Model Driven Engineering for Large-Scale Clinical Research

Jim Davies, Jeremy Gibbons, David Milward, Seyyed Shah, Monika Solanki, and James Welch
Department of Computer Science, University of Oxford
Email: Firstname.Lastname@cs.ox.ac.uk

***Abstract*—Scientific progress in medicine is increasingly dependent upon the large-scale integration of data from a range of sources. This data is acquired in different contexts, held in different systems, and subject to different processes and constraints. The value of the integrated data relies upon common data points having a consistent interpretation in each case; to achieve this at scale requires appropriate informatics support. This paper explains how a model-driven approach to software engineering and data management, in which software artefacts are generated automatically from data models, can be used to achieve this. It introduces a simple data modelling language, consistent with standard object modelling notations, together with a set of tools for model creation, maintenance, and deployment. It reports upon the application of this approach in the provision of informatics support for two large-scale clinical research initiatives.**

I. Introduction

Scientific progress in medicine consists principally in the introduction of new diagnostics and new therapeutics: new ways of figuring out what is wrong, and new ways of doing something about it. The causes of disease, the expression of symptoms, and the response to treatment, can be extremely complex, involving many levels of biology—genomics, proteomics, metabolomics—and a wide variety of lifestyle and environmental factors.

To obtain the evidence to support the introduction of a new diagnostic or therapeutic, while taking account of the variations in biology and environment, we need to make detailed, comparable observations of a suitably large number of individuals. These observations will be made by different people, under different circumstances, and stored and processed in different information systems. A significant degree of coordination is required to achieve an adequate combination of detail and comparability.

One means of achieving this involves prior agreement upon a prescribed dataset: a precise specification of the observations needed to answer a particular question. An information system is then implemented or configured to match this dataset, and staff are trained to acquire data to match the prescribed interpretation. This can be time-consuming and expensive. Furthermore, the data acquired is unlikely to be re-used: not only is the dataset as a whole tailored towards a particular question, but the same may be true of the individual observations.

It should be clear that this is unnecessary. A dataset can be composed by drawing upon, and adding to, a library of existing data definitions. As a precise specification, it can be used as the basis for the automatic generation or configuration of an information system for data capture, reducing costs while enhancing the value of the data obtained. The enhanced value comes in part from the fact that some of the data has been recorded against existing definitions, but also from the fact that definitions are readily available—in a re-usable, computable form—for all of the data collected.

Costs can be further reduced, and scientific progress accelerated, by re-using data already collected: in other scientific studies; in the course of care delivery; or from other sources, including new sensor technologies. Precise data definitions are important here also: we need to know whether the data already collected is fit for purpose, whether its interpretation is consistent with the analysis that we intend to perform. In most cases, these definitions will be created retrospectively, by progressively adding information about the context of collection, and any subsequent processing, until the interpretation of the data is clear.

Having created precise definitions for existing data, we can enhance the value of the data, and reduce the costs of research, by making these definitions available in a standardised, computable form. In designing and proposing a new clinical study, researchers can take account not only of what questions have been asked before, but also of what data already exists, in other research databases or in clinical information systems. In this way, we can eliminate unhelpful variation in study design and unnecessary duplication of effort in data collection.

Furthermore, if we have both precise definitions of existing data, and precise definitions of data requirements for a study, then we can provide automatic support for aspects of study design and delivery: definitions can be compared, queries can be generated for data extraction, and—if summary metadata on the contents of existing databases is available—the feasibility of a particular study can be assessed in advance.

Scientific progress in medicine is increasingly dependent upon this kind of automatic support for data management and integration. Without it, we will be unable to assemble the evidence needed—detailed, comparable data on thousands or millions of individuals—to produce new insights, to validate new discoveries, and to support the introduction of new diagnostics and therapeutics into clinical practice.

