phuse·connect
connect·share·advance
5-8 November
ICC Birmingham
EU 2023
The Clinical Data Science Conference

Paper DH12

Automating ADaM Dataset Generation via Definitive Specification Writing with A Domain-Specific Language

Kang Zhao, AstraZeneca, Shanghai, China

ABSTRACT

ADaM datasets are standardized assets in supporting clinical trial analysis and is required for regulatory submissions to FDA and other health authorities. To develop ADaM datasets, study programmers first need to specify analytical variables based on a provided Statistical Analysis Plan. Computer programs are then written as implementations. The final step is batch execution with explicit execution orders declared by study programmers.

The automation of ADaM development has been challenging. Solutions resort to modularized library development together with metadata repository to achieve some level of automation. Although it enables reusing partial codes from previous studies, it instigates duplicated efforts in specification writing and coding.

In this paper, we introduce a novel solution to make the specifications as single source-of-truth while achieving a completely automated deriving process.

INTRODUCTION

Conventional ADaM development incurs duplicated efforts in writing derivation rules and actual coding.

We review conventional ADaM development process to pinpoint problems and directions for a solution. Next, we draw clearly about our scope for ADaM automation in execution automation. We use a concrete example to show the rationale with the new design, which in essence improves the quality of variable derivation rules. Finally, we explain how it leads to execution automation.

ADAM DEVELOPMENT

There are two roles involved in deriving ADaM datasets. Study programmer, or first-line programmer, is responsible for writing ADaM specifications according to Study Analysis Plan. The drafting is often based on company specification standards, which by itself is maintained by a dedicated team for different therapeutic areas as well as keeping track with industry standards, for example, CDISC ADaM Implementation Guidance. Study programmer is also responsible to implement computer programs, utilizing code packages or libraries as many companies do establish over time. This was the case with SAS®, and nowadays there is a trend towards R. Eventually, ADaM datasets are generated following the order declared manually in the code programs in a controlled execution environment. On the discovery of any problems, or to improve robustness to handle data issues, code programs are updated. Ideally these cases should be reflected in the specification too, though it's left to the study programmer to what extend the specification is accurate.

QC Programmer, also known as second-line programmer, is responsible for quality checking on the dataset based on his/her consumption of the ADaM specification. He/she may also choose to base his/her checking on SAP, but it would incur more efforts spending. The accuracy in the specification directly impacts the understanding accuracy for the QC programmer. In bad cases, QC programmer must reach out back to the specification author for clarifications. It is less ideal for QC programmer to directly investigate the computer programs from the first-line programmer.

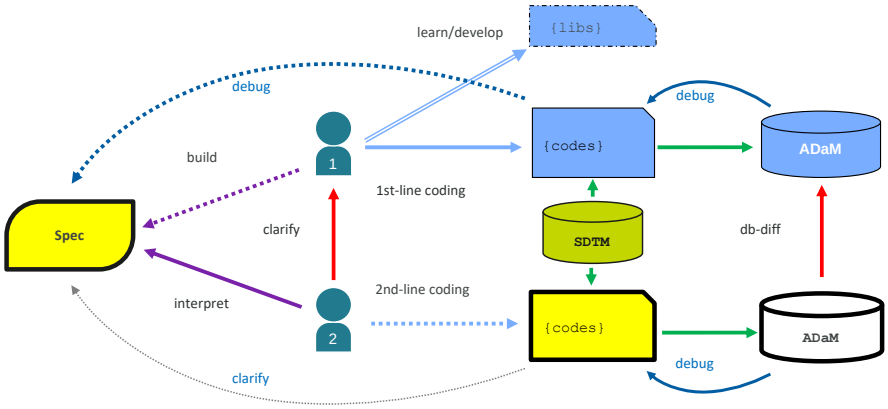

Figure 1 Conventional ADaM development workflow

PROBLEMS

The problems in current flow are in two-folds. Firstly, there are duplicated efforts in writing specifications and coding. Writing specifications and writing codes take place in different environments. Secondly, the indirect linkage between specification and dataset makes the derivation process prone to consistency problems between specifications and codes.

It is obvious the root is in duplication of specification and codes, therefore the direction for solutions is to write specifications of good quality to a level of being absolutely definitive. Efforts were made regarding different aspects, including on specification formats and derivation rules. We focus on solving the question of how to write definitive derivation rules.

