Paper AD09 – PHUSE EU Connect 20204

End-to-end automation using a model-driven development workbench

Stuart Malcolm, Veramed, UK

ABSTRACT

To achieve the CDISC 360 vision of end-to-end standards-based automation requires a paradigm-shift in the way we work with metadata.

This paper introduces model-driven development (MDD) as an agile approach to the creation, management and processing of standards-based clinical trial metadata, and presents examples of how MDD can be used for automation using CDISC Standards.

INTRODUCTION

Model-driven development (MDD) is a long-established software engineering approach that prioritizes the use of abstract models to design systems. These models, which represent the system's structure, behavior, and functionality, serve as the primary artifacts throughout the development lifecycle.

By focusing on high-level models rather than programming code, MDD aims to enhance productivity, improve communication among stakeholders, and ensure that deliverables align with requirements. Additionally, MDD facilitates automation of editor and code generation, reducing manual coding efforts and minimizing errors, ultimately leading to more adaptable and maintainable systems.

ALIGNMENT WITH CDISC 360 AND 360-I INITIATIVE

> *"360 is the CDISC initiative aimed at implementing standards as linked metadata with a conceptual foundation providing the additional semantics needed to support metadata driven-automation across the end-to-end clinical research data lifecycle.*
>
> *New software tools will be able to consume this new metadata to ease standards implementations while increasing data processing efficiencies.*
>
> *Standards-based metadata-driven automation is a critical component in realizing the primary benefits expected of the CDISC standards: substantially improved efficiency, consistency, and re-usability across the clinical research data lifecycle."*

Figure 1 CDISC 360 Vision Statement

The CDISC 360i project is defined as an implementation project to follow the initial CDISC 360 proof-of-concept project, with the following goals [1]:

- Define end-to-end standards
  - Digitalize information from protocol to reporting
  - Link concepts to representation standards
  - Enrich with transformation & derivation logic
- Study design & build
  - Select concept and concept groups in digital Schedule of Activities
  - Automates study builds (forms, SDTM specs, ADaM specs, TFL specs…)
  - Provides derivations & transformation algorithms
- Automate data flow
  - Demonstrate end-to-end automation (e.g. from SoA to Concept Groups)
  - Automate transformations & derivations between data states

MDD aligns well with the goals of the CDISC 360 project due to its emphasis on automation, standardization, and efficiency. CDISC 360 aims to implement standards as linked metadata to support metadata-driven automation across the clinical research data lifecycle. MDD supports this by using high-level models to define and automate data flows.

MDD enables the automatic generation of code and other artifacts from models, minimizing manual intervention and reducing errors. And promotes consistency and reusability, which are key aspects of CDISC 360's vision.

In addition, by using standardized models, MDD ensures consistency across different studies and phases of clinical trials, and model-to-model transformation to ensure interoperability and integration of data across the clinical research lifecycle.

WHAT IS A MODEL?

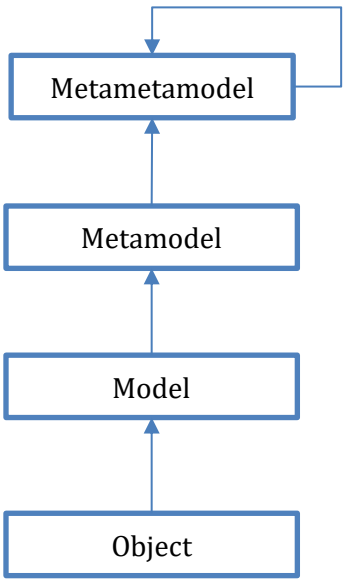
In the context of model-driven development, we can define a model as:

- An abstraction of things in the real world that incorporates both behavior and structure
- It aims to express the meaning of terms and concepts used by domain experts to discuss the problem
- ..and to find the correct relationships between different concepts
- ..is described using a defined notation which is machine-readable

This definition is written in the English language – which is ok, however it is open to interpretation. One solution to this possible ambiguity would be to precisely define the terms we are using to define a model. However the structure and meaning of this model definition would, in turn, need a definition!

Fortunately, this work has already been done – and there is an established formal definition of metamodeling [2] and also associated standard modelling languages – so called unified modelling language (UML) [3]

Metamodeling involves four primary levels of abstraction:



M3 – Meta-Meta-Model Level

This is the highest level of abstraction, defining the language and rules for creating meta-models at the M2 level. The Meta-Object Facility (MOF) is an example, providing the framework for defining meta-models like UML

M2 – Meta-Model Level

Meta-models define the language and rules used to create the models at the M1 level. An example is the UML meta-model, which specifies the elements and relationships that can be used in UML diagrams.

