows disparate datasets to be described before being accessed, it means

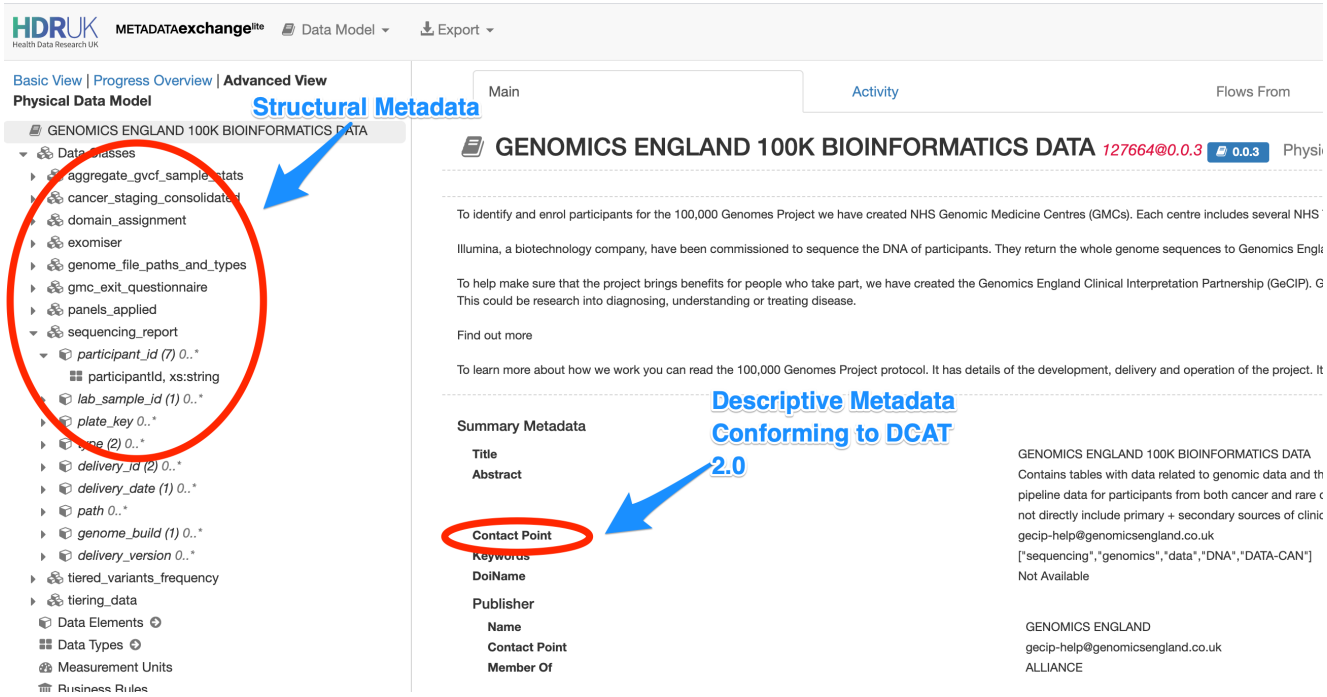

Figure 6.5: Incorporation of DCAT within MDML-based Registry

measures of provenance and quality can be added to the core technical structure and researchers are able to access whether a dataset is worthy of being added into a survey or project, without neccessarily accessing the data itself. This in turn means that the semantics of that data become visible and therefore can be managed automatically.

Implementation of MDML in the NHS

The MDML based metadata registry has been in use for the past 4 years at Genomics England, as well as at 7 other NHS Hospital Trusts in London. Its core use is to manage datasets which are used by Genomics England to specify dataset requirements, a dataset or model is defined in the metadata registry, this is then used to automatically generate output specifications as excel, XML or as Case Report Forms. The latter is a specification for generating forms in the Open Clinica system, but which can automatically be loaded into Open Clinica to generate both the forms and data repository to store the data. Rules, embedded into the DataTypes as regular expressions, or as groovy code, are then used to validate datasets.

By using the metadata registry Genomics England was able to specify exactly what data they required from UK Hospital Trusts by building their own data models centrally online. The other organizations are then able to use these models to verify that outbound data is valid, and thus ensuring quality of data input.

The main technique used is to import a new dataset into the metadata registry, normally in csv or excel format, to configure a mapping using XML, and then import the dataset definition into the metadata registry directly. From this new model a number of artefacts such as spreadsheets and XML Schema files can be generated from one source. This eliminated any doubt arising from spreadsheets which were disseminated after meetings or discussions. Spreadsheets and XML files could then be generated for local form or code generation, and rules could be attached to data elements and data types which could be automatically downloaded via a REST interface, rather than by disseminating spreadsheets and documents.

Data curation, which previously was carried out using excel spreadsheets, and involved regular physical meetings between data managers and clinical leads from all over the UK now takes place online. The models which are generated are given a unique identifier, and once all parties agree to issue a new model it is *finalized* thus preventing further change to that version of the model. At any point in time the model, and its history can be referred to online, removing doubt and ambiguity over what the constituents are, further the model can be accessed using a REST interface, making it available for code generation utilities remotely. The results of using the metadata registry and toolkit are listed below:

- Reduction in time and effort in creating and curating datasets over spreadsheets and document-based techniques.
- Ability to clearly access the current version of a dataset and verify that it is the correct version, using unique identifiers.
- Ability to generate artefacts based on the DataModel, such as forms, xml, XML Schemas automatically
- Ability to validate data against the model automatically using rules stored against individual data elements
- Ability to map between different datasets in a detailed and precise fashion, again with the use of unique identifiers.

In practise the fact that datasets can now be automatically mapped and modeled is a big advantage over the typical ISO 11179 conformant registry. Processes can now be semi-automated, and semantic matching between different datasets becomes a possibility. This capability can be used by linked datasets, and allows researchers to access data which may otherwise have been missed.

MDML allows researchers to build a metadata registry able to correctly capture the structure of the technical metadata in systems which are holding the data. It does not attempt to interprete the data while it is being ingested, rather it is able to infer some semantic structure from the way in which the data have been structured after the metadata has be input into the system. This means that the system is recording key aspects of the data structure, that are associated with the *semantics* of the data.

MDML is a domain specific language which has been used effectively to reduce expert involvement in tedious data wrangling tasks. We have not been able to measure the full effect of this during the study as it has been an iterative evolutionary process. Also the amount of data being examined now is well in excess of the amount that a human expert could be expected to validate, so it is safe to say that by using manual techniques, data validation would not have been carried out on the core dataset. Instead it would have been carried out by researchers on their local copy while carrying out specific analysis tasks on an ad-hoc basis. This would have meant that each researcher would have been carrying out the same validation in parallel, on their local subset of data. Such a situation would be much less efficient than maintaining a high quality data repository that can be accessed without having to carry out data wrangling tasks locally. Using a model driven engineering paradigm in data-wrangling has provided an effective implementation of the main goals identified in ISO11179, using an updated meta-model and applying a simplified version and identification protocol for data-sets. This gives the ability to identify complex data patterns as models and identify them with a globally unique identifier. In turn, this enables rules associated with data elements, or with relationships between data elements, to be applied to data-sets using automated techniques, for validation, transformation and linking. In doing this the degree of complexity is reduced, the data becomes easier to manage, and the human intervention in the process is significantly reduced. This research effort, carried out with Genomics England, resulted in the use of a model-driven metadata registry that has significantly reduced tedious 'data-wrangling' work. In addition the generation of simple low code software applications has been simplified. Applications such as web-based questionaires, excel, word and xml reports have been automatically generated due to the application of model driven engineering techniques. In summary the application of model-driven engineering techniques to the basic principles outlined in the ISO11179 standard has led to a significant increase in the amount of automation possible in the data wrangling, data validation and data management process, this in turn has lead to the toolkits adoption throughout 35 UK datahubs.

One can argue that there is less *semantic* metadata using this MDML metamodel over the ISO11179-3 metamodel, and it is true that there is less semantic context available from automatically capturing structural metadata from existing systems as opposed to talking to an expert in those systems. However we argue that unless each ISO11179-3 registry is set up by the same experts, using the same rules and interpretations to classify the datasets being registered on each registry, then there will be no semantic interoperability between different registries. Given the ever increasing amounts of data being generated there is no chance that such a system is feasible. Using MDML the semantics and context that is available from the internal structure and design of the systems holding the data is captured, in uniform way, by any MDML conformant registry, ensuring a basic degree of semantic interoperability.

Chapter 7

Mapping and Transformation of Datasets

This chapter shows how the MDML metamodel can be applied to matching datasets by applying it to some standard healthcare datasets.

7.1 Overview

One of the big problems identified earlier in this Thesis is mapping different models. This applies not only to user-created models, but also conceptual codesets standards which are very often built into software applications. For instance one application may use LOINC codes in its data elements, another one may reference SNOMED, this can be reflected in the models which are derived from that software. A researcher may build up a set of queries using LOINC codes, and then find, on being presented with a new system using SNOMED codes, that she has to change all the queries. The problem is that she may not have been provided with a map that can be applied.

Most codesets are built around the idea that *"if I have a concept, for instance* diabetes *and I designate a code for it, and every piece of software with data relating to diabetes uses that code, then I will achieve semantic interoperability between all the software systems using that code"*. The problem, as we have already documented, is that not everybody uses the same codeset, and codesets themself change regularly.

In addition we humans use natural language to communicate, and natural language is not a single formal language, but made up of many languages and dialects which vary in the way they handle concepts, time, grammar and rules. Most natural languages omit much of their

content and so rules are implicit, and very often even people who speak a particular language find it hard to vocalise the rules they use on a day to day basis.

Matching one data element X in model A with another data element Y in model B poses a number of problems. Users are trying to see if X is equivalent to Y, researchers today will want to see if there are studies done 10 or 20 years ago relating to their focus of interest, but possibly using different terms. A metadata registry built around MDML will enable the X and Y to be uniquely identified and referenceable, it also means that relationships can be taken into account when measuring if X is the same as Y. Matching using standard search techniques is a multi-stage process which is based around comparing 2 strings. Various techniques are generally used to measure how good a search is, and whilst they tend involve a lot of mathematical techniques, at core the ultimate measurement is empirical. This principle is the same when investigating if datasets matching can be effectively carried out using MDML, we can in fact borrow many of the techniques used in natural language processing and information retrieval to match data elements, however the ultimate match is the one that an expert in the field would make.

This chapter looks at matching codesets using MDML. Initially we considered using MDML itself as a basis for the mapping, in that it it is possible to built some logical functions into the language to make these mappings, however there was little obvious benefit to do so as it means starting from scratch. Many aspects of the *matching* problem have been researched in a variety of different fields and contexts. Instead we examined existing information retrieval techniques to see how these can be adapted and applied into the area of dataset mapping, in particular we looked closely at ontology matching techniques and worked on adapting these to the problem at hand.

7.2 Matching

MDML is a means of capturing metadata about exising heterogeneous datasets, and the metadata registry built around it is capable of handling many different datasets in a unified way. At core these are models of particular domains, they normally contain a vocabulary as well as some structural features, and they can be expressed as database schemas, folksonomies, XML Schemas, JSON Schemas, UML models or formal (OWL) ontologies. Arguably these can be placed in order of formality, however there are a number of aspects which are used to structure the different models and thus help with the matching process.

Tagging and labelling of data is used in some websites to classify and group different items and allow them to be searched. Tagging and labelling can be used in the search process

to identify items and add them to collections, however the lack for formal reference for the vocabulary used can hinder this process. Web search engines use statistical techniques to try and get around the absence of any formal means of identifying the meaning of any particular tag or label.

A directory or tree structure is a very common way of classifying documents in computer systems, and this can be used to an extent in the matching process. Tree's appear when files are put in a structure and when classes are grouped into a hierarchy. This kind of hierarchical classification is behind most library and archival classification systems and it needs to be identified and measured as a factor in any metadata matching process.

XML is normally defined by XML Schemas, which identify how complex and sinple elements are related and structured. They can describe hierarchical classification using nesting, they can also group elements, relate them and provide a basic scheme for data types which broadly matches most other data typing schemas at play.

Relational databases define fields and their relations, which are grouped into tables. In addition rules can be embedded into the database as stored procedures and triggers to ensure that business rules are enforced. Fields are typed, and some columns are linked to other tables as designated foreign keys, this enables referential integrity constraints to be enforced. Relational models are limited in that they cannot express taxonomies directly, as can object databases. To express hierarchy in a relational structure one needs to add in link tables, that in turn need some means of defining the hierarchy, for instance a numerical level field.

