PharmaSUG China 2023 - Paper AA-101

A Domain Specific Language for ADaM Dataset Generation in Regulatory Clinical Research

Kang Zhao, AstraZeneca

ABSTRACT

ADaM datasets are critical in supporting clinical research analysis, and are standard assets for data submissions to FDA (U.S.) etc. To generate ADaM datasets, it starts with programmer analysts specifying analytical variables based on each provided study analysis plan. Then computer programs are written as implementations. Final step is batch execution with manually declared execution orders.

The execution automation has been challenging, and millions of dollars have been invested to 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 specifications and codes.

The Data Transformation Team in China Biometrics at AstraZeneca introduce a novel solution to keep single source-of-truth while achieving complete execution automation. A domain-specific language is designed and used for describing derivation rules, then an execution engine is implemented which can automatically execute specification-defined derivation steps to generate data outputs.

INTRODUCTION

Figure 1 illustrates traditional flow for deriving ADaM datasets.

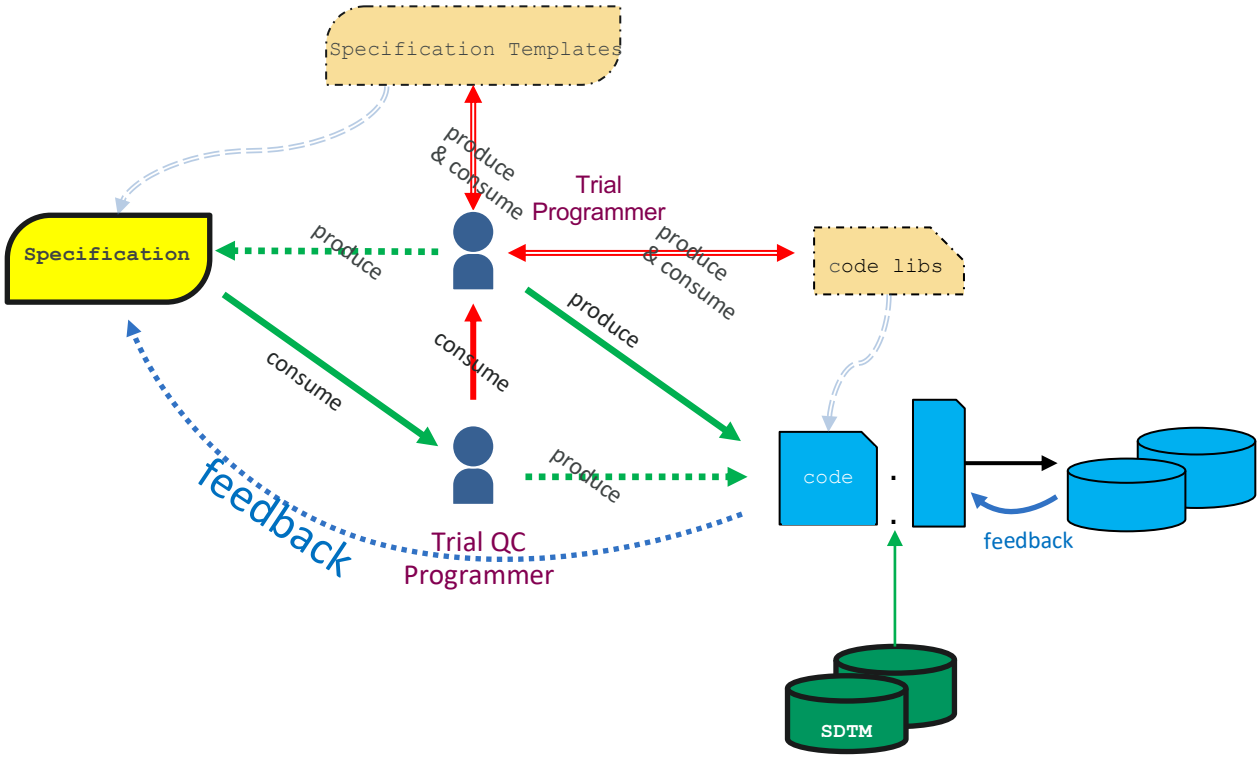

Figure 1. Traditional ADaM Dataset Generation Process Illustration

There are two roles involved in deriving ADaM datasets. Trial programmer, also known as first-line programmer, is responsible for writing ADaM specifications according to Study Analysis Plan (SAP). The drafting is often based on company (specification) standards, which by itself is maintained by a dedicated team for maintenance for different therapeutic areas as well as keeping track with industry standards, for

example, CDISC ADaM Implementation Guidance. Trial 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 trial programmer to what extend is the specification accurate.