M1 – Model Level

This level consists of models that describe the structure and behavior of the instances at the M0 level. For instance, the Variables and Attributes of an SDTM dataset, etc.

M0 – Instance / Object

This is the most concrete level, representing real-world instances or objects.

Figure 2 Levels of abstraction used in meta-modeling

As an example of these levels of abstraction, lets re-examine the definition of a model (defined above in English), which can also be drawn graphically as shown below:

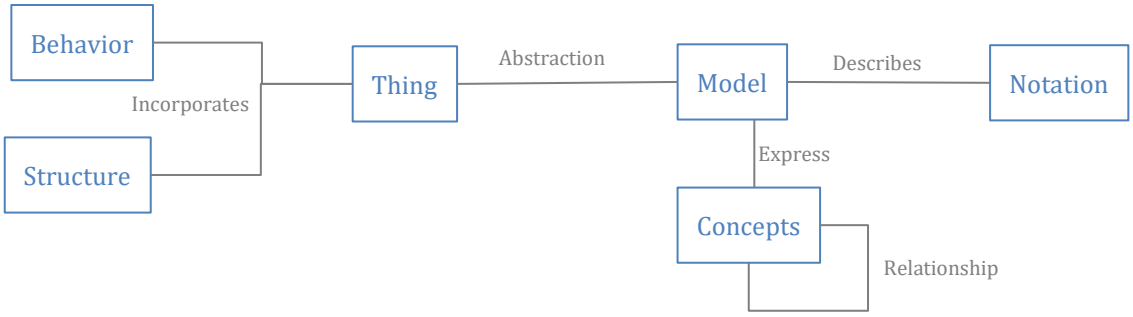


Figure 3 Model of 'model' previously described in English

This definition of a Model is the M1 definition – i.e. it describes the structure of a model, it is arguable cleaner in that it does not contain the 'padding' of the English-language definition, however we do not (yet) have any definition of what the boxes and lines mean.

So, lets define what the above diagram is – lets call it an entity-relationship diagram – or ER Diagram for short:

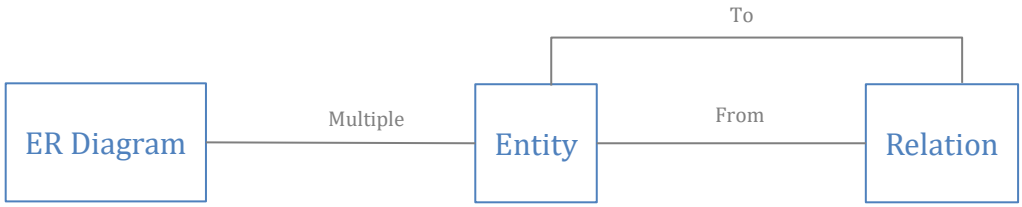


Figure 4 Meta-Model of an ER-Diagram used to define a 'model'

Which, in English-Language would be that an ER Diagram contains multiple Entity, each of which can have a relationship from one Entity to another.
Note that with this definition, we could define many other entity-relationships other than our original 'model' in Figure 3 above. Also, there are many other types of diagrams that we might want to use to define models – for example, state-transitions diagrams or sequence diagrams to define behavior of a model, etc.
So, we need a meta-meta-model (M3) which defines the schema of our M2 ER Diagram, for example:

Figure 5 Meta-Meta mode used to define an ER-Diagram

The meta-model-model shown above is a simplified version of a common meta-meta-model design pattern. A complete definition is the Object Management Group (OMG) Meta-Object Facility (MOF) [9] which is defined as ISO Standard 19508:2014

MACHINE-READABLE MODELS

The advantage of this model-driven approach is that the M3 meta-meta-model can be defined in terms of programming data-structures and operations. For example:

- The Unified Modeling Language (UML) [3] is a standardized visual modeling language used to specify, visualize, construct, and document the artifacts of software systems, as well as for business modeling and other non-software systems
- JSON Schema [5] is a standard that defines the structure, content, and data types of JSON objects to ensure data consistency and validation.
- Langium [6] is an open-source language engineering tool with a meta-model which is a grammar for defining models as domain-specific text languages.
- LinkML [7] is a modeling language that allows you to author schemas ("models") in YAML that describe the structure of data.
- Eclipse Modelling Framework (EMF) [8] is a modelling framework and code generation facility for building tools and applications based on structured data model.