This paper explains how a model-driven approach to software and data engineering can provide the support required for large-scale clinical research. It introduces a simple data modelling language, consistent with standard object modelling notations, together with a set of tools for model creation, maintenance, and deployment. It then reports upon the experience of applying these tools within two large-scale clinical research initiatives.

One of these initiatives involves the re-use of data captured in existing clinical information systems. In the area of translational research, in which new innovations are developed and evaluated in the context of clinical practice ('from bench to bedside'), the data needed to support the science is often the same as that needed to support high-quality care delivery and service improvement. In existing clinical systems, however, the same observation may be recorded in many different ways, and there are significant challenges.

The other involves the development of new information systems for data acquisition and management, as well as the customisation and re-use of existing systems wherever possible. Here, the emphasis is upon defining new, generic datasets, in specific therapeutic areas, with the aim of supporting a wide range of studies using the data collected. A particular challenge for informatics development and data re-use stems from the constant updating of these datasets to take account of new knowledge, new constraints, and new objectives.

Together, these initiatives provide an initial validation of the approach. The software tools are in use by clinical researchers, rather than the software engineers responsible for their design. Data is being collected to the definitions that they have produced, using software that they have generated. It is too early to undertake any quantitative, comparative evaluation; however, it is our hope that a report of the approach taken, and the experience gained thus far, will be useful to those working in the field of automated software engineering.

II. The Models Catalogue Language and Architecture

A. Models Catalogue

A software *model* is an abstraction of a software program, which will be instantiated and run using *real data*, software modelling is becoming important for a number of reasons, not least that it enables developers the opportunity to *re-use* code. The Unified Modelling Language UML [**?**] or Ecore [**?**] (a subset of UML) are in widespread use for defining and representing software *models*. UML and Ecore are defined at a level which is *more abstract* than the model itself, and one which we term the *meta-modelling* layer.

The data we have been dealing with in these projects is mostly in the form of textual forms, sometimes in word or PDF format, sometimes in a more specialised form of XML or Excel. The data therefore does have a clear structure, although the clinician running the trial very often has little visibility, knowledge or appreciation of how the data is stored, and who else might in future want to access or manipulate that data. Although the healthcare providers have work-flows which the patients and doctors follow in the treatment cycle, these are not being considered in this paper.

Sometimes there are standards available governing aspects of the data in question, for instance reporting of a particular clinical trial may need to be in XML which conforms to a particular XSD, however that doesn't mean that the data has been collected with that particular standard in mind. For the most part clinicians are interested in fairly static data sets, a patient record, the results of a treatment, a clinical trial form, the dynamic aspects are less important, at any rate in terms of the computational representation. What is important is the meaning of the data once it is captured, and in particular how it relates to other data in the same field, for instance is *tumour weight* in one experiment the same as *tumour weight* in another. These two terms may be used in different contexts, for instance prostate cancer and liver cancer, and they may be represented by different values domain in different hospital trusts, for instance one may use grams the other kilograms, the values may be comparable but an adjustment will need to be made to make that comparison.

Thus there are two things we need to identify and reference in building the toolkit, firstly does this *concept* **mean** the same as that *concept*, and secondly does this *concept* have the same *representation* as that concept. The way in which we can match up concepts in software engineering is by associated them with classes of software objects, and then defining methods which can reference them.

B. ISO11179

ISO11179 is the international standard relating to metadata and in particular metadata registries. It forms the basis of the design of the models catalogue toolkit.