Trial 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 has to reach out back to the specification author for clarifications. It is less ideal for QC programmer to directly look into the computer programs from the first-line programmer.

PROBLEMS

The problems in current flow are:

1. Duplicated efforts in writing specs and codes. Writing specifications and writing codes take place in different environments. For example, at AstraZeneca it's Microsoft® Excel for writing specifications, and SAS Studio® for writing SAS® codes and R Studio® for writing R codes.
2. Prone to consistency problems between specifications and codes. It may not be an obvious problem, as long as codes are correct therefore the result datasets.

Since they root in the duplication of specification and codes, the idea for solutions is to write specifications of good quality. Efforts were made regarding different aspects, including:

1. Specification formats.
2. Derivation rules.

In order to improve the quality of derivation rules, there have been attempts, which can be categorized into two:

1. 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.
2. 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. In the next section, we introduce a systematic redesign to answer the question of what derivation rules of good quality are.

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 domain specific - as its name indicates - problems. The challenge for our language is to seek balance between human expressiveness for disambiguation, and computer interpretability for deterministic behaviors.

LANGUAGE DESIGN

We will lay out a structure based on our experiences and understandings to required capabilities to derive ADaM datasets successfully.

Expressions

Expressions are values, including primary values such as integers, strings, and complex values such as transformation, filtering, etc. We implement expressions in the following categories:

Selection

```
DOMAIN.VARIABLE
```

Multiple Selections

```
[DOMAIN.VARIABLE1, DOMAIN.VARIABLE2]
```

Conversions

```
DOMAIN.VARIABLE uppercase
```

Extractions

```
the date part of DOMAIN.VARIABLE
```

Calculations

```
DOMAIN.VARIABLE1 - DOMAIN.VARIABLE2
DOMAIN.VARIABLE + 1 day
```

Filters

```
DOMAIN.VARIABLE where <conditions>
```

Aggregations

```
the largest of ( DOMAIN.VARIABLE1 per DOMAIN.VARIABLE2 )
the first of DOMAIN.VARIABLE1
  ordered by DOMAIN.VARIABLE2
  per DOMAIN.VARIABLE3
```

Statements

Statements are different from expressions in that the interest is not return values, but rather cause side-effects. A typical side-effect for example is value persistence. In our design, the side-effect is write to database so that the value is saved and can be used for deriving dependent variables.

Assignment

```
set to <expression>
```

Control flow

```
if <condition1> then <statement1>
else if <condition2> then <statement2>
else ...
```

Imputation

Imputations is alike assignments. It is worthy of attentions because deriving ADaM datasets involves imputations on missing values, which is a differentiating factor from deriving SDTM datasets. We provide hooks for pre-processing and post-processing imputation respectively. Pre-processing imputation impute referenced variables before deriving target variable; post-processing imputation impute target variable after it's derived and persisted, and overwrites the target variable.

```
Preprocessing Imputations
Postprocessing Imputations
```

AN EXAMPLE

We list an example here for more concrete ideas of the language design. It's important to mention that we require the use of two-level references to dependent variables. 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. An Concise Example Definition of A Target Variable ADSL.DTHDTN

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" (standing for 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 ) ). )
```

Two key points to note are:

1. Expressions are identified correctly at all levels. This demonstrates its capability for describing more

complex derivation rules.

2. 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 illustration for 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 ). ) ) )
```

FLOW DESIGN

With the new language for describing definitive derivation rules, we present a new flow on top of it. Figure 2 is an illustration. The component in the middle has two parts, a frontend and a backend. 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 templates 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, you save the cost to establish standards as conventional automation approaches require, while achieving 100% automated execution.