The key thing in common with all model-driven approaches is a “software-based” meta-meta-model which has two important benefits:

- Model-to-model transformations
- Model-to-text transformations

Model-to-model (M2M) transformations involve converting one model into another within the same or different modeling languages. This means that once a model conversion has been defined, it can be reused for any source model without having to create a mapping for each conversion. Typical examples of M2M include:

- OMOP to SDTM conversion
- USDM to SDTM Trial Design domains
- ADaM to Define.XML conversion

Model-to-text (M2T) transformations involve converting models into textual representations, such as source code, documentation, or configuration files. This process is crucial in model-driven engineering, as it automates the generation of various textual artifacts from underlying models.

MODELLING WORKBENCH

A modelling workbench is a core tool in the model-driven development workflow. It is an integrated development environment that is designed to support the creation, manipulation, and analysis of models. It typically includes tools for designing, simulation and validation of models, along with code and documentation generation capabilities.

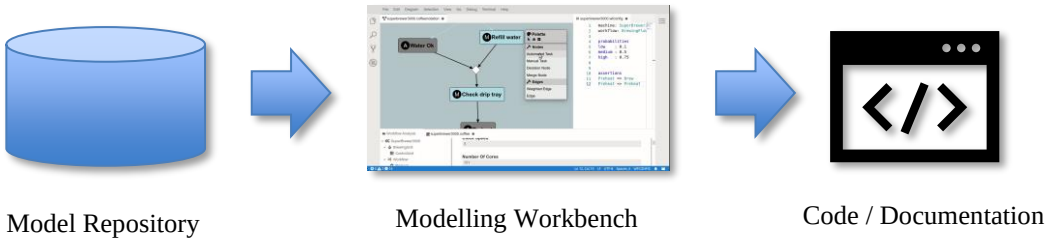


Figure 6 Modelling Workbench integrated development environment

A modelling workbench is created by a developer who uses M2 models to create editors and generators that are built into the workbench, and the workbench is used to edit M1 models, and generate code, documentation, etc. from those models.

USAGE SCENARIO

So, for example, a tool developer might create SDTM meta-models and then generate SDTM Editor and generator that is built into the workbench.

Then, a clinical programmer can use the workbench to create/edit the SDTM domains (what we traditionally call the SDTM Metadata – variables, types, etc.) and then generate SDTM programs, and define.xml from the SDTM model

MODELLING CDISC STANDARDS

Let’s use the SDTM v2.1 standard [10] as an example. Section 2 of the standard defines SDTM key concepts of domains and observations:

“The model is built around the concept of *observations*, which consist of discrete pieces of information collected during a study. Observations normally correspond to rows in a dataset. A *domain* is a collection of observations on a particular topic”

Then section 2.1 further defines observations in terms of variables and different types of variables:

“Each observation consists of a series of named *variables*. Each variable, which normally corresponds to a column in a dataset, can be classified according to its *role*. A role describes the type of information conveyed by the variable about each distinct observation and how it can be used.”

The standard then goes on to define the 5 major roles: Identifier, Topic, Timing, Qualifier and Rule.

If we use UML as our M3 meta-meta-model language, then we can define the SDTM Meta-Model as:

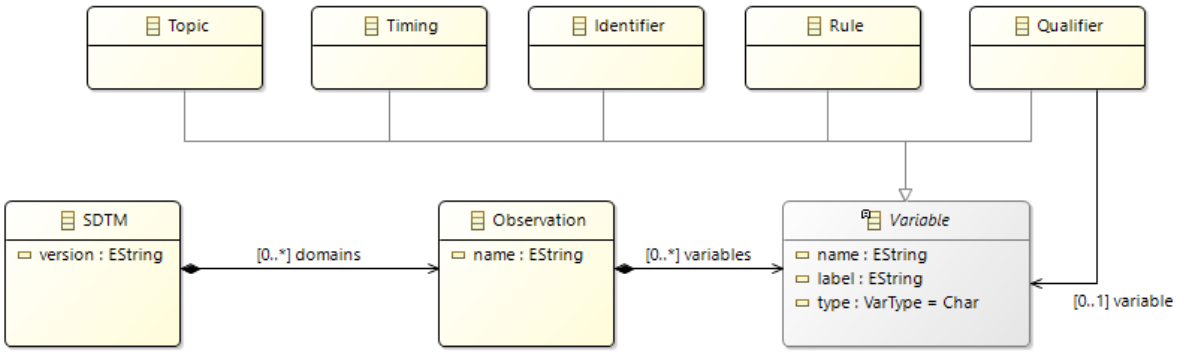


Figure 7 UML meta-model for SDTM v2.1

This meta-model contains enough information to be able to automate the generation of an editor that can be included in a modelling workbench to allow users to create SDTM models. To di this we need to be able to generate a Model Editor from the meta-model.