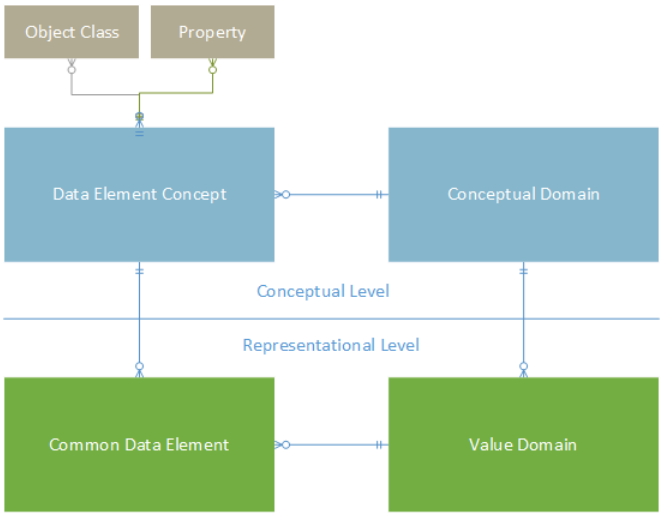

Fig. 1. Core model for ISO11179 Metadata Registry

The ISO11179 standard uses the notion of a *data element concept*, *a data element*, *a value domain*, and a *conceptual domain*. The standard currently confines itself to the detailed level of concepts and data elements and has no notion of

collections of data elements or data element concepts, but instead attaches two attributes: an *object class* and a *property* to each data element concept and these attributes allow the data element concept's to be aggregated or classified. This core model of the ISO11179 is illustrated in figure 1. The data element concept and conceptual domain entities belong to the *conceptual level* whilst the common data element and value domain both belong to the *representational level*.

For example an Integer data-type in a programming language may be used to represent inches in a measurement program, it may also be used to count vehicles in a logistics application. A data element is said to be comprised of a data element concept(DEC) which is its meaning and a value domain(VD) which is its representation.

Entity	ISO Definition	ISO11179 Implementation Guidelines
Data Element Concept(DEC)	An idea that can be represented in the form of a data element, described independently of any particular representation.	A concept that can be represented in the form of a Data Element, described independently of any particular representation.
Common Data Element(CDE)	A unit of data for which the definition, identification, representation, and permissible values are specified by means of a set of attributes.	A unit of data for which the definition, identification, representation and Permissible Values are specified by means of a set of attributes.
Value Domain (VD)	The description of a value meaning.	A description of a Value Meaning.
Conceptual Domain (CD)	A set of valid value meanings, which may be enumerated or expressed via a description.	A set of valid Value Meanings.

The table lists the definitions of the key entities used in the standard Conceptual domains comprise sets of value domains, they provide a collection mechanism of *Value Meanings* which provide representation for a particular data element concepts. Data Element Concepts can then be grouped according to their Object Class, or Property, however whilst this works for a system that is focussed entirely on the metadata units, any working software system will need to group the data elements in a structure that is easily transformed into the components such as Classes and Entities that are used in most information systems.

C. A Metadata Language and Abstract Architecture

The ISO11179 specification illustrates different aspects of the standard using UML class diagrams, and we have used these to inform the development of a very simple domain specific language for metadata, which we are calling the *Language for Enterprise Metadata Modelling LEMM*. The next section gives a description of the language and highlights how it differs from the meta-model outlined in the standard. Probably the key changes are that we have added in containers to handle data element collections, calling these *DataClasses* and to handle collections of these *DataClasses* which we have introduced entities called *DataModels*.

A simplified overview model, without attributes and methods, showing the Ecore model for the LEMM DSL is shown in Figure 2.

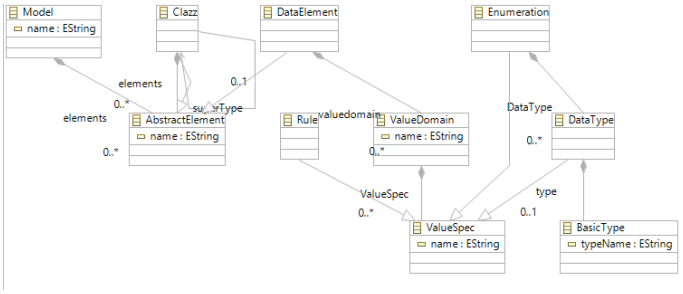

Fig. 2. Overview of LEMM in Ecore

1) DataModel: A datamodel is a grouping or containment entity which groups a set of *DataClasses* together. DataModels can be thought of as datasets, or even database schemas, very often in the medical domain they are defined either by XML Schema definition files, or by equivalent schemas written in Excel. DataModels are collections of *ConceptElements* which in turn can be either *Data Elements* or *Classes*. There is no real notion of composition or multiplicity, a instance of a DataModel can contain an instance of a Data Element or not as required by the instance. DataModels are named, have a description and have a version identity.