There were attempts to improve the quality of derivation rules of two types. One is to add a code column in the specification Excel file next to the column containing the derivation rules. The rationale behind this is resorting to codes. Another form is to split the column containing the derivation rules into multiple columns. The thought process behind this is to decrease derivation rule's complexity by splitting into smaller blocks. However, neither is extensible, and causes more hassle when writing derivation rules constrained by specification writing environment.

ADAM AUTOMATION

The ADaM automation can mean a variable of things. For one, it can be referring to automated specification building, i.e., from study analysis plan to ADaM specification. Conceptually, specification building is to define a variable system - or a collection of datasets - which in turn is a collection of variables. A dataset can be thought of as a table. Every variable is a column in its corresponding table. Such a variable system is the metadata to describe the ADaM transformation process.

For another, it could be referring to automated execution of defined transformation processing, i.e., provided executables and upstream data, the derivation of output data is automated. This is our target in this paper. In the next section, we introduce an innovative approach to achieve automated executions.

THE REDESIGN

We propose a new solution by introducing a new language to express derivation rules. Creating domain specific languages is a common practice to deal with problems specific to a business domain. The challenge for our language is to seek balance between human expressiveness for disambiguation, and computer interpretability for deterministic behaviors. (An in-detail layout of the language is introduced in "A Domain Specific Language for ADaM Dataset Generation in Regulatory Clinical Research".)

An Example

It's important to mention that we require the use of two-level references to dependent variables. (See Yu and Kang, "Improving CDISC Basic Data Structure Requirements on PARAMCD for ADaM Automation".) **Table 1** is an example of a target variable's metadata. We include only domain name, variable name and derivation rule to be concise.

Domain	Variable	Derivation
ADSL	DTHDTN	Numeric value of ADSL.DTHDTC. If DTHDTC is partial or missing when DM.DTHFL = 'Y', 1. Set to max(LSTALVDT+1, imputed date) if max(LSTALVDT+1, imputed date) <= DCO; 2. set to missing otherwise; Impute date as follows: For missing day only - using the 1st of the month; For missing day and month - using the 1st of January.

Table 1. A Concise Example Definition of A Target Variable ADSL.DTHDTN. "DCO" is Data Cut-Off.

The modified version following the proposed grammar:

```
if DM.DTHFL = 'Y'
  and the latest of ([ADSL.LSTALVDT + 1 day, ADSL.DTHDTC])
  is not later than 2023-07-01,
then set to the latest of ([ADSL.LSTALVDT + 1 day, ADSL.DTHDTC]),
else set to blank.
Preprocessing Imputations:
- if month of ADSL.DTHDTC is missing, then impute with January.
- if day of ADSL.DTHDTC is missing, then impute with 01.
```

In this example, a top-level if-else statement is used, together with an explicit pre-processing imputation block. It also replaces a placeholder in the original description "DCO" (Data Cut-Off) with trial-specific value "2023-07-01", so that the metadata holds complete trial information, allowing it to be source-of-truth. Overall, with the refined structure and improved grammar, readability is not harmed and the meaning is more concrete and clear.

As an illustration of how computer interprets the text:

```
(
  ( if
    (
      ( DM.DTHFL = 'Y' )
      and
      (
        ( the latest of
          (
            ( [
              ( ADSL.LSTALVDT + ( 1 day ) ) ,
              ADSL.DTHDTC
              ] ) ) ) is not later than 2023-07-01 ) )
    then
    ( set to
      ( the latest of
        (
          ( [
            ( ADSL.LSTALVDT + ( 1 day ) ) ,
            ADSL.DTHDTC
            ] ) ) ) )
    else
    ( set to blank ) )
)
```

There are two key points to note. Firstly, expressions are identified correctly at all levels. This demonstrates its capability for describing more complex derivation rules. Secondly, chained sub-conditions are in the same scope. This is critical for expression evaluations, however not a focus in this paper.

For the completion of the discussion, the pre-processing imputation block is:

```
( Preprocessing Imputation :
  (
    ( -
      ( if
```

```
        (
          ( month of ADSL.DTHDTC ) is missing ) ,
        then
        impute with January ). )
    ( -
      ( if
        (
          ( day of ADSL.DTHDTC ) is missing ) ,
        then
        impute with 01 ). ) )
)
```

NEW WORKFLOW

With the new language to express derivation rules in a definitive way, we propose a new workflow for ADaM development. **Figure 2** is an illustration.