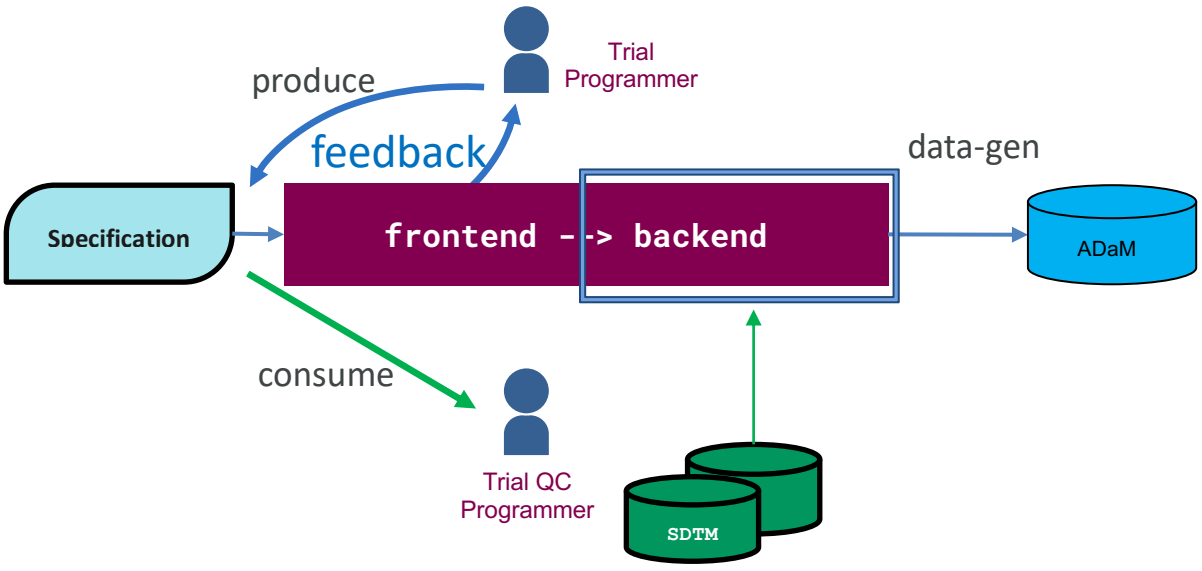

Figure 2. Automated ADaM Dataset Generation

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

AUTOMATING ADAM DATASET GENERATION

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 3 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 3, each block is a task to derive its corresponding variable.