There are various types of Model-Editor that we want create, for example:

- Tree-Editor. These provide a tree similar to a folder-structure in a file browser, along with forms to edit the body of the model.
- Language-Editor. This is a textual representation of the model – similar to programming code, including syntax highlighting, code-completion, etc.
- Graphical-Editor. This provides a visual representation of the model – for example dependencies between datasets as a node graph, or study design as a flow chart, etc.
- Table-Editor. This is a tabular representation of a model – similar to good-old Excel! But with syntax and semantic rules bult into the editor.

Note that the editors are read-write views onto the model, so a workbench may allow multiple different editors for the same model at the same time.

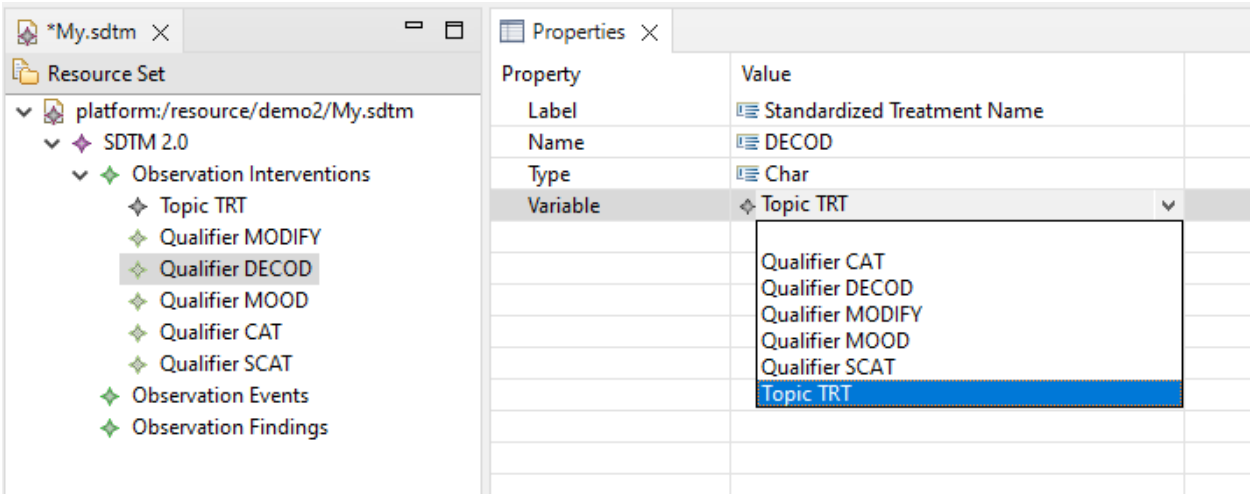


Figure 8 Tree-Editor for SDTM Model

Figure 8 above shows an example of a Tree-Editor for the SDTM model shown in Figure 7 – note that the DECOD Qualifier variable selected in the left-hand tree is being edited in the right-hand form. Because the model defines the qualifier variables have a relationship to other variables, the drop-down box is pre-populated with other variables to select.
This editor can be generated directly from the meta-model shown in Figure 7

While a Tree-Editor is useful for editing existing models, it might be more efficient to allow users to create new models using a text editor and a language that is closer to "programming code" – that allows copy/paste and supports syntax highlighting and code-completion. An alternative SDTM Language-Editor is shown below:

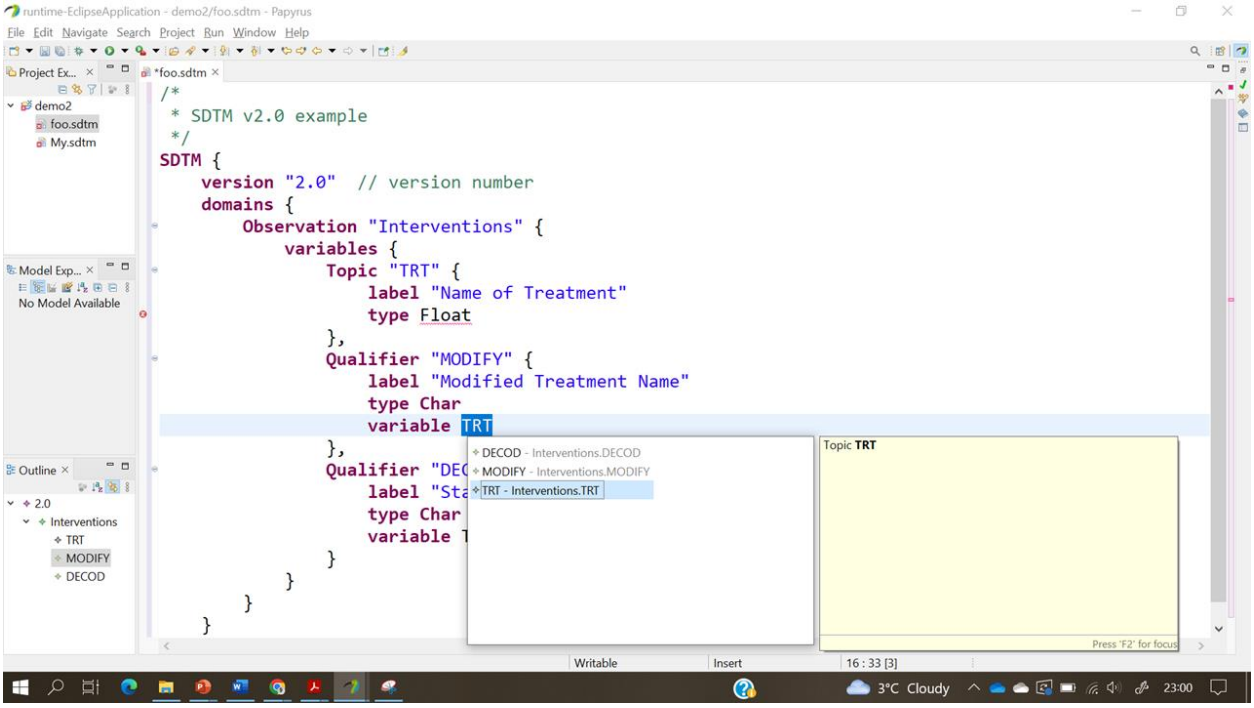


Figure 9 SDTM Language-Editor

Note that the language editor is a different view onto the same model as the tree editor was editing. The language editor

CODE GENERATION

Model-to-text (M2T) supports the automation of key deliverables required for clinical trial analyses – for example generation of programs (for e.g. creation of SDTM, ADaM or TFL deliverables) or supporting metadata such as Define.XML files, documentation such as reviewer guides (SDRG or ADRG)

Model-driven code generation techniques are detailed in the companion paper presented at PHUSE EU Connect 2023 [11]

REFERENCES

[1] PHUSE October 2024 Webinar Wednesday, CDISC Vision and Roadmap: Update on CDISC's Mission and 360i Plan – Chris Decker, CDISC CEO.
[2] Metamodelling Wikipedia article. https://en.wikipedia.org/wiki/Metamodeling
[3] Unified Modelling Language. https://www.uml.org/
[4] An overview of the UML Meta-Model. https://www.cs.sjsu.edu/~pearce/modules/lectures/uml2/index.htm
[5] JSON Schema. https://json-schema.org/specification
[6] Langium Grammar Language. https://langium.org/docs/reference/grammar-language/
[7] LinkML Metamodel. https://linkml.io/linkml/schemas/models.html#metamodel
[8] ISO 19508:2014 Object Management Group Meta Object Facility (MOF) Core. https://www.omg.org/spec/MOF/ISO/19508/PDF/
[9] Object Management Group Meta-Object Facility Specification. https://www.omg.org/mof/
[10] CDISC Study Data Tabulation Model v2.1 https://www.cdisc.org/standards/foundational/sdtm/sdtm-v2-1
[11] Code Generation: Turning Metadata into SAS, R and Python. PHUSE EU Connect 2023. Whitepaper: https://www.lexjansen.com/phuse/2023/ad/PAP_AD17.pdf slides: https://phuse.s3.eu-central-1.amazonaws.com/Archive/2023/Connect/EU/Birmingham/PRE_AD17.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Author Name: Stuart Malcolm
Company: Veramed
Email: stuart.malcolm@veramed.co.uk
LinkedIn: https://www.linkedin.com/in/stuart-malcolm/