2) DataClass: A DataClass is a grouping or collection of *attributes* which can be data elements or classes, the attributes are currently *mandatory*, so that DataClass with 5 attributes must have those 5 attributes instantiated in an instance for it to be considered of that DataClass. A dataclass is named to differentiate it from the term *Class* as used in object oriented programming languages, it captures the structural rather than behavioural aspects of a class. DataClasses represent *Concepts*, and can be *Generalized* into a hierarchy, giving some of the benefits of inheritance to the language.

D. Data Elements

Data Elements can also represent *Concepts* and are by their nature *atomic*. Each data element is related to a value domain on a one-to-one basis, and the relationship is a two-way relationship.

E. Value Domain

A Value Domain is the domain in which the data element is represented, it can consist of one or more *ValueSpecs*.

F. ValueSpec

A *ValueSpec* can be a simple datatype, an enumeration of datatypes, or a rule - such as a regular expression - which defines the way in which a series of characters is formed into a string attribute.

Listing 1. Language for Enterprise Modelling

```
grammar uk.ox.ac.cs.elm.ElmDSL with org.eclipse.xt
```

```
generate elmDSL "http://cs.ox.ox.uk/elm/ElmDSL"
DataModel:
        'datamodel' name = ID '{'
        (elements += ConceptElement)*
         '}'
         (extensions += Extension)*
;
ConceptElement:
        DataClass | DataElement
;
QualifiedName:
        ID ('.' ID)*
;
DataClass:
  'dataclass' name = ID ('extends' superType
        (elements += ConceptElement)*
        (extensions += Extension)*
         '}'
;
DataElement:
  'dataelement' name = ID
  '{'
     (valuedomain += ValueDomain)
     (extensions += Extension)*
  '}'
;
ValueDomain:
  'valuedomain' name = ID
  ValueSpec += ValueSpec*
  (extensions += Extension)*
;
ValueSpec:
  '{'
      (Rule | Enumeration | DataType)
  '}'
;
Rule:
  'rule' name = ID
;
Enumeration:
  'enumeration' name = ID
  '{'
   DataType += DataType*
   '}'
;
DataType:
  'datatype' name=STRING ':' type=BasicType ';'
;
BasicType:
  typeName=('string'|'int'|'boolean'|'date')
  ;

Extension:
        'extension' key=ID ':' value=STRING
;
ConceptDomain:
        'conceptdomain' name=ID '{'
             (concepts += ValueDomain)*
        '}'
;
```

¡¡¡¡¡¡¡ Updated upstream The ElmDSL is used to represent a model for Cancer data (part of the Cancer Outcomes and Services Dataset [**?**]COSD)) in Figure 3