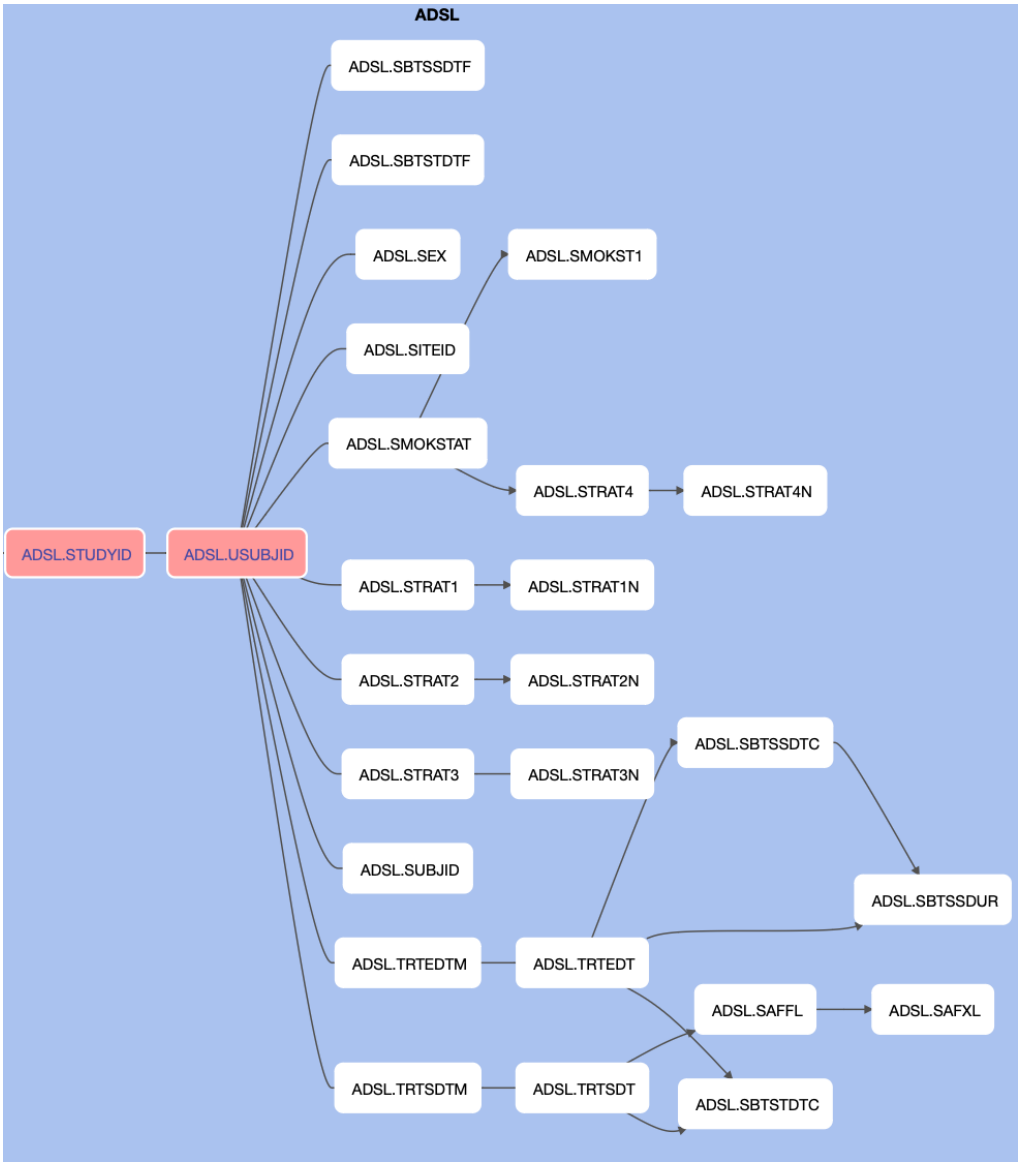

Figure 3. Example Subset of Variable Dependencies When Deriving ADSL Dataset

On top of variable dependencies, you have a direct visualization at domain-level as shown in Figure 4. This is also automatically obtained by utilizing the Task Group in Apache Airflow™.

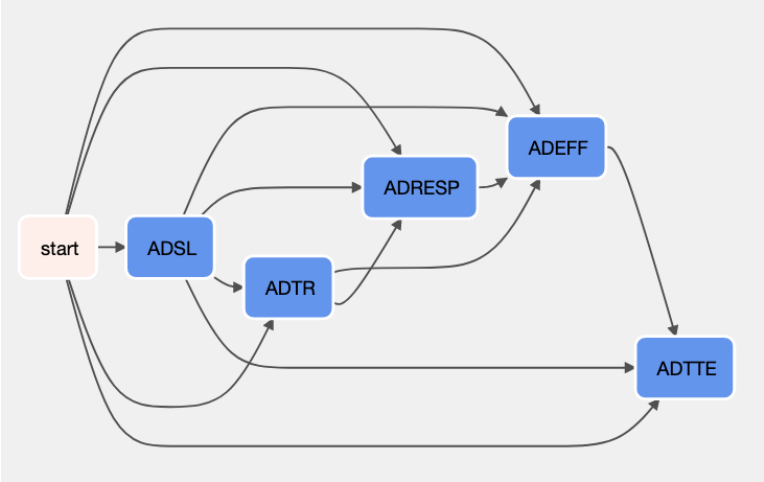

Figure 4. Example Domain Dependencies When Deriving ADaM Datasets

CONCLUSION

With a reasonable design of a domain specific language as proposed in this paper, trial's ADaM specifications have refined quality improving not only efficiencies but also correctness. This has a longer term impacts considering how standards are evolved from trials over time.

Admittedly, to onboard this solution it involves learning a new language to describe derivation rules, however we believe it is more of a technical debt to tackle than a burden to carry.

Computer interpretable derivation rules makes it possible to achieve fully automated execution of the deriving process, compared with traditional standardization based automation strategy.

Future work include 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.

REFERENCES

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

Fowler, Martin. 2010. Domain Specific Languages. Addison-Wesley.

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.

ACKNOWLEDGMENTS

Heartful thanks to China Biometrics Leader Team from AstraZeneca, Shanghai, China, for their supports on the project and this paper.

CONTACT INFORMATION

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

Kang Zhao
AstraZeneca
kang.zhao@astrazeneca.com
https://www.linkedin.com/in/kangzhao-bbgw