Some medical datasets are defined using formal ontologies, and these can be captured using the MDML metadata registry, they are more expressive than simple relational models and use classes, relations and data types in a similar way to MDML. In addition they defined a number of different kinds of relations, such as specialisation, exclusion, instantiation. When two formal ontologies are being matched there is a greater degree of granuality required, since ontologies are able to capture meaning in a more formal way than these other modelling systems. For the most part we are not concentrating on ontology matching in this study, since it is a subject which has been thoroughly investigated. The focus of this research is heterogeneous matching betweeen different medical *datasets or models*.

There are 2 main components to matching, firstly solving the similarity of naming problem, and secondly solving the structural and granularity problem. These problems are very similar, and overlap with the matching problems encountered when trying to match up instance data when integrating or merging databases. They are illustrated in Figure 7.1, where we see the

patient records as they appear in the GP surgery database, and then how they are represented in the hospital clinic's system.

GP Surgery: Patient Details

id	nhs_id	dob	age	sex	address	town	county	postcode	first name	last name
1	333-122-1222	08-02-55	65	2	87, Rochester lane	Lincoln	Lincs	LN1 8HU	Jane	Edwards
2	232-090-2333	23-11-98	22	1	The Smithy, Lancet Lane	St.Giles	Lincs	LN1 3JO	Richard	Smith
3	487-343-9092	01-01-04	16	2	43, Redway Court	Lincoln	Lincs	LN3 6HU	Lucy	Whittington

Local Hospital : Clinic Attendance

ID	Name	BirthDate	AddressID	gender	AttendanceDate
23443	Lucy Whittingdon	2004/Jan/1	112	f	2018/11/03
32999	Jane Edwards	1955/Feb/2	113	f	2018/11/03
43223	Richard Smith	1998/Nov/23	114	m	2018/11/03

Local Hospital : Address Details

aid	address1	address2	town	county	postcode
112	43	Redway Court	Lincoln	Lincs	LN3 6HU
113	87	Rochester Lane	Lincoln	Lincs	LN1 8HU
114	The Smithy	Lancet Lane	St.Giles	Lincs	LN3 6HU

Figure 7.1: Diversity of Information Storage

This example demonstrates how the same semantic information is stored in heterogeneous systems. We have a set of 3 records of patient visits to a hospital clinic on recommendation from their local GP's surgery, at present we are simply looking at the data about the patient, at the GP end it is stored in a table called *Patient Details* and at the hospital end the same information is stored in 2 tables called *Clinic Attendence* and *Address Details*.

An automated process of mapping datasets is in effect trying to map the database headings, and here we will have to map the headings for one table with the headings of 2 tables. The following listing illustrates the type of problems associated with matching raw instance data, most of which can be solved or reduced by the use of MDML

- there is no obvious connection via the table names, which are diverse

- there is no obvious way we can connect the Clinic Attendence with Address Details unless the AddressID header has been linked to the Address Details table and aid column as a foreign key.
- there is no obvious way to connect data in the Patient Details table with the Clinic attendence table, the id's are different and the name is represented in 2 columns in one table and in 1 column in the other table.
- dates are represented using different formats in the Patient Details table and the Clinic Attendance table
- address numbers are also split across different columns in the Patient Details table and the Address Details table, in the former address is split into 2 columns in the latter.
- some attributes can be derived from others, such as age from DOB in the first table.

The initial task was to arrive at a core matching algorithm to use for text matching for data elements, this one done by running a series of tests by making repeated matches between 2 datasets, in this case READ Code Version 2 and SNOMED CT. We have small sample sets of codes that have been matched and approved by clinicians, and therefore can be used to verify and rank the results of our experimental search process. In the next sections we describe the approach used, the results and how they are integrated into the metadata registry.

Name Matching

Initially we set up a system whereby the names of **DataItems** were matched by using their name attribute, using fuzzy matching defined by using the Levenstein distance algorithm. The algorithm is relatively straightforward and it measures the number of steps that would be needed to match or translate a text item A to another text item B, using insertion, deletion and substitution. It was first published by Levenstein [49] in the 1960s, and is still the standard for fuzzy text matching. Similar techniques had been explored by Damerau [12], and the Levenstein-Damerau algorithm allows transposition of 2 adjacent characters, in addition to insertion, deletion and substitution to achieve the translation. There have been a number of variants on the original algorithm, however the main improvement to its efficiency was made by Lowrance et al. [89].

The core idea is that the levenstein distance between 2 strings A and B, is given by the piecewise function lev(A,B):

$$lev(a,b) = \begin{cases} |a| & if\ |b| = 0 \\ |b| & if\ |a| = 0 \\ lev(tail(a), tail(b)) & if(a[0] = b[0]) \\ 1 + min \begin{cases} lev(tail(a), tail(b)) \\ lev(a, tail(b)) otherwise \\ lev(tail(a), tail(b)) \end{cases} \end{cases}$$

This was initially written as a Java functions, converted to Groovy [23], and later replaced by a library function from the Lucene search library [24]. The Groovy code is illustrated in appendix C. Initially a matching capability was made simply to provide search capability, however the task of matching whole datasets became significant, and thus further work was carried out to create an improved matching service for datasets. The goal was to create a set of 10 "possible" or "likely" matches from dataset B for each **DataElement** in dataset A, so that clinicians could review the list and approve a match.

Structural Matching

Structural matching is the other aspect in matching up **DataItems**, and our approach is based on graph theory, which we apply to MDML in order to represent different **DataModels** and **DataClasses**. By representing MDML structures in graph form we are able to apply similarity flooding techniques in order to compare different structures, this process is described in the next section.

Similarity Flooding

The basic similarity flooding algorithm was put forward by Melnik et al [55] in 2002 to compare Schemas, in particular SQL database schemas, but also XML Schemas. The technique itself has been adapted for different research goals, and variations on graph similarity have been put forward by many researchers [92, 64, 80], however for this exercised we have stuck with the original algorithm.

TestModel1 and TestModel2 are configured as perhaps the most simple sample **DataClass's** consisting two **DataClass's** arranged with a hierarchical relationship, so that the FormData and FormInputData are effectively inheriting the structure of their *parent* **DataClasses**. Both of these classes have a single **DataElement**, at_treatment and area_team_treatment, both of which have *type* relationship with the *varchar* type. The essential difference here is one of naming of the different DataItems, the structure is in essence the same.

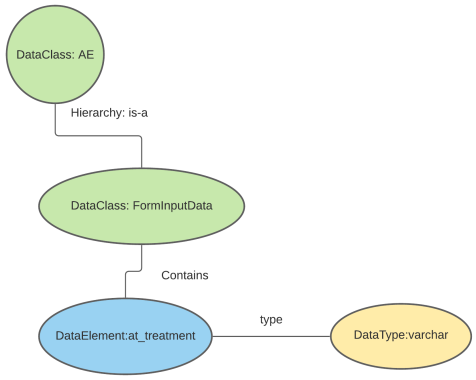

Figure 7.2: MDML TestModel1 as Graph

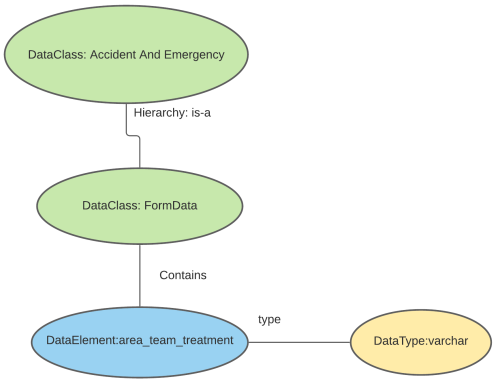

Figure 7.3: MDML TestModel2 as Graph

TestModel3 is slightly different in that it has a different overall structure, although one side of the structure is identical to that illustrated in the previous two TestModels. This enables us to see if sub-sections of the graph structure can be matched and identified through this technique. The TestModels used are illustrated in Figures 7.2, 7.3, and 7.4.

The Similarity Flooding method works by matching 2 graphs, so since we apply this to MDML **DataItems** the first step consists of converting the DataItem's into graphs, let's call them A and B. From here we build a connectivity graphs, this consists of taking pairs of nodes from $A \times B$, the pairwise connnectivity graph is defined as: $((x, y), p, (x_1, y_1)) \in PCG(A, B) \Leftrightarrow (x, p, x_1) \in A \text{and} (y, p, y_1) \in B$. Each pair of nodes, or **DataItems** is connected by an edge, we replace the edge by 2 edges running in each direction, each one carrying a weight, between 0 and 1. The idea is that if a node in one model is similar to a node in the other model, then one will expect it to have a similar set of edges connecting it to similar nodes. At the start of

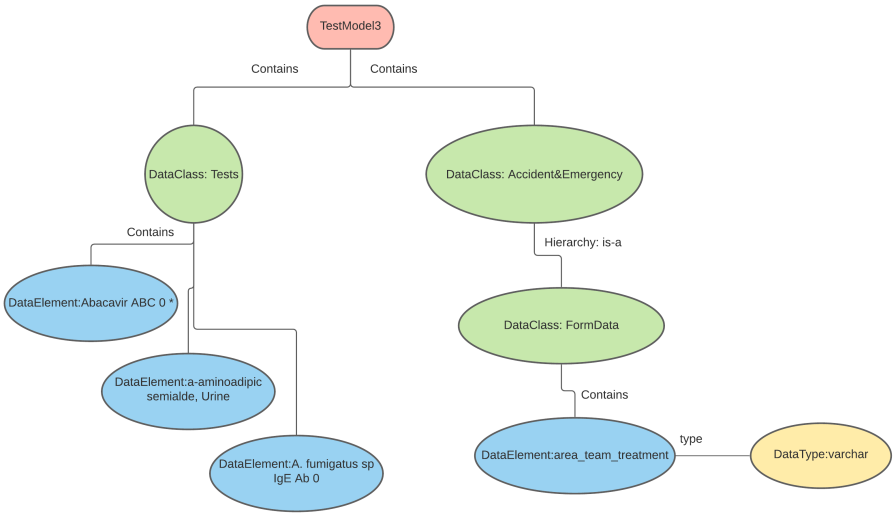

Figure 7.4: MDML TestModel3 as Graph

the calculation a propagation coefficient of 1.0 is set for each set of nodes, in each direction: $\omega((x, y), (x_1, y_1))$ and $\omega((x_1, y_1), (x, y))$. Where there are 2 edges outgoing the weighting is 0.5 for each edge. We then run a propagation iteration, repeatedly until either a fixed number of iterations has passed, or until the value of σ the similarity coefficient for the 2 models converges. For the most part 3 iterations is sufficient to achieve convergence, and in this case we limit the iterations to 3.

Let $\sigma(x, y) \geq 0$ be the similarity measure of $x \in A$ and $y \in B$, then we can refer to σ as the mapping factor, this starts off at 1.0 for each map pair, and is then incremented by the σ values of its related nodes, multiplied by the propagation coefficients on each respective edge. The core of the calculation is represented by the following:

$$\sigma^{i+1}(x, y) = \sigma^i(x, y) +$$
$$\sum_{(a_u,p,x \in A,(b_u,p,y) \in B)} \sigma^i(a_u, b_u)\dot{\omega}((a_u, b_u), (x, y)) +$$
$$\sum_{(x,p,a_v) \in A,(y,p,b_v) \in B} \sigma^i(a_v, b_v)\dot{\omega}((a_v, b_v), (x, y))$$

7.3 Methodology

The 2 matching techniques, text and structural, were applied primarily on MDML models of SNOMED and READ datasets, by using the metamodel and language developed in chapter

5 we have developed a unique way to match data elements. In the first case of name matching we separate different components of the model and build indexes which are based on each model component, i.e. DataElement, DataClass, etc. These indexes are then used in the prime matching exercise, and a number of different algorithms were then applied. By using the metadata registry application developed from chapter 5 we are able to model datasets such as SNOMED CT and use them to make comparisons of different data elements.

To implement the name matching experiments several software libraries were examined, including Lucene, Meta and other Java and python libraries, before settling with Terrier [50], a Java-based information retrieval library with the latest probability based algorithms available, and a sophisticated experimental test rig to test against. This enabled us to run tests using a variety of different search algorithms and compare the results against a pre-curated set of mappings.