In the new workflow, study programmer will focus on specification writing, including what variables are available, and how they are derived. Programmer will be alerted on errors detected in the specification. For example, referencing undefined variables, or typos in variable names; un-parsable derivation rules.

QC programmer can review the specification at any moment, depending not on result data but only timeliness requirement of quality control activities.

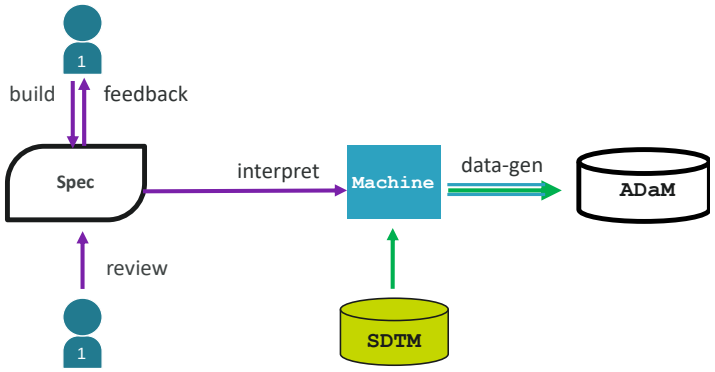

Figure 2. A new flow to derive ADaM datasets

In the next section, we will delve into how we achieve fully automated executions with the new design.

ADAM AUTOMATION

Our implementation at AstraZeneca has two parts, a frontend and a backend, as shown in Figure 3. The frontend is able to do static code analysis and provides in-time check for derivation rules. The backend provides execution environment for consuming input SDTM datasets and generates output ADaM datasets.

As can be seen, it is largely simplified with overheads removed, compared to Figure 1. Specifically, the development and maintenance of code libraries are saved.

On the other hand, the specification template component in Figure 1 is left out because it should not be a dependent for automation. Automation that depends on standardization is not only slow to achieve, but also by design unable to achieve complete coverage rate. With the new flow design, ADaM automation is unblocked from the establishment of standards and standard libraries as conventional automation approach requires, while achieving 100% automated execution.

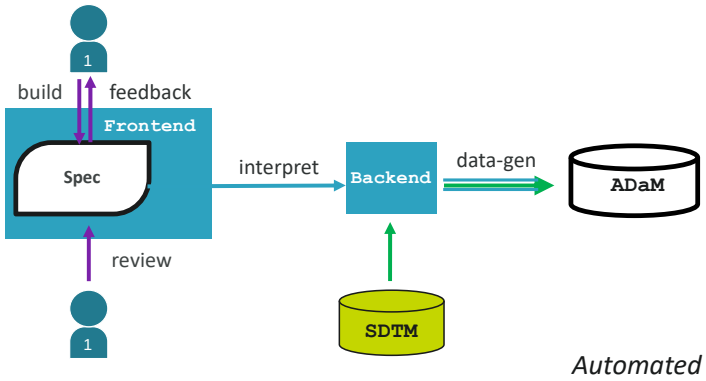

Figure 3 Architect diagram for an ADaM dataset generation application

DEPENDENCY ANALYSIS

We highlight one particular useful feature in the frontend: variable dependency analysis. With the definitive reference of defined variables from upstream SDTM and dependent ADaM variables, visualizing the dependencies is a simple task of searching. Computationally this leads to potential optimization of the variable system itself, and possible benefits to standardization, too.

The dependency visualization for the previous example target variable in **Table 1** is in Figure 4.