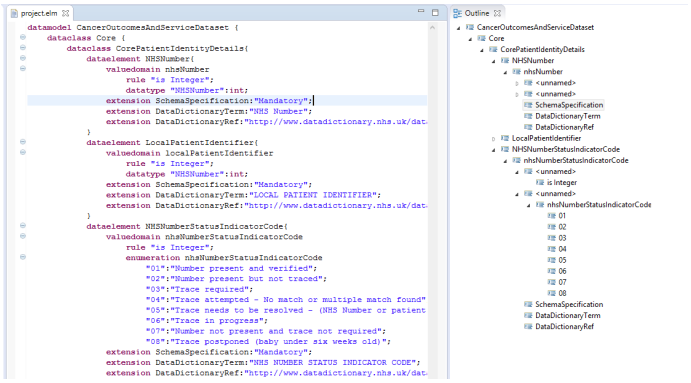

Fig. 3. COSD data in LEMM Model

The original dataset was collected and managed in Excel, as shown in Figure 4

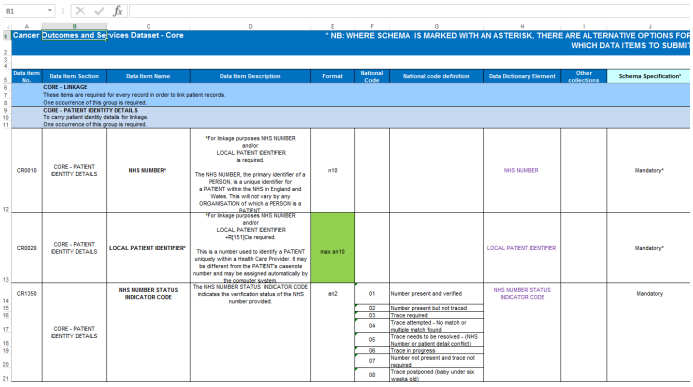

Fig. 4. COSD data in Excel Format

======= ¿¿¿¿¿¿¿ Stashed changes

III. Implementation

Groovy was chosen as a language for implementation for two reasons, firstly it has a very efficient web framework called Grails built on the Spring framework, which is not only proven to be very robust and scalable, but is also relatively easy to implement and so enables quick agile development cycles. Previous implementations using Java/Spring and Java/Roo have proved very time-consuming to experiment with, whereas Grails has proven to be more flexible and easier to experiment with. The Groovy language also offers both some functional capability, as well as dynamic meta-programming capability. Domain specific languages (DSLs) can be easily built on this framework, and this capability offers some scope to build a

DSL based on the DataMOF DSL specification and meta-model expanded in the previous section.

The Model Catalogue design discussed here has been implemented by building a core Models Catalogue using Groovy within the Grails framework. Prototype implementation was carried out using the EMF/ECore framework, however the Grails framework offered a robust web-based alternative, with enough Model-Driven capability to test out the core ideas expressed here using a maintainable java-based stack that could be worked on by mainstream developers without a detailed knowledge of the Model Driven Engineering.

The front end user interface was implemented using a combination of HTML with Javascript and CSS, the principal framework used being Angular JS. Communication with the client was carried out using a REST controller, enabling a variety of clients potentially to link up with the Model Catalogue.

GORM was used as a persistence mechanism, with a MySQL relational database as storage, although different GORM adapters made it possible to attach NO SQL datastores such as Neo4J and MongoDB. The full architectural stack is shown in figure5