The task was to match a set of READ codes, provided to us by NHS Digital against the SNOMED CT **DataModel** as imported into the metadata registry. For this set of 28 codes we were provided with a set of matching SNOMED codes, curated and approved by clinicians, enabling us to us this as a reference point. Each of these codes and descriptions was taken as a query made against the index on the SNOMED **DataModel**, the query results being 10 ordered **DataElements** from the model. The ideal behaviour being that each code would have the correct SNOMED code chosen in position 1, a score was provided that weighted the results accordingly and which is described in detail in the next section.

The similarity matching algoithms is shown below 1, this was coded into Java code and then applied to MDML models of SNOMED concepts using the MDML syntax.

Algorithm 1: Similarity Flooding

```
Data: GraphModel A, GraphModel B
Result: Compare GraphModel A with GraphModel B
Initialization: Search for DataItems;
Convert models into graphs;
Build Pairwise Connectivity Graph ;
Build Propagation Graph from Connectivity Graph;
while While n < limit do
    recalculate similarity;
    if convergence achieved then
        report results; stop;
    else
        continue similarity calculation;
```

Term Frequency / Inverse Document Frequency

The Term Frequency/Inverse Document Frequency or TF/IDF retrieval algorithm was first published by Jones and Roberton in 1976 [42, 75, 41]. It is one of the most successful search algorithms to be used in search engines ever since being published, and much work has been done on applying and tweaking the algorithm by various parties [91, 5]. Most other search algorithms are defined with reference to this algorithm, even the ones that performed in our tests marginally better. The codebase in Terrier codes the original algorithm presented in [75].

LDG Algorithm

The LDG algorithm is an extension of the Divergence from Randomness (DFR) algorithms, which have been documented in chapter 2, as introduced by Clinchant [8, 9], as part of the TREC[65, 86] project.

Divergence From Independence model based on Standardization

The DFIZ algorithm[43] is a statistical algorithm for term weighting without using term frequency normalization, it is claimed that it's performance matches other weighting methods for general information retrieval, and for our tests that would seem to be the case.

Search Algorithm Results

The most effective search algorithms for this exercise were the TF/IDF, the LDG and the DFIZ, as shown in Figure 7.5. What is surprising is that the BM25 which is generally considered one of the most effective search algorithms was not even close in this exercise. However measuring search is very much an empirical process, and it is very often the case that different algorithms are best for different use cases.

Algorithm	MAP	GM-MAP
PL2F	0.0014	0.0002
TF-IDF	0.0188	0.0006
BM25	0.0011	0.0001
TF	0.0025	0.0002
ML2	0.0014	0.0002
MDL2	0.0046	0.0003
LGD	0.019	0.0006
IFB2	0.0183	0.0006
DirichletLM	0.009	0.0004
DFReeKLIM	0.0074	0.0004
DFR_BM25	0.0186	0.0006
BB2	0.0183	0.0006
BM25F	0.0011	0.0001
DFIZ (B4)	0.0191	0.0006

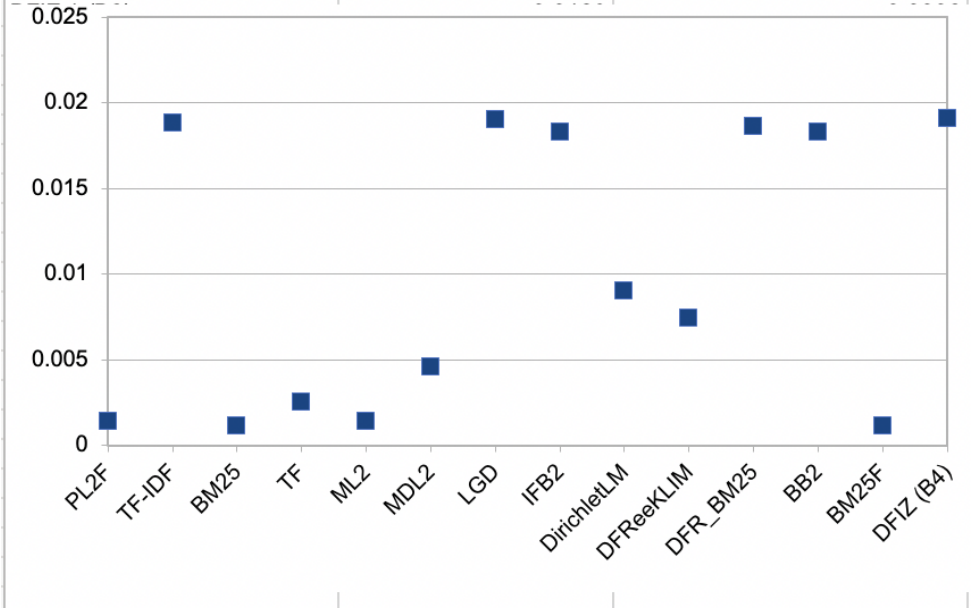

Figure 7.5: Search Algorithm Comparison

Results

The results obtained from these relatively simple models indicate that we are able to match similar groups of **DataItems** effectively for small models. When larger models are considered

the complexity rises to the point whereby the results are not meaningful in any useful way. The process is slower, more iterations are needed and a lot more user input is required.

What is notable is that finding similarity of small models in larger models is effective, being able to apply this algorithm to MDML models is a novel application of the similarity algorithm. In running this algorithm one can expect a result of 1 for a full match, and 0 for no match. Within 3 iterations we are able to at least 3 classes exactly and all the others to some degree, which may not give a high enough confidence to be able to run it unsupervised. However it does give enough confidence to allow an initial sorting to occur which can then be verified by an expert. Full results are shown in the appendix C.

7.4 Summary

Applying search algorithm over MDML models allows mapping of data schemas to take place at the *abstraction level* rather than the *instance level*, and in turn this allows partial automation of the data management process. This can be carried out at present and bringing the specifications into a unified schema within a registry process allows mappings to be added and used at different levels of granularity with a precise set of unique references. This in turn allows other applications to use the mapping results stored in the registry for data cleansing, validation and conformity checking via standardised API's.

The search techniques described using the TF-IDF algorithms have allowed a significant reduction in time taken to complete the mapping task for clinicians, and thus proved effective in practice. Basing the mapping process on the DSL metamodel allows the process to achieve a much higher degree of automation than simply using the core structure of the different datasets, it also allows the mappings to be specified precisely with ease and thus simplifies the administration associated with the process. The similarity techniques described have as yet only been tested on models, and have not been applied in the field, however it is expected that when added to the search techniques they will also further enhance the data mapping use case. Again similar benefits of precised mappings, recorded in a form that can be accessed automatically accrue from using MDML.

Chapter 8

Conclusions

8.1 Overview

As stated in chapter 1, this thesis provides several novel research contributions including:

- An improved metamodel for metadata management in the form of a domain specific language - Metadata Modelling language (MDML), based on a formal specification, for metadata registries, enabling automatic registration and classification of data elements.
- A registry design based on MDML which allows improved semantic mapping of data items from diverse sources to be registered, defined and mapped in a way which is machine-accessible for automatic classification and code-generation tasks.
- A deep-search capability for researchers to access useful information about research data before being able to access the data directly.
- The ability to contain other descriptive metadata standards within the MDML model, allowing standards such as DCAT to be incorporated within MDML.

The latter capability, based on this work, is currently in use online publically at the largest medical data registry in the UK, the HDR UK Health Data Gateway [85], currently encompassing over 37 healthcare and governmental organisations.

8.2 Publications

In the course of this work a number of peer-reviewed publications have been written detailing aspects of the work described here, 2 are from journals and 4 are from conferences, as detailed below:

1. **A formal, scalable approach to semantic interoperability** Jim Davies, James Welch, David Milward and Steve Harris [14]

2. **Dataset Management Using Metadata** David Milward [58]

3. **Model Driven Data Management in Healthcare** David Milward [57] (This won the overall **Best Paper Award** at the Modelsward Conference, Prague, 2019).

4. **Domain Specific Modelling for Clinical Research** Jim Davies, Jeremy Gibbons, Adam Milward, David Milward, Seyyed Shah, Monika Solanki and James Welch [39]

5. **Formal model-driven engineering of critical information systems** Jim Davies, David Milward, Chen-Wei Wang and James Welch [38]

6. **Compositionality and Refinement in Model-Driven Engineering** Jim Davies, Jeremy Gibbons, David Milward and James Welch [36]

In addition a presentation was made at the DSLDI workshop at SPLASH (Boston, 2018) on the subject: **Healthcare Data Management using Domain Specific Languages for Metadata Management**, which followed up on the paper (4) presented in 2015.

The work presented in chapter 4 was originally carried out by the author as an attempt to build a formal model of the ISO registry using Z. It formed the basis of further work by Prof. Davies, James Welch, the author and Steve Harris. The resultant work was published in the paper [14]. This paper gives a slightly extended treatment of the subject, and there are a few minor differences, such as *purpose sets* in this thesis being called *belief sets* in the paper, in terms of this formal specification one is a synonym for the other.

8.3 Conclusion and Future work

The development of MDML has helped researchers to access technical metadata for heterogeneous datasets in a way not previously possible, and in a way that allows meaningful integration of existing healthcare research to be incorporated into new studies.

The new MDML metamodel addresses key shortcomings in the ISO 11179 standard, and allows metadata registries to be populated on an automatic or semi-automatic basis instead of the manual way ISO 11179 registries have needed to be set up until now.

This work involves the integration of many smaller technologies and technical spaces, and clearly the core principles stretch well beyond the domain of healthcare. However there are still many aspects to work on in order to improve the existing system.

The data matching and search capability for both name matching and structural matching need further tuning and assessment, which will take both time and resources. The code generation form Xtext to the current groovy-java implementation is not smooth and involves a lot of small fixes, which with a more code-generation capability could perhaps be better managed. Currently validation capabilities rely on a combination of regular expressions and the DROOLS rule engine, ideally the verification rules engines would be built into the framework, perhaps with the addition of metadata-specific rules language.

This work is a demonstration of applying formal methods to a software engineering problem to produce a clear specification, which has resulted in a software system capable of enabling semantic interoperability in UK healthcare research. The system is based on the MDML language, which in turn is based around the semantic model specified in chapter 4. Increasing the level of semantic interoperability between research hubs in turn enables researchers to obtain better quality of data from more sources, helping to raise the quality of healthcare research. It provides an object model which derives from and describes the structure of the entities implemented on homogeneous information systems, this allows semantic interoperability over structure. No attempt has been made to address behaviour in the way of object-oriented programming languages, and this is a potential area of future research. Initially we did attempt to model the registry using the B-toolkit, and this is perhaps still a valid area of future research.

There is of course more work to be done, not everything specified in the specification in chapter 4 has been seemlessly implemented, the ability to constrain and validate data is only partially implemented through using regular expressions and policies. Constraints and rules cannot be imported from OWL based ontologies, and there is work to do in ensuring that Data Models imported into other Data Models are fully consistent with one another before being finalised. In the area of matching at present we rely on empirical techniques, and ultimately expert input. Moreover given the diverse nature of data modelling it is likely that we will continue to rely on empirical techniques, however there matching and similarity techniques reduce the manual work which experts need to do, and therefore it would be good if we could always get the *right match* in the top 5 search suggestions rather than the top 10. It is likely that as some of the registries increase in size and content that different statistical techniques will become viable with regard to the matching problem.

Appendix A

Key ISO 11179 Definitions

9 Concepts package

9.1 Concepts metamodel region

9.1.1 Overview

The Concepts metamodel region is illustrated in Figure 8. The purpose of the Concepts Metamodel Region is to describe *Concept*s (9.1.2.1) (abstract units of knowledge) and the various *Relation*s (9.1.2.4) which might hold among Concepts. Ontologies are supported as *Concept_Systems* (9.1.2.2) with formal semantics through the use of *Assertions* (9.1.2.3). Annex E provides examples using SKOS, ORM, OWL and CLIF.