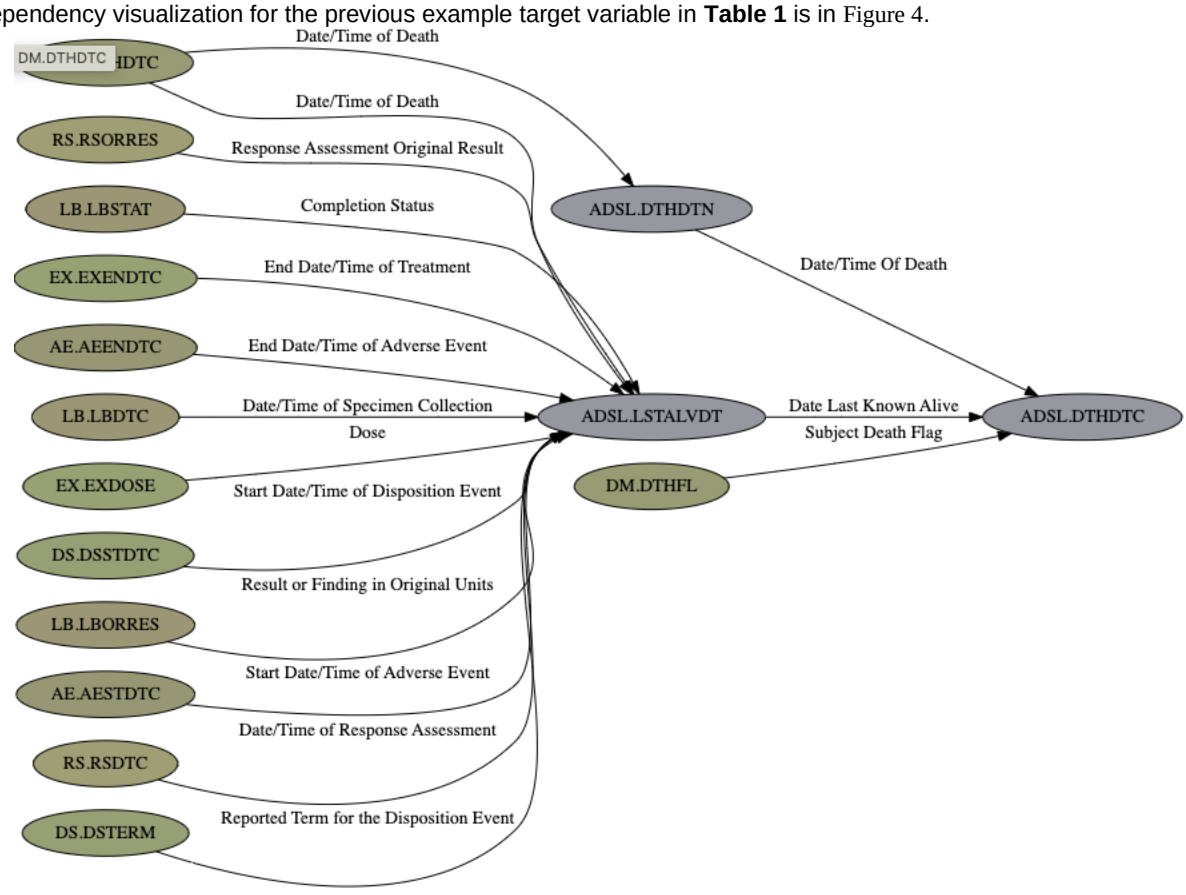

Figure 4 Dependencies for example target variable and tracing backward to SDTM variables

ORCHESTRATION FOR EXECUTION

Now that the metadata is fully machine-readable, with variable name, type, and length well-defined, plus derivation rules being computer interpretable, we are able to automate the deriving execution process.

Figure 5 shows a partial depiction of ADSL variables from an example late-phase oncology trial. What's important is the dependencies among the variables, and that there are no circular dependencies. A graph of such kind is called a directed acyclic graph (DAG). A DAG is automatically generated thanks to the two-level references we require to use. CDISC Basic Data Structure is more complicated. We solve that by introducing extra notations (Bo, Y, 2023, "Improving CDISC Basic Data Structure Requirements on PARAMCD for ADaM Automation").

We use Apache Airflow™ as the scheduler and implement customized operators to interpret derivation rules for every ADaM

variable. Again, as shown in Figure 5**Error! Reference source not found.**, each block is a task to derive its corresponding variable.

CONCLUSION

By introducing a domain specific language, ADaM development is transformed with refined quality improving both efficiencies and correctness. It also leads to fully automated execution of the deriving process, compared with the traditional standardization-based automation strategy.

Admittedly, to onboard the solution it involves learning a new language to describe derivation rules. Many efforts are made to make the language intuitive. Besides, we believe it is more of a technical debt to tackle than a burden to carry. Besides, with improved study specification quality, it is more beneficial than harmful to standard content management, too.

Future work includes refining the language and provide management solution on metadata standards. We also want to apply the same methodology to generate SDTM datasets. For automation, future work is to provide tools to improve confidence and assure qualities.