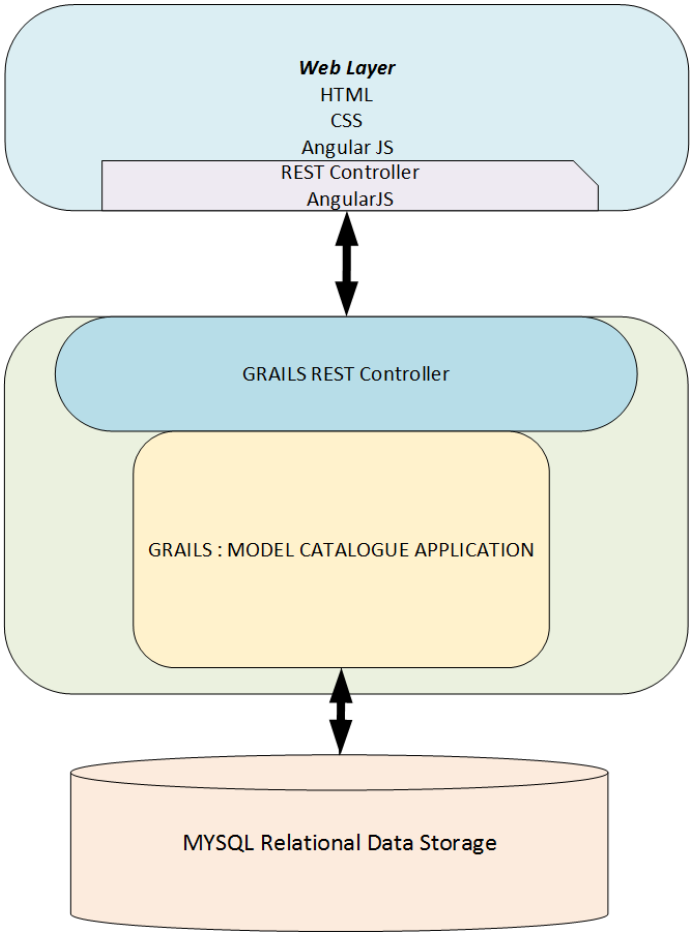

Fig. 5. Overview Architecture

The Grails/GORM framework enabled the Ecore model to generate the basic Grails Domain model, and from that a Groovy DSL was built to handle transformations internally between different representational languages such as XML and excel. A series of importers was built, although due to the simplicity of the DataMOF DSL it was not possible to build exporters for every format. The internal domain model used a basic *Catalogue Element* which was able to link elements via the *relationship* and *relationshipType* classes. Any catalogue element is able to be related to any other catalogue element through a relationship class, this relationship is constrained by the relationshipType object which can prevent different catalogue element types being related, so that a Model cannot directly be related to a say a Datatype Enumeration. Relationship types can be added to the Model dynamically, so that even though the relationship between a Model and Datatype enumeration is prevented initially, a new type could be introduced by an administrator or super user to add in that relationship. Some of the main relationships that are currently modelled in the *Models Catalogue* are as follows:

Source	Relationship	Destination
Model	containment	DataElement
Model	containment	Class
Model	hierarchical	Model
DataElement	supersession	DataElement

The Core architecture can seen in figure5

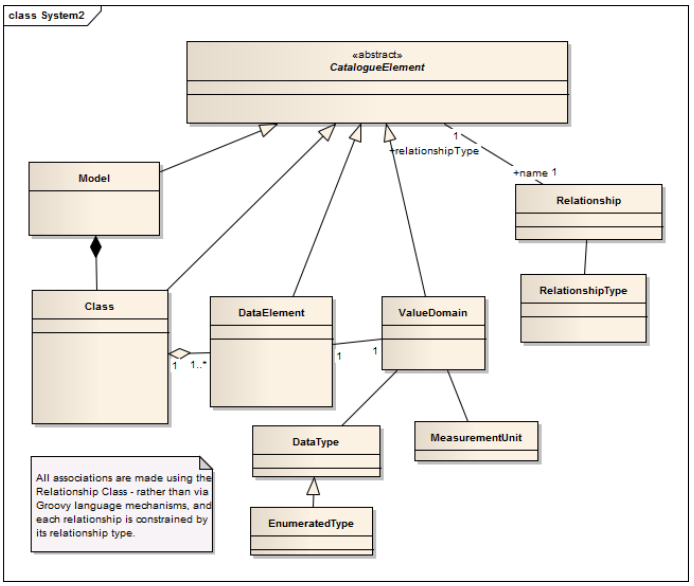

Fig. 6. Overview Architecture

A. Modelling Overview

IV. Experience

Key ideas:

- Data Elements can be used across different DataModels - favourites or shopping cart.
- Unique Identification of DataElements

- Documentation can be automatically generated for review purposes
- Consistency checking carried out on data elements
- Generation of XSDs, XML, and Forms as output, interfacing with OpenClinica.
- Data Provenance

A. NHIC

B. Genomics England - reuse of HIC Models

C. Genomics England - deployment

V. Conclusion

A. Ongoing work