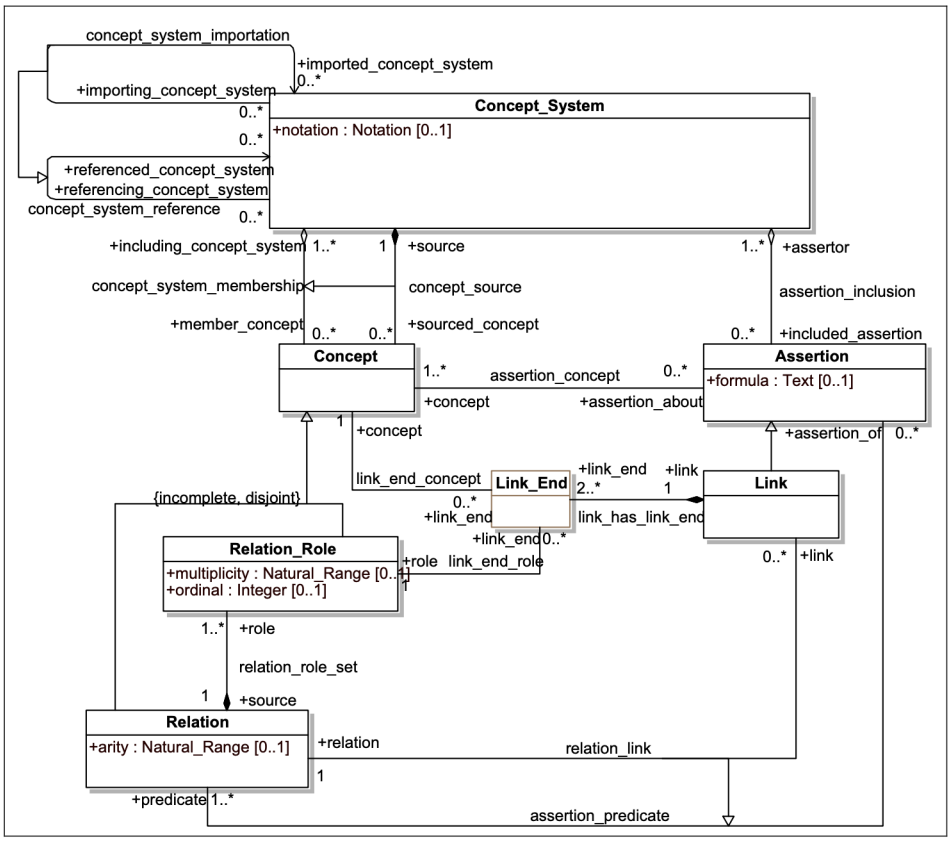

Figure 8 — Concepts metamodel region

10 Binary Relations package

10.1 Binary Relations metamodel region

10.1.1 Overview

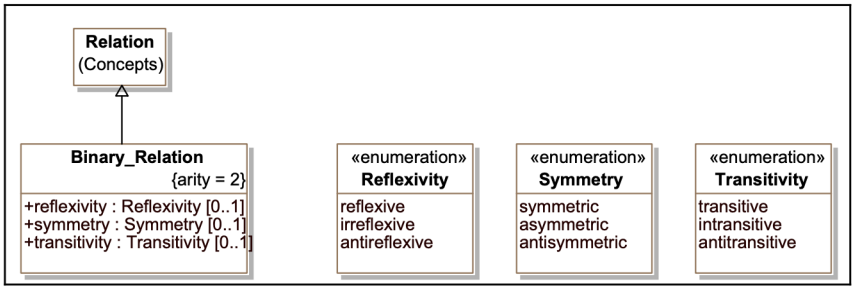

Figure 10 — Binary Relations metamodel region

10.1.2 Classes in the Binary_Relations metamodel region

10.1.2.1 Relation class

The *Relation* class is described in 9.1.2.4.

10.1.2.2 Binary_Relation class

10.1.2.2.1 Direct superclass

Relation (9.1.2.4)

10.1.2.2.2 Description of Binary_Relation

Binary_Relation is a class each instance of which models a **binary relation** (3.2.10), a **relation** (3.2.119) of **arity** (3.2.4) 2 (i.e. having two **link ends** (3.2.70)).

Most common semantic relations are binary, e.g., equals, less than, greater than, is-a, part-of, etc. A example of a relation which is not binary would be betweeness. Binary relations are commonly represented as edges (or directed edges for asymmetric binary relations) in graphs, cf. the RDF (Resource Description Framework) of the W3C.

Below is a table of examples of some binary relationships and their characterization.

Table 4 — Examples of binary relations and their characterization

Relation	Symmetry	Reflexivity	Transitivity
equals	symmetric	reflexive	transitive
not equals	symmetric	antireflexive	intransitive
less than	antisymmetric	antireflexive	transitive
less than or equal	asymmetric	reflexive	transitive
similar	symmetric	reflexive	intransitive

11 Data Description package

11.1 High-level Data Description metamodel region

11.1.1 Overview

A high level overview of the metamodel can be found in Figure 11. It shows four classes: *Conceptual_Domain* (11.1.2.2), *Value_Domain* (11.1.2.3), *Data_Element* (11.1.2.4), and *Data_Element_Concept* (11.1.2.5). The Figure also shows four associations among the four classes: *value_domain_meaning* (11.1.3.1), *data_element_domain* (11.1.3.2), *data_element_meaning* (11.1.3.3), and *data_element_concept_domain* (11.1.3.4).

The following text describes the classes and associations shown in the Figure. It also describes a constraint on the high level metamodel not visible in the UML diagram. More detailed descriptions, e.g. of the class attributes, follow in subsequent subclauses.

Figure 11 can be partitioned into two horizontal parts, one upper part comprised of *Data_Element_Concept* and *Conceptual_Domain* and a second lower part comprised of *Data_Element* and *Value_Domain*. This view effectively splits the metamodel between a conceptual (or semantic) level (at the top) and a representational level (below). The representational level describes the information artifacts (in contrast to the semantic constructs of the upper level).

This high-level metamodel omits many details, e.g. attributes and some associations, in the interest of clarity of exposition. For a complete characterization of the metamodel the reader must consult the more detailed discussions which follow. A consolidated detailed metamodel is presented in 11.6.

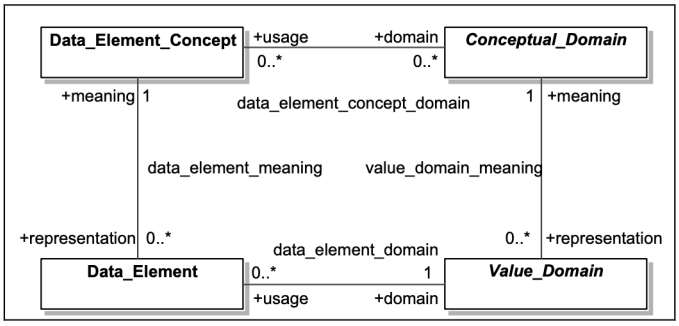

Figure 11 — High-level Data Description metamodel

11.1.2 Classes of High-level Data Description metamodel

11.1.2.1 Overview

The classes shown in Figure 11 are described below starting with *Conceptual_Domain* (11.1.2.2), and proceeding clock-wise around the Figure.

11.1.2.2 Conceptual_Domain class

11.1.2.2.1 Direct superclass

Concept (9.1.2.1)

11.1.2.2.2 Description of Conceptual_Domain

Conceptual_Domain is a class each instance of which models a **conceptual domain** (3.2.21), a set of **value meaning**s (3.2.141) which may either be enumerated or expressed via a description.

For example, one possible conceptual domain could be countries of the world. It might be associated with two **value domain**s (3.2.140): three letter country codes, and full country names. The conceptual domain might be used in several **data element concept**s (3.2.29), e.g., “person’s country of residence”, “person’s country of birth”, “person’s country of citizenship”.

A conceptual domain can facilitate the mapping of equivalent values of two or more value domains that share the conceptual domain.

Conceptual_Domain may participate in the following associations:

— *data_element_concept_domain* (11.1.3.4) to zero or more *Data_Element_Concept*s (11.1.2.5) which reference the domain for its associated value meanings;

— *value_domain_meaning* (11.1.3.1) to zero or more *Value_Domain*s (11.1.2.3) which provide representation for the conceptual domain.

Conceptual_Domain is further described in 11.3.2.1.

11.1.2.3 Value_Domain class

Value_Domain is a class each instance of which models a **value domain** (3.2.140), a collection of **permissible value**s (3.2.96). A value domain provides representation, but has no implication as to what **data element concept** (3.2.29) the values are associated with, nor what the values mean. Permissible values are **designation**s (3.2.51), bindings of **sign**s (3.2.123) (values) to their corresponding **value meaning**s (3.2.141).

Value_Domain is associated with a *Conceptual_Domain* Go to page 22 through the association *value_domain_meaning* (11.1.3.1). Through this association, a value domain provides a representation for the **conceptual domain** (3.2.21) and the conceptual domain provides meaning for the value domain.

An example of a conceptual domain and a set of value domains is ISO 3166, Codes for the representation of names of countries. For instance, ISO 3166 describes the set of seven value domains: short name in English, official name in English, short name in French, official name in French, alpha-2 code, alpha-3 code, and numeric code.

Additional examples of value domains are:

— Sex, which contains two designations (permissible values), M -> Male and F-> Female;

— Parent, which contains two designations (permissible values), M -> Mother and F -> Father.

NOTE These two value domains are defined over the same set of values (signs), but they are mapped to separate conceptual domains.

Value_Domain shall participate in the following association:

— *value_domain_meaning* (11.1.3.1) to exactly one *Conceptual_Domain* (11.1.2.2) which provides meaning to the value domain.

Value_Domain may participate in the following association:

— *data_element_domain* (11.1.3.2) to zero or more *Data_Element*s (11.1.2.4) for which the value domain provides a set of permissible values.

Value_Domain is further described in 11.3.2.5.

Value domains may be reused for multiple data elements - see the discussion of countries of the world in 11.1.2.2 above.

11.1.2.4 Data_Element class

Data_Element is a class each instance of which models a **data element** (3.2.28), a unit of **data** (3.2.27) that is considered in context to be indivisible. A data element is a basic unit of data of interest to an organization, for which the definition, identification, representation, and permissible values are specified by means of a set of attributes. Examples of data element include: a column in a table of a relational database, a field in a record or form, an XML element, the attribute of a Java class, or a variable in a program. The description of data elements is a major purpose of ISO/IEC 11179 Metadata Registries.

Data_Element shall participate in the following associations:

— *data_element_meaning* (11.1.3.3) to exactly one *Data_Element_Concept* (11.1.2.5) which the data element represents, and which gives meaning to the data element;

— *data_element_domain* (11.1.3.2) to exactly one *Value_Domain* (11.1.2.3) which specifies the permissible values that the data element may assume.

Data_Element is further described in 11.5.2.1.

11.1.2.5 Data_Element_Concept class

11.1.2.5.1 Direct superclass

Concept (9.1.2.1)

11.1.2.5.2 Description of Data_Element_Concept

Data_Element_Concept is a class each instance of which models a **data element concept** (3.2.29), a **concept** (3.2.18) that is an **association** (3.1.2) of a **property** (3.2.100) with an **object class** (3.2.88). A data element concept is a concept that can be represented in the form of a **data element** (3.2.28), described independently of any particular representation.

A data element concept is a usage of a **conceptual domain** (3.2.21), e.g., “person’s country of residence” vs. “country”, which effectively narrows the meaning of the conceptual domain.

A data element concept is an abstraction of one or more data elements. Each data element addresses issues of concrete representation, e.g., codes, measurement units, etc. A data element concept may be represented by multiple data elements, which may vary in their **value domain**s (3.2.140).

A data element concept can facilitate the mapping of equivalent values of two or more data elements that share the data element concept.

Data_Element_Concept may participate in the following associations:

— *data_element_concept_domain* (11.1.3.4) to zero or more *Conceptual_Domain*s (11.1.2.2);

— *data_element_meaning* (11.1.3.3) to zero or more *Data_Element*s (11.1.2.4).

Data_Element_Concept is further described in 11.2.2.3.