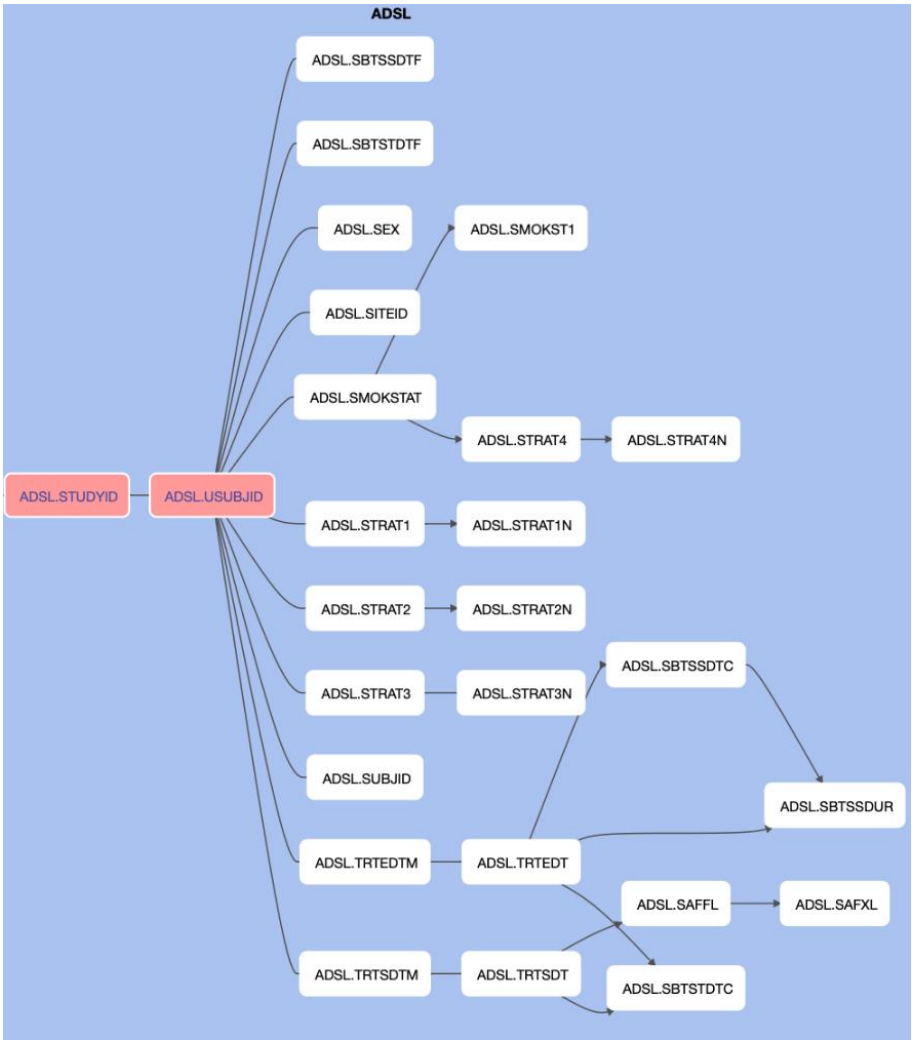

Figure 5 Example Subset of Variable Dependencies When Deriving ADSL Dataset

REFERENCES

Kang Zhao. 2023. A Domain Specific Language for ADaM Dataset Generation in Regulatory Clinical Research. *PharmaSUG China 2023 Proceedings*, Shanghai, China.

Yu Bo, and Kang Zhao. 2023. Improving CDISC Basic Data Structure Requirements on PARAMCD for ADaM Automation. *PharmaSUG China 2023 Proceedings*, Shanghai, China.

CDISC Analysis Data Model Team. Analysis Data Model, Version 2.1. 2009. Available at https://www.cdisc.org/standards/foundational/adam/adam-v2-1.

CDISC Analysis Data Model Team. Analysis Data Model Implementation Guide, Version 1.3. Available at https://www.cdisc.org/standards/foundational/adam/adamig-v1-3-release-package.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
Kang Zhao
AstraZeneca
No. 88, North Xizang Road, Jing'an District
Shanghai/200000, China
kang.zhao@astrazeneca.com
https://www.linkedin.com/in/kangzhao-bbgw