Paper 11101-2016

Metadata Statistics As an Aid for Code Review

Frank Poppe, PW Consulting; Joris Huijser, De Nederlandsche Bank

ABSTRACT

Code review is an important tool for ensuring the quality and maintainability of an organization’s ETL processes.
This paper introduces a method that analyzes the metadata to assign an alert score to each job.
For the jobs of each flow, the metadata is collected, giving information about the number of transformations in a job, the amount of user-written code, the number of loops, the number of empty description fields, whether naming conventions are followed, and so on.
This is aggregated into complexity indicators per job.
Together with other information about the jobs (e.g. the CPU-time used from logging applications) this information can be made available in SAS® Visual Analytics, creating a dashboard that highlights areas for closer inspection.

INTRODUCTION

One of the main missions of The Dutch Central Bank (De Nederlandsche Bank, DNB) – as part of the European System of Central Banks – is retaining financial and macro-economic stability in The Netherlands. The supervision of financial institutions contributes to this mission. Banks, insurance companies and pension funds send monthly, quarterly or yearly reports about their financial status to DNB.

The Statistics Division of DNB is responsible for the collection, storage, transformation and publication of data received from these financial institutions. Because the information and knowledge drawn from this data can have severe consequences for some of these supervised financial institutions, the quality of the data is of great importance for DNB-supervisors, as well as for supervisors from our external consumers, such as the European Central Bank (ECB) and the International Monetary Fund (IMF).

In this respect code quality is equally important for sustaining the quality of data. Meanwhile changes in regulations for supervision – mandated by the Dutch government, or more and more by the ECB – put a lot of pressure on the maintainability of our systems. Growing legacy has exposed the necessity of code review.

The need was felt for a quick assessment of all the jobs that would pinpoint those jobs that need closer inspection. Existing methods use e.g. the number of code lines and the number of data sets read and written to score jobs. We felt this was inadequate. E.g., a job that uses standard DI Studio transformations might, through those transformations, generate lots of code, while still being perfectly maintainable.

We want to look for jobs that have a large number of transformations, or have user written code through any means, etcetera. All this information is available in the metadata. To extract and analyze that information jobs were developed. These jobs then aggregate the Information and turns them into indicators per job.

Because jobs may change over time we also want to implement a mechanism to repeat the analysis on a regular basis. Furthermore the data also can be combined with data from other sources. DNB already has a system in place that collects information on the scheduled jobs and connects that to information extracted from the SAS metadata. The information includes the moments the jobs have run, how much CPU-time the job and each transformation used, etcetera. The resulting table is available in SAS Visual Analytics, and a dashboard can be created that shows the areas that deserve attention.

ANALYZING THE METADATA

WHY LOOK AT THE METADATA?

Once we decided we wanted to try and develop a tool that would assist in pinpointing those jobs that required closer inspection we started to think about the number of lines of code. But we realized that the majority of the code is no longer written by programmers, but is generated by DI Studio. If you ever looked at the generated code you will have seen that what was a straightforward transformation on the canvas of DI Studio sometimes becomes long and tedious code.
Nowadays the maintainability of a job is no longer determined by the number of lines of code. It is much more important how complex it is looks in the interface of the application that is used to maintain that job: DI Studio.

So we decided to look at the number of steps (i.e. transformations) in a job, and whether it only uses standard transformation or other ways of introducing code. Those other ways might be:

- Using so-called User Generated Transformation (UGT).
- Using code blocks through so-called User Written Code transformations (UWC).
- Replacing the generated code with own code

Our guidelines for programmers state that preferably standard transformation have to be used. When that is not possible, alternatives can be considered, in the order given above.

Other complicating factors in a job that we wanted to take into account were the use of the loop-control transformation, and using jobs within jobs. There can be very good reasons for doing so, but it does complicate jobs, and it certainly complicates the analysis when there are run-time errors!

All the information on jobs, steps and the transformations being used can be found in the metadata, so we started to look there.

WHERE TO LOOK IN THE METADATA?

The metadata is well documented. That is, the documentation is complete and correct (see the references [1] and [2] for the 9.3 and 9.4 documentation, respectively).

But the documentation doesn’t help in understanding how and where to find the information you are looking for. The question is: how is the information you see in DI Studio and in the SAS Management Console spread over the different objects in the metadata? The metadata is a collection of different types of objects, and each type of object can have particular associations with other objects. The types of objects, the attributes of those objects, the kind of associations it can have and the cardinality of it are all documented.
For a table and its columns that is fairly simple. There is an object defining the table: the PhysicalTable object. Such an object can have a *Columns* association with a Column object. The cardinality of this association is 0…*, i.e. a table can have zero or any number of columns. All associations are reversible, and the association from the Column object back to the PhysicalTable is called *Table*. This is an association of cardinality 1…1, i.e. a Column is always associated with exactly one column. This implies that deleting a table will also delete all the columns of that table (while the reverse is not true).

For the relation between jobs and the steps of the jobs, and then to the particular transformation used in those steps, it is less obvious to which metadata objects one should look.
The same is true for the relations between the jobs and the deployed jobs, and then between the deployed jobs and the flows.

In our experience the most instructive way was to have DI Studio open on a typical job and its properties, and at the same time use a tool to inspect the metadata to locate the same information there. In the following paragraphs we describe some tools to inspect the metadata.