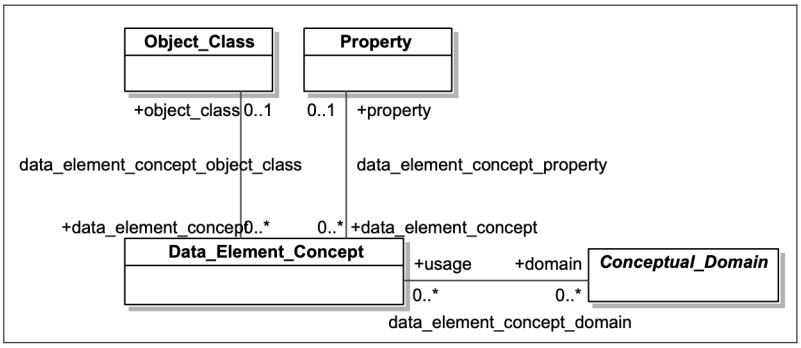

Figure 12 — Data_Element_Concept metamodel region

11.2.2 Classes in the Data_Element_Concept region

11.2.2.1 Object_Class class

11.2.2.1.1 Direct superclass

Concept (9.1.2.1)

11.2.2.1.2 Description of Object_Class

Object_Class is a class each instance of which models an **object class** (3.2.88). An object class is a **concept** (3.2.18) that represents a set of ideas, abstractions, or things in the real world that can be identified with explicit boundaries and meaning and whose properties and behavior follow the same rules. It may be either a single or a group of associated concepts, abstractions, or things.

An *Object_Class* may have a *data_element_concept_object_class* association (11.2.3.3) with zero or more *Data_Element_Concepts* (11.2.2.3), where the object class describes the ideas, abstractions or things in the real world that are represented by the **data element concept**s (3.2.29).

EXAMPLE The *Object_Class* “Person” could be associated with the *Data_Element_Concept* “Person height”.

11.2.2.2 Property class

11.2.2.2.1 Direct superclass

Concept (9.1.2.1)

11.2.2.2.2 Description of Property

Property is a class each instance of which models a **property** (3.2.100), a quality common to all members of an **object class** (3.2.88). A property may be any feature that humans naturally use to distinguish one individual object from another. It is the human perception of a single quality of an object class in the real world. It is conceptual and thus has no particular associated means of representation by which the property can be communicated.

A *Property* may have a *data_element_concept_property* association (11.2.3.1) with zero or more *Data_Element_Concept*s (11.2.2.3).

According to the cardinality constraints for the *described_value_domain_meaning* association (11.3.3.3) there must also exist an instance z of the *Described_Conceptual_Domain* such that z is the meaning of x. Then it must be the case that z is equal to y, i.e., the meaning of x must be same according to both the *value_domain_meaning* and *described_value_domain_meaning* associations.

Constraint #3: Mapping Enumerated_Value_Domains across Enumerated_Conceptual_Domains

Suppose that there exists an instance *u* of the class *Enumerated_Value_Domain* (11.3.2.6), such that the instance *v* is the meaning of *u* according to the *value_domain_meaning* (11.3.3.1) association (since every instance of a *Enumerated_Value_Domain* is also a *Value_Domain* (11.3.2.5)) where v is some instance of a *Conceptual_Domain* (11.3.2.1) (either a *Described_Conceptual_Domain* (11.3.2.4) or an *Enumerated_Conceptual_Domain* (11.3.2.2)). There must exist such an instance *v* according to the cardinality constraints on the *value_domain_meaning* association.

Now for each instance u of the class *Enumerated_Value_Domain* there must exist a non-null set W of the members of the *Permissible_Values* (11.3.2.7) class according to the *permissible_value_set* (11.3.3.5) association. For each element w_i of W there is an exactly one instance m_i of the class *Value_Meaning* (11.3.2.3) such that m_i is the meaning of w_i according to the *permissible_value_meaning* (11.3.3.4) association. Let M be the set union of these m_i. Now consider the (possibly empty) sets E_i each of which is the unions of instances of the class *Enumerated_Conceptual_Domain* which are the containing domains of the various *Value_Meaning*s of each m_i. Then it must be the case that for every m_i in M there exists an instance e in the set E_i such that e is equal to v.

NOTE The final existential quantification (rather than universal quantification) over the elements of each set E_i arises because we no longer constrain *Value_Meanings* to exist in a single *Enumerated_Conceptual_Domain*.

11.3.4.3 value_domain_subset association constraints

A *Value_Domain* (11.3.2.5) that particates in a *value_domain_subset* (11.3.3.6) association may not reference itself, either directly or indirectly through a chain of such associations.

11.3.4.4 Consistent dimensionalities

Conceptual_Domains (11.3.2.1) may have an attribute *dimensionality* (11.3.2.1.3.1). *Value_Domains* (11.3.2.5) may have an attribute *unit_of_measure* (11.3.2.5.2.4) of type *Unit_of_Measure* (11.4.2.1). Suppose that we have an instance c of the class *Conceptual_Domain* and an instance v of a *Value_Domain* such that c is the meaning of v according to the *value_domain_meaning* (11.3.4.2) association (or some equivalent path as above). Suppose that d (of type *Dimensionality* (11.4.2.3)) is the *dimensionaility* attribute of the instance c. Suppose that e is the *dimensionality* of the *unit_of_measure* of v. Then it must be the case that the d is equal to e.

In plain English, the *dimensionality* of the *unit_of_measure* of a *Value_Domain* must be the same as the *dimensionality* of the *Conceptual_Domain* which provides the meaning of the *Value_Domain*.

11.6 Consolidated Data Description Metamodel

A consolidated metamodel is shown in Figure 16. This combines the Data_Element_Concept, Data_Element, and Conceptual and Value_Domain regions of the model.

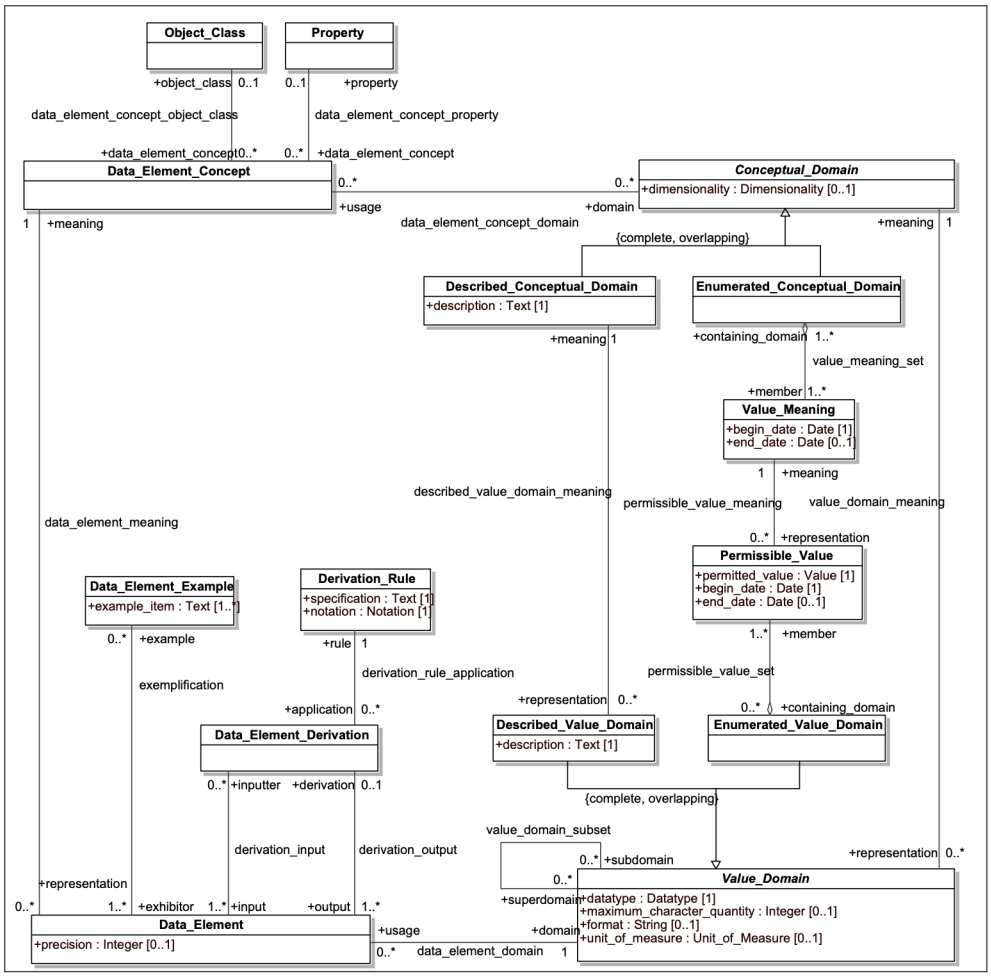

Figure 16 — Consolidated Data Description metamodel

11.7 Types of Concepts in the Data Description Metamodel

Figure 17 shows the classes in the Data Description package which are types of *Concepts* (i.e. they are sub-typed from the *Concept* class).

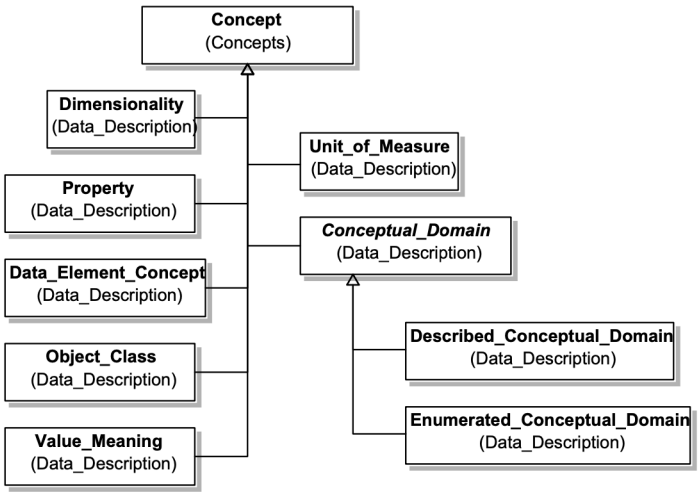

Figure 17 — Types of Concepts in the Data Description package

12 Basic attributes

12.1 Use of basic attributes

This Clause is intended to provide continuity from ISO/IEC 11179-3:1994, which edition focused on basic attributes of data elements. However, the scope of this Clause extends beyond just data elements, to include: data element concepts, conceptual domains, value domains, permissible values and value meanings.

A mapping among the 1994 basic attributes, the 2011 basic attributes and the 2011 metamodel can be found in Annex C.

Clauses 6 through 11 describe a model for specifying metadata in a registry. However, sometimes the requirement for metadata specification exists outside the context of a registry, for example as part of an International Standard.

A specification of metadata consists of a set of attributes, and relationships among those attributes. This Clause specifies a set of *basic* attributes to be used in contexts other than a metadata registry. *Basic* means that they are frequently needed to specify a metadata item. The attributes specified in this Clause are also considered *basic* in the sense that additional attributes might be required when the metadata items are used in a particular context.

6 Basic package

6.1 Overview of Basic package

The Basic package specifies common datatypes and classes for use elsewhere in the metamodel. The contents of the package are grouped into two regions: “basics types” and “basic classes”, as described below.

6.2 Basic Types metamodel region

6.2.1 Overview of Basic Types

The Basic Types metamodel region specifies the **datatype**s (3.1.8) used in the specification of the **metamodel** (3.2.80). A datatype is a set of distinct values, characterized by properties of those values and by operations on those values (ISO/IEC 11404). All of the other types used in the model are based on this core set of types, and any compliant implementation of a metadata registry should include an implementation of the semantics specified in these core types.

NOTE The datatypes that are described in this section are used in specification of the metamodel itself, and are not intended to constrain the datatypes that may be used in 11.3.2.9.

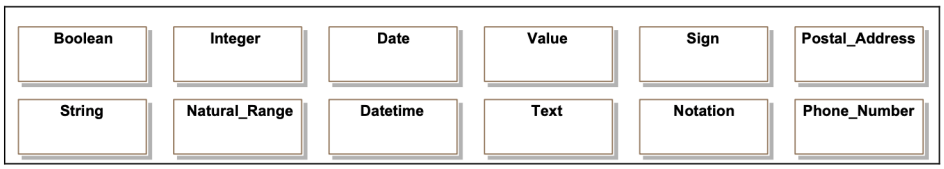

Figure 3 — Basic types metamodel region

Appendix B

MDML: Revised Grammar

```
grammar uk.ac.ox.cs.MDML with org.eclipse.xtext.common.Terminals

generate mdml "http://www.ac.uk/ox/cs/MDML"

DomainRegistry :
{DomainRegistry} (registeredItems += Model)*
;

Model:
model = (ConceptModel|DataModel)
;

Concept:
'Concept' name = Path ('extends' superType=[Concept])? '{'
        descriptors += Term*
        '}'
;

Term:
term=ID
;

ConceptModel:
'ConceptModel' name = Path 'status:' status = Status 'domainid:' guid = GUID '@' '(' Version ')' '{'
        (concepts += Concept)*
```

```
26          (relations += Relationship)*
27          (constraint += Constraint)*
28          (predicate += Predicate)*
29          '}'
30  ;
31
32
33  DataModel:
34  'DataModel' name = Path 'status:' status = Status 'domainid:' guid = GUID '@' '(' Version ')' '{'
35          (elements += DataItem)*
36          (relations += Relationship)*
37          (constraint += Constraint)*
38          (predicate += Predicate)*
39          '}'
40  ;
41
42  DataItem:
43  dataitem = (DataModel | DataClass | DataElement | DataType | Import | ExtensionItem )
44  ;
45
46
47  QualifiedName:
48  ID('.' ID)*
49  ;
50
51  Path:
52  QualifiedName('/' QualifiedName)*
53  ;
54
55  GUID:
56  ID(('.' ID)|('/' ID))*
57  ;
58  Version:
59  INT '.' INT '.' INT
60  ;
61
```

```
Reference:
"ref" name=ID ":" type=[DataItem]
;

RelationSource:
DataModel | DataClass | DataElement | DataType
;

RelationDestination:
DataModel | DataClass | DataElement | DataType
;

Relationship:
'relationship' (src = RelationSource ':' dest = RelationDestination)
;

ExtensionItem:
'extension' key = ID '=' value = STRING';'
;

Import:
'import' importedNamespace = QualifiedNameWithWildcard
;

QualifiedNameWithWildcard:
QualifiedName '.*'?
;

DataType:
PrimitiveType | ReferenceType | EnumeratedType
;

EnumeratedType:
'enumtype' 'status:' status = Status 'domainid:' guid = GUID '@' '(' Version ')'
name = QualifiedName ':' type=PrimitiveType(array ?='[' (length=INT)? ']')?
;
```

```
98
99  ReferenceType:
100 'reftype' 'status:' status = Status 'domainid:' guid = GUID '@' '(' Version ')'
101 name = QualifiedName ':' type=[DataClass]
102 ;
103
104 PrimitiveType:
105 {PrimitiveType} ('datatype' 'status:' status = Status 'domainid:' guid = GUID '@' '(' Version ')'
106 name = QualifiedName ':' type=BasicType (rule=[Constraint])?
107 )
108 ;
109
110 BasicType:
111 typename=('STRING'|'INT'|'BOOLEAN')
112 ;
113
114 BOOLEAN:
115 ('0'|'1')
116 ;
117 ContainerElement:
118 DataClass| DataElement
119 ;
120
121 DataClass:
122 'DataClass' name = QualifiedName 'status:' status = Status 'domainid:' guid = GUID'@' '(' Version ')'
123 ('extends' superType=[DataClass])? '{'
124         (dataelements += ContainerElement)*
125         (extension += ExtensionItem)*
126         (relations += Reference)*
127         '}'
128 ;
129
130 DataElement:
131 'DataElement' 'status:' status = Status 'domainid:' guid = GUID '@' '(' Version ')'
132 name = QualifiedName ':' type = [DataType|QualifiedName]
133 (extension += ExtensionItem)*
```

```
134 ;
135
136 Status:
137 ('draft'|'finalised'|'superseded')
138 ;
139
140 Constraint:
141 {Constraint} 'constraint' name = QualifiedName '=' (( src = DataElement ':')?
142 pred = [Predicate] (':' dest = DataElement)?
143 )
144 ;
145
146 Predicate:
147 'predicate' name = QualifiedName ':' predicate = STRING
148 ;
```

Appendix C

Search Experiments: Background

The core dataset used for this experiment was SNOMED CT, which was converted into MDML, and then converted into TREC format - for ease of integration into the *Terrier* experimental framework. A sample of the file C.1 is shown below:

```
<DOC>
<DOCNO>4275850</DOCNO>
<DOCREF>447257003</DOCREF>
<DOCTITLE>"Linked to" reference set attribute (foundation metadata concept)</DOCTITLE>
<DOCDESC>"Linked to" reference set attribute</DOCDESC>
</DOC>
<DOC>
<DOCNO>4275851</DOCNO>
<DOCREF>258750005</DOCREF>
<DOCTITLE>% abnormal forms (qualifier value)</DOCTITLE>
<DOCDESC>% abnormal forms
percent abnormal forms</DOCDESC>
</DOC>
<DOC>
<DOCNO>4275852</DOCNO>
<DOCREF>258745004</DOCREF>
<DOCTITLE>% activity (qualifier value)</DOCTITLE>
<DOCDESC>% activity
percent activity</DOCDESC>
</DOC>
<DOC>
<DOCNO>4275853</DOCNO>
<DOCREF>258746003</DOCREF>
<DOCTITLE>% aggregation (qualifier value)</DOCTITLE>
<DOCDESC>% aggregation
percent aggregation</DOCDESC>
</DOC>
<DOC>
```

Figure C.1: Snomed in TREC format

The SNOMED model was used to create the index on which searches were made. The query-text file C.2 then provided the 28 read codes, in the appropriate terrier bulk input format, as input for the search engine.

```
 6 <TOP>
 7 <NUM>2</NUM><TITLE>
 8 Mean corpusc  haemoglobin(MCH)
 9 </TITLE>
10 </TOP>
11 <TOP>
12 <NUM>3</NUM><TITLE>
13 Mean corpusc  Hb  conc  (MCHC)
14 </TITLE>
15 </TOP>
16 <TOP>
17 <NUM>4</NUM><TITLE>
18 Plasma viscosity
19 </TITLE>
20 </TOP>
21 <TOP>
22 <NUM>5</NUM><TITLE>
23 Erythrocyte sedimentation rate
24 </TITLE>
25 </TOP>
26 <TOP>
27 <NUM>6</NUM><TITLE>
28 Prothrombin time
29 </TITLE>
30 </TOP>
31 <TOP>
32 <NUM>8</NUM><TITLE>
33 Thrombin time
34 </TITLE>
35 </TOP>
36 <TOP>
37 <NUM>9</NUM><TITLE>
38 International normalised ratio
39 </TITLE>
40 </TOP>
```

Figure C.2: READ code samples

The searches, essentially all descriptions of READ (version2 codes) which were provided with an agreed mapping to SNOMED CT. The results were stored in a another terrier file C.3 as a list of ordered results for each query.

Each of the results is given a weighting based on its priority in the list, and the whole set of results analysed to produce an average results of each search algorithm, these are then displayed in the spreadsheet and graph.

```
 1 1 0 1239 1
 2 1 0 1502 1
 3 1 0 4462 1
 4 1 0 4569 1
 5 1 0 5472 1
 6 1 0 5502 1
 7 1 0 6471 1
 8 1 0 6480 1
 9 1 0 6664 1
10 1 0 6824 1
11 2 0 414 1
12 2 0 1894 1
13 2 0 3785 1
14 2 0 4720 1
15 2 0 5894 1
16 2 0 6736 1
17 2 0 7113 1
18 2 0 7555 1
19 2 0 7749 1
20 2 0 7808 1
21 3 0 141 1
22 3 0 148 1
23 3 0 813 1
24 3 0 1610 1
25 3 0 2429 1
26 3 0 3059 1
27 3 0 3272 1
28 3 0 3398 1
29 3 0 3614 1
```

Figure C.3: Search match results

Levenstein Distance

```
1       private static int levensteinDistance(String str1, String str2) {
2               def str1_len = str1.length()
3               def str2_len = str2.length()
4               int[][] distance = new int[str1_len + 1][str2_len + 1]
5               (str1_len + 1).times { distance[it][0] = it }
6               (str2_len + 1).times { distance[0][it] = it }
7               (1..str1_len).each { i ->
8                       (1..str2_len).each { j ->
9                               distance[i][j] = [distance[i-1][j]+1, distance[i][j-1]+1, str1[i-1]==
                                    str2[j-1]?distance[i-1][j-1]:distance[i-1][j-1]+1].min()
10                      }
11              }
12              distance[str1_len][str2_len]
```

Similarity Results

TestModel1-TestModel2

The output from the Similarity Flooding routine comparing TestModel1 and TestModel2 is shown below:

```
1               Matching TestModel1 with TestModel2.
2               Creating propagation graph: 0.001 sec
3               Propagation graph contains 3 bidirectional arcs and 15 nodes
4               Iterating over (3 arcs, 4 nodes) x (3 arcs, 4 nodes):.(0:2.2607766610417563)
                    .(1:0.6934715407122183).(2:0.25678160250146415).(3:0.097639922514407)
                    .(4:0.037270580061882164).(5:0.014234727199230591).(6:0.005437105760557341)
                    .(7:0.0020767853531655158). Time: 0.001 sec
5               [at\_treatment,area\_team\_treatment: sim=1.0, init=1.0, N=1.0, N1=1.0]
6               [FormInputData,FormData: sim=1.0, init=1.0, N=1.0, N1=1.0]
7               [AE,Accident And Emergency: sim=0.6180340557275542, init=1.0, N
                    =0.6180344478216819, N1=0.6180340557275542]
8               [varchar,varchar: sim=0.6180340557275542, init=1.0, N=0.6180344478216819, N1
                    =0.6180340557275542]
9               [varchar,FormData: sim=3.8699690402476777E-4, init=1.0, N
                    =0.0010131712259371832, N1=3.8699690402476777E-4]
```

10 [FormInputData,area_team_treatment: sim=3.8699690402476777E−4, init=1.0, N
=0.0010131712259371832, N1=3.8699690402476777E−4]

11 [varchar,area_team_treatment: sim=3.8699690402476777E−4, init=1.0, N
=0.0010131712259371832, N1=3.8699690402476777E−4]

12 [FormInputData,Accident And Emergency: sim=3.8699690402476777E−4, init=1.0, N
=0.0010131712259371832, N1=3.8699690402476777E−4]

13 [at_treatment,Accident And Emergency: sim=3.8699690402476777E−4, init=1.0, N
=0.0010131712259371832, N1=3.8699690402476777E−4]

14 [AE,varchar: sim=3.8699690402476777E−4, init=1.0, N=0.0010131712259371832, N1
=3.8699690402476777E−4]

15 [AE,FormData: sim=3.8699690402476777E−4, init=1.0, N=0.0010131712259371832,
N1=3.8699690402476777E−4]

16 [at_treatment,varchar: sim=3.8699690402476777E−4, init=1.0, N
=0.0010131712259371832, N1=3.8699690402476777E−4]

17 [varchar,Accident And Emergency: sim=3.8699690402476777E−4, init=1.0, N
=0.0010131712259371832, N1=3.8699690402476777E−4]

18 [at_treatment,FormData: sim=3.8699690402476777E−4, init=1.0, N
=0.0010131712259371832, N1=3.8699690402476777E−4]

19 [AE,area_team_treatment: sim=3.8699690402476777E−4, init=1.0, N
=0.0010131712259371832, N1=3.8699690402476777E−4]

Significant similarities are arrived at after 3 iterations of the propagation algorithm:

- at_treatment,area_team_treatment: 1.0
- FormInputData,FormData: 1.0
- AE,Accident And Emergency: 0.6180340557275542
- varchar,varchar: 0.6180340557275542

The other map pairs yield an insignificant similarity score in comparison.

TestModel1-TestModel3

---

1 [at_treatment,area_team_treatment: sim=1.0, init=1.0, N=1.0, N1=1.0]

2 [FormInputData,FormData: sim=1.0, init=1.0, N=1.0, N1=1.0]

3 [AE,Accident And Emergency: sim=0.6180340557275542, init=1.0, N
=0.6180344478216819, N1=0.6180340557275542]

[varchar,varchar: sim=0.6180340557275542, init=1.0, N=0.6180344478216819, N1=0.6180340557275542]

[varchar,FormData: sim=3.8699690402476777E−4, init=1.0, N=0.0010131712259371832, N1=3.8699690402476777E−4]

[FormInputData,area_team_treatment: sim=3.8699690402476777E−4, init=1.0, N=0.0010131712259371832, N1=3.8699690402476777E−4]

[varchar,area_team_treatment: sim=3.8699690402476777E−4, init=1.0, N=0.0010131712259371832, N1=3.8699690402476777E−4]

[FormInputData,Accident And Emergency: sim=3.8699690402476777E−4, init=1.0, N=0.0010131712259371832, N1=3.8699690402476777E−4]

[at_treatment,Accident And Emergency: sim=3.8699690402476777E−4, init=1.0, N=0.0010131712259371832, N1=3.8699690402476777E−4]

[AE,varchar: sim=3.8699690402476777E−4, init=1.0, N=0.0010131712259371832, N1=3.8699690402476777E−4]

[AE,FormData: sim=3.8699690402476777E−4, init=1.0, N=0.0010131712259371832, N1=3.8699690402476777E−4]

[at_treatment,varchar: sim=3.8699690402476777E−4, init=1.0, N=0.0010131712259371832, N1=3.8699690402476777E−4]

[varchar,Accident And Emergency: sim=3.8699690402476777E−4, init=1.0, N=0.0010131712259371832, N1=3.8699690402476777E−4]

[at_treatment,FormData: sim=3.8699690402476777E−4, init=1.0, N=0.0010131712259371832, N1=3.8699690402476777E−4]

[AE,area_team_treatment: sim=3.8699690402476777E−4, init=1.0, N=0.0010131712259371832, N1=3.8699690402476777E−4]

[AE,Accident&Emergency: sim=1.0, init=1.0, N=1.0, N1=1.0]

[FormInputData,FormData: sim=0.8864362879215121, init=1.0, N=0.885298796974033, N1=0.8864362879215121]

[at_treatment,area_team_treatment: sim=0.8280061162141271, init=1.0, N=0.826570788289769, N1=0.8280061162141271]

[varchar,varchar: sim=0.7097895172108771, init=1.0, N=0.7084056205227244, N1=0.7097895172108771]

[FormInputData,AE_RE: sim=0.608954285919126, init=1.0, N=0.6099392877056049, N1=0.608954285919126]

[at_treatment,participant_id: sim=0.40857260568190945, init=1.0, N=0.40954663834869176, N1=0.40857260568190945]

[AE,TestModel3: sim=0.40715533436970025, init=1.0, N=0.4403653395706653, N1=0.40715533436970025]

[FormInputData,Test: sim=0.37639351380389235, init=1.0, N=0.40721857406119083, N1=0.37639351380389235]

[at_treatment,Test: sim=0.2365787218960365, init=1.0, N=0.2361174266384606, N1=0.2365787218960365]

[FormInputData,Accident&Emergency: sim=0.1557159035199282, init=1.0, N=0.16836558306789223, N1=0.1557159035199282]

[at_treatment,a−aminoadipic semialde, Urine: sim=0.09605078904749666, init=1.0, N=0.10393006220777992, N1=0.09605078904749666]

[at_treatment,A. fumigatus sp IgE Ab: sim=0.09605078904749666, init=1.0, N=0.10393006220777992, N1=0.09605078904749666]

[at_treatment,Abacavir ABC: sim=0.09605078904749666, init=1.0, N=0.10393006220777992, N1=0.09605078904749666]

[varchar,FormData: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[at_treatment,Accident&Emergency: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[FormInputData,area_team_treatment: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[varchar,Abacavir ABC: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[varchar,Accident&Emergency: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[varchar,area_team_treatment: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[AE,a−aminoadipic semialde, Urine: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[AE,A. fumigatus sp IgE Ab: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[FormInputData,TestModel3: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[at_treatment,TestModel3: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[AE,Test: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[AE,AE_RE: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[AE,participant_id: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[AE,varchar: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[FormInputData,a−aminoadipic semialde, Urine: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[FormInputData,A. fumigatus sp IgE Ab: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[AE,FormData: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[at_treatment,AE_RE: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[FormInputData,participant_id: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[FormInputData,varchar: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[at_treatment,varchar: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[varchar,TestModel3: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[at_treatment,FormData: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[AE,Abacavir ABC: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[varchar,a−aminoadipic semialde, Urine: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[varchar,A. fumigatus sp IgE Ab: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[varchar,Test: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[varchar,AE_RE: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

[varchar,participant_id: sim=3.2312836728789783E−7, init=1.0, N=8.062822082971129E−7, N1=3.2312836728789783E−7]

58 [FormInputData,Abacavir ABC: sim=3.2312836728789783E−7, init=1.0, N
=8.062822082971129E−7, N1=3.2312836728789783E−7]

59 [AE,area_team_treatment: sim=3.2312836728789783E−7, init=1.0, N
=8.062822082971129E−7, N1=3.2312836728789783E−7]

This yields similar results to the first comparison (for the corresponding branches of TestModel3 and TestModel1).

List of Figures

Glossary

ANTLR ANother Tool for Language Recognition 94

CDE Common Data Element 28

CDISC Clinical Data Interchange Standards Consortium 5, 29

CNS Central Nervous System 37

COSD Cancer Outcomes and Services Dataset 23, 27, 35, 36, 37, 38, 39, 40, 42, 44, 50

CT Clinical Terms 20, 21, 29, 49, 50, 132

DCM Detailed Clinical Models 28

DITA Darwin Information Typing Architecture system 46

DSL Domain Specific Language 15, 30

FCA Formal Concept Analysis 27

FHIR Fast Healthcare Interoperability Resources 20, 24, 28, 29, 56, 93

GPU Graphics Processing Unit 2

HETS Heterogeneous Tool Set 5

HL7 Health Level 7 23, 24, 28, 29

ICD International Classification of Diseases 20, 21, 22, 29

IDL Interface Definition Language 12

IEC International Electrotechnical Commission 4, 30, 31, 51, 63, 67

IR Information Retrieval 18, 19

ISO International Standards Organisation 4, 5, 7, 9, 23, 26, 27, 28, 29, 30, 31, 33, 34, 35, 38, 42, 43, 44, 45, 46, 47, 48, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 63, 67, 91, 92, 93, 108, 109, 110, 121, 137

LOINC Logical Observation Identifiers Names and Codes 22, 93

MDA Model-driven Architecture 10

MDML Metadata Management Language 46, 91, 94, 108, 124, 131, 132

MDR metadata registry 28, 30

MOF Meta Object Facility 10, 12, 13, 14, 15, 27

NCI National Cancer Institute (US) 26, 27

NDD NHS Data Dictionary 35, 44, 46, 48

NHS National Health Service 27, 35, 44

ODM Operational Data Model 5, 29

OMG Object Management Group 10, 12

OMOP Observational Medical Outcomes Partnership 93

OpenEHR Open Electronic Health Record 24, 29, 30

OWL Web Ontology Language as recommended by the W3C 4, 13, 21, 28, 35, 49, 55

QVT Query View Transformation 5

RDF Resource Description Framework 4, 13, 28

SKOS W3C Simple Knowledge Organization System 28

SNOMED Systematized Nomenclature of Medicine 20, 21, 22, 27, 29, 35, 49, 50, 131, 132

SPARQL SPARQL Protocol and RDF Query Language 28

UML Unified Modelling Language 5, 10, 11, 12, 13, 14, 27, 46, 47, 48

XMI XML Metadata Interchange 35, 46, 47

XML eXtensible Markup Language 36, 46

Bibliography

[1] Formal concept analysis.

[2] The web ontology language.

[3] Piers Akerman, 2014. Metadata references to crime prevention - see online (accessed 30 October 2020).

[4] Blagoj Atanasovski, Milos Bogdanovic, Goran Velinov, Leonid Stoimenov, Aleksandar S. Dimovski, Bojana Koteska, Dragan Jankovic, Irena Skrceska, Margita Kon-Popovska, and Boro Jakimovski. On defining a model driven architecture for an enterprise e-health system. *Enterprise Information Systems*, 12(8-9):915–941, 2018.

[5] Prafulla Bafna, Dhanya Pramod, and Anagha Vaidya. Document clustering: Tf-idf approach. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pages 61–66, 2016.

[6] Pablo F. Castro and Nazareno Aguirre. Algebraic foundations for specification refinements. formal methods: Foundations and applications. In *SBMF 2016. Lecture Notes in Computer Science*. Springer, 2016.

[7] Chong-wei Xu and J. Hughes. Realizing the open-closed principle. In *Fourth Annual ACIS International Conference on Computer and Information Science (ICIS'05)*, pages 274–279, 2005.

[8] Stéphane Clinchant and Eric Gaussier. Bridging language modeling and divergence from randomness approaches: a log-logistic model for ir. In *2nd International Conference on the Theory of Information Retrieval*, Cambridge, UK, 2009. `https://hal.inria.fr/hal-00953850`.

[9] Stéphane Clinchant and Eric Gaussier. Information-based models for ad hoc ir. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and*

Development in Information Retrieval, SIGIR '10, page 234–241, New York, NY, USA, 2010. Association for Computing Machinery.

[10] Bernardo Cuenca Grau, Bijan Parsia, and Evren Sirin. Working with multiple ontologies on the semantic web. *Lecture notes in computer science*, 2004.

[11] Weiqi Wei Harold R. Solbrig Cui Tao, Guoqian Jiang and Christopher G. Chute. Towards Semantic-Web Based Representation and Harmonization of Standard Meta-data Models for Clinical Studies. *AMIA Summits on Translational Science Proceedings*, 2011:59–63, 2011.

[12] Fred J. Damerau. A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7(3):171–176, March 1964. https://doi.org/10.1145/363958.363994.

[13] Jim Davies, James Welch, and Edward Crichton. Model-driven engineering of information systems: 10 years and 1000 versions. *Science of Computer Programming*, 2012. under review.

[14] Jim Davies, James Welch, David Milward, and Steve Harris. A formal, scalable approach to semantic interoperability. *Science of Computer Programming*, 192:102426, 2020. http://www.sciencedirect.com/science/article/pii/S016764232030037X.

[15] Andrea De Mauro, Marco Greco, and Michele Grimaldi. What is big data? a consensual definition and a review of key research topics. *AIP Conference Proceedings*, 1644(1):97–104, 2015. https://aip.scitation.org/doi/abs/10.1063/1.4907823.

[16] Ferdinand de Saussure and Wade Baskin. *Course in General Linguistics: Translated by Wade Baskin. Edited by Perry Meisel and Haun Saussy.* Columbia University Press, 2011. https://www.jstor.org/stable/10.7312/saus15726.

[17] Designed and run by a community of researchers from a variety of organizations. Metadata automation dream challenge, 2020. http://dreamchallenges.org/project-list/closed/.

[18] MHS Digital. Nhs data dictionary, 2008. https://www.datadictionary.nhs.uk/.

[19] Martin Dugas, Philipp Neuhaus, Alexandra Meidt, Justin Doods, Michael Storck, Philipp Bruland, and Julian Varghese. Portal of medical data models: information infrastructure for medical research and healthcare. *Database J. Biol. Databases Curation*, 2016, 2016.

[20] Eclipse. Ecore tools, 2008.

[21] Stanley E. Fish. How to do things with austin and searle: Speech act theory and literary criticism. *MLN*, 91(5):983–1025, 1976.

[22] Danyel Fisher, Rob DeLine, Mary Czerwinski, and Steven Drucker. Interactions with big data analytics. *Interactions*, 19(3):50–59, May 2012. https://doi.org/10.1145/2168931.2168943.

[23] Apache Software Foundation. Apache groovy programming language, 2021. https://groovy-lang.org/.

[24] Apache Software Foundation. Lucene search library, 2021. https://eclipse.org/Xtext/.

[25] Martin Fowler. *Domain Specific Languages*. Addison-Wesley Professional, 1st edition, 2010.

[26] Richard Gartner. *Metadata*. Springer International Publishing, 2016.

[27] Michael McGill Gerard Salton. *Introduction to Modern Information Retrieval McGraw Hill Book Company*. MJ McGill - New York, 1983.

[28] Artur Oliveira Gomes and Marcel Vinícius Medeiros Oliveira. Formal development of a cardiac pacemaker: From specification to code. In Jim Davies, Leila Silva, and Adenilso da Silva Simão, editors, *Formal Methods: Foundations and Applications - 13th Brazilian Symposium on Formal Methods, SBMF 2010, Natal, Brazil, November 8-11, 2010, Revised Selected Papers*, volume 6527 of *Lecture Notes in Computer Science*, pages 210–225. Springer, 2010.

[29] NIST Big Data Public Working Group. Big data interoperability framework, 2014. https://www.nist.gov/publications/nist-big-data-interoperability-framework-volume-1-definitions.

[30] Petra Duhm-Harbeck Jens K. Habermann Josef Ingenerf Hannes Ulrich, Ann-Kristin Kock. Metadata repository for improved data sharing and reuse based on hl7 fhir. *Studies in Health Technology and Informatics*, 228:162–166, 2016.

[31] Michi Henning. A new approach to object-oriented middleware. *IEEE Internet Computing*, 8(1):66–75, 2004.

[32] HL7-FHIR-Foundation. Fast health interoperability resources. *https://www.hl7.org/fhir//*, 2017.

[33] ISOJTC1. Io11179 international standard for metadata registries. *http://metadata-standards.org/11179//*, 2015.

[34] ISOTC215. Health informatics – electronic health record (ehr) standard. *http://www.en13606.org/*, 2008.

[35] Glenn Greenwald Spencer Ackerman Josh Taylor Mona Chalabi James Ball, Paul Farrell and Cory Doctorow. Metadata references available at
"https://www.theguardian.com/world/2020/feb/07/web-browsing-histories-are-being-give
"https://www.theguardian.com/australia-news/2014/oct/30/metadata-retention-what-will
"https://www.theguardian.com/commentisfree/2014/aug/06/explainer-what-is-metadata-sh
"https://www.theguardian.com/news/datablog/2013/may/14/metadata-rise-of-data-web-sea

[36] David Milward James Davies, Jeremy Gibbons and James Welch. Compositionality and refinement in model-driven engineering. *Formal Methods: Foundations and Applications SBMF 2012. Lecture Notes in Computer Science*, 2012.

[37] Wen-Shan Jian, Chien-Yeh Hsu, Te-Hui Hao, Hsyien-Chia Wen, Min-Huei Hsu, Yen-Liang Lee, Yu-Chuan Li, and Polun Chang. Building a portable data and information interoperability infrastructure framework for a standard taiwan electronic medical record template. *Computer Methods and Programs in Biomedicine*, 88(2):102 – 111, 2007.

[38] Chen-Wei Wang Jim Davie, David Milward and James Welch. Formal model-driven engineering of critical information systems. *Science of Computer Programming*, 103:88 – 113, 2015.

[39] Adam Milward David Milward Seyyed Shah Monika Solanki Jim Davies, Jeremy Gibbons and James Welch. Domain specific modelling for clinical research. In *Proceedings of the Workshop on Domain-Specific Modeling*, DSM 2015, page 1–8, New York, NY, USA, 2015. Association for Computing Machinery. https://doi.org/10.1145/2846696.2846701.

[40] Michael G. Hinchey Jonathan P. Bowen. *Industrial-Strength Formal Methods in Practice*. Springer, London, 2012.

[41] Spärck Jones Karen. Experiments in relevance weighting of search terms. *Information Processing & Management*, 15(3):133 – 144, 1979. http://www.sciencedirect.com/science/article/pii/0306457379900608.

[42] Spärck Jones Karen. Idf term weighting and ir research lessons. *Journal of Documentation*, 60(5):521–523, Jan 2004.

[43] İlker Kocabaş, Bekir Dinçer, and Bahar Karaoğlan. A nonparametric term weighting method for information retrieval based on measuring the divergence from independence. *Information Retrieval Journal*, 17(2):153 – 176, 2014. http://search.ebscohost.com/login.aspx.

[44] George A Komatsoulis, Denise B Warzel, Francis W Hartel, Krishnakant Shanbhag, Ram Chilukuri, Gilberto Fragoso, Sherri de Coronado, Dianne M Reeves, Jillaine B Hadfield, Christophe Ludet, et al. caCORE version 3: Implementation of a model driven, service-oriented architecture for semantic interoperability. *Journal of Biomedical Informatics*, 41(1):106–123, 2008.

[45] Isaac Kunz, Ming-Chin Lin, and Lewis Frey. Metadata mapping and reuse in caBIG. *BMC Bioinformatics*, 10(Suppl 2):S4, 2009.

[46] José M. Laguna, Miguel A.and Marqués and Yania Crespo. On the semantics of the extend relationship in use case models: Open-closed principle or clairvoyance? In Barbara Pernici, editor, *Advanced Information Systems Engineering*, pages 409–423, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[47] Moritz Lehne, Julian Sass, Andrea Essenwanger, Josef Schepers, and Sylvia Thun. Why digital medicine depends on interoperability. *npj Digital Medicine*, 2, 08 2019.

[48] Hugo Leroux, Simon McBride, Laurent Lefort, Madonna Kemp, and Simon Gibson. A method for the semantic enrichment of clinical trial data. In *Health Informatics: Building a Healthcare Future Through Trusted Information; Selected Papers from the 20th Australian National Health Informatics Conference (HIC 2012)*, volume 178, page 111. IOS Press, 2012.

[49] Vladimir Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707, February 1966. https://ui.adsabs.harvard.edu/abs/1966SPhD...10..707L.

[50] Craig Macdonald, Richard McCreadie, Rodrygo LT Santos, and Iadh Ounis. From puppy to maturity: Experiences in developing terrier. *Proc. of OSIR at SIGIR*, pages 60–63, 2012.

[51] Mar Marcos, Jose A. Maldonado, Begoa Martanez-Salvador, Diego Bosca, and Montserrat Robles. Interoperability of clinical decision-support systems and electronic health records using archetypes: A case study in clinical trial eligibility. *Journal of Biomedical Informatics*, 46(4):676 – 689, 2013.

[52] Catalina Martanez-Costa, Marcos Menarguez-Tortosa, and Jesualdo Tomas Fernandez-Breis. An approach for the semantic interoperability of iso en 13606 and openehr archetypes. *Journal of Biomedical Informatics*, 43(5):736 – 746, 2010.

[53] Catalina Martanez-Costa, Marcos Menarguez-Tortosa, and Jesualdo Tomas Fernandez-Breis. Clinical data interoperability based on archetype transformation. *Journal of Biomedical Informatics*, 44(5):869 – 880, 2011.

[54] Steven Mellor, Scott Kendall, Axel Uhr, and Dirk Weise. *MDA Distilled: Principles of Model-driven Architecture*. Addison Wesley, 2004.

[55] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the 18th International Conference on Data Engineering*, ICDE '02, page 117, USA, 2002. IEEE Computer Society.

[56] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice Hall, 2000.

[57] David Milward. Model driven data management in healthcare. In *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development*, MODELSWARD 2019, page 105–116, Setubal, PRT, 2019. SCITEPRESS - Science and Technology Publications, Lda. https://doi.org/10.5220/0007391101050116.

[58] David Milward. Dataset management using metadata. *Communications in Computer and Information Science*, 2020.

[59] Till Mossakowski. Heterogeneous specification and the heterogeneous tool set. In *Proceedings of CombLog'04*, page 129, 2005.

[60] Till Mossakowski, Christian Maeder, and Klaus Luttich. The heterogeneous tool set, hets. In *Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'07, page 519–522, Berlin, Heidelberg, 2007. Springer-Verlag.

[61] Michael Newcombe. How amazon web services uses formal methods. *Commun. ACM*, 58(4):66–73, March 2015. https://doi.org/10.1145/2699417.

[62] Sylvie M.N. Ngouongo, Matthias Löbe, and Jürgen Stausberg. The iso/iec 11179 norm for metadata registries: Does it cover healthcare standards in empirical research? *Journal of Biomedical Informatics*, 46(2):318 – 327, 2013. http://www.sciencedirect.com/science/article/pii/S1532046412001827.

[63] NHS. Cosd : Cancer outcomes and services dataset. http://www.datadictionary.nhs.uk/data_dictionary/messages/clinical_data_sets/data_sets/cancer_outcomes_and_services_data_set.

[64] Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Matching node embeddings for graph similarity. page 2429–2435, 2017.

[65] NIST. Trec: Text retrieval conference, 2020. https://trec.nist.gov/.

[66] OMG. Omg - meta object facility, v1.4, 2002. http://doc.omg.org/formal/2002-04-03.pdf.

[67] OMG. Semantic versioning, 2003. https://semver.org/.

[68] OMG. Omg unified modeling language, 2013. http://www.omg.org/spec/UML/2.5/Beta2/.

[69] OMG. Omg meta object facility (mof) core specification, 2014. http://www.omg.org/spec/MOF/2.4.2/PDF/.

[70] Terence Parr. Another tool for language recognition, 2020. https://www.antlr.org/.

[71] David Powers. Evaluation: From precision, recall and f-factor to roc, informedness, markedness and correlation. *Mach. Learn. Technol.*, 2, 01 2008.

[72] Gil Press. Cleaning big data: Most-time-consuming, least enjoyable data science task. *Forbes*, 2016.

[73] Alan Rector and Luigi Iannone. Lexically suggest, logically define: Quality assurance of the use of qualifiers and expected results of post-coordination in snomed ct. *Journal of Biomedical Informatics*, 45(2):199 – 209, 2012. http://www.sciencedirect.com/science/article/pii/S1532046411001687.

[74] Myriam Ribiere and Rose Dieng-Kuntz. A viewpoint model for cooperative building of an ontology. *Conceptual Structures:Integration and Interfaces*, 2002.

[75] Stephen Robertson and Spärck Jones Karen. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27(3):129–146, 1976. https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/asi.4630270302.

[76] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley Professional, 2004.

[77] Gerard Salton, C. S. Yang, and C. T. Yu. A theory of term importance in automatic text analysis. *Journal of the American Society for Information Science*, 1975.

[78] Hannes Schlieter, Martin Burwitz, Oliver Schonherr, and Martin Benedict. Towards model driven architecture in health care information system development. In *12th International Conference on Wirtschaftsinformatik (WI 2015)*, March 2015.

[79] A. Anil Sinaci and Gokce B. Laleci Erturkmen. A federated semantic metadata registry framework for enabling interoperability across clinical research and care domains. *Journal of Biomedical Informatics*, 46(5):784 – 794, 2013. http://www.sciencedirect.com/science/article/pii/S1532046413000750.

[80] Sara Sousa, Rita T.and Silva and Catia Pesquita. Evolving knowledge graph similarity for supervised learning in complex biomedical domains. *BMC Bioinformatics*, 21(1):6, Jan 2020.

[81] Sam Spencer. Aristotle metadata registry, 2020. https://www.aristotlemetadata.com/.

[82] Robert Stevens and Uli Sattler. Post-coordination: Making things up as you go along. http://ontogenesis.knowledgeblog.org/1305, 2013. http://ontogenesis.knowledgeblog.org/1305.

[83] Jeni Tennison, Kieron O'Hara, and Nigel Shadbolt. Apecks: using and evaluating a tool for ontology construction with internal and external ka support. *International Journal of Human-Computer Studies*, 56(4):375 – 422, 2002. http://www.sciencedirect.com/science/article/pii/S1071581902910001.

[84] Jenifer Tennison and Nigel Shadbolt. Apecks: a tool to support living ontologies, 1998. http://ksi.cpsc.ucalgary.ca/KAW/KAW98/tennison/.

[85] Health Data Research UK. Health data gateway, 2020. https://www.healthdatagateway.org.

[86] Ellen M. Voorhees. Trec: Continuing information retrieval's tradition ofexperimentation.

[87] W3C. Resource description framework. http://www.w3.org/RDF/. http://www.w3.org/RDF/.

[88] W3C. Data catalog vocabulary, 2020. https://www.w3.org/TR/vocab-dcat-2/.

[89] Robert A. Wagner and Roy Lowrance. An extension of the string-to-string correction problem. *J. ACM*, 22(2):177–183, April 1975. https://doi.org/10.1145/321879.321880.

[90] Jim Woodcock and Jim Davies. *Using Z*. Prentice Hall, 1996. www.usingz.com.

[91] Zhang Yun-tao, Gong Ling, and Wang Yong-cheng. An improved tf-idf approach for text classification. *Journal of Zhejiang University-SCIENCE A*, 6(1):49–55, 2005. Project (No. 60082003) supported by the National Natural Science Foundation of China.

[92] Laura A. Zager and George C. Verghese. Graph similarity scoring and matching. *Applied Mathematics Letters*, 21(1):86 – 94, 2008. http://www.sciencedirect.com/science/article/pii/S0893965907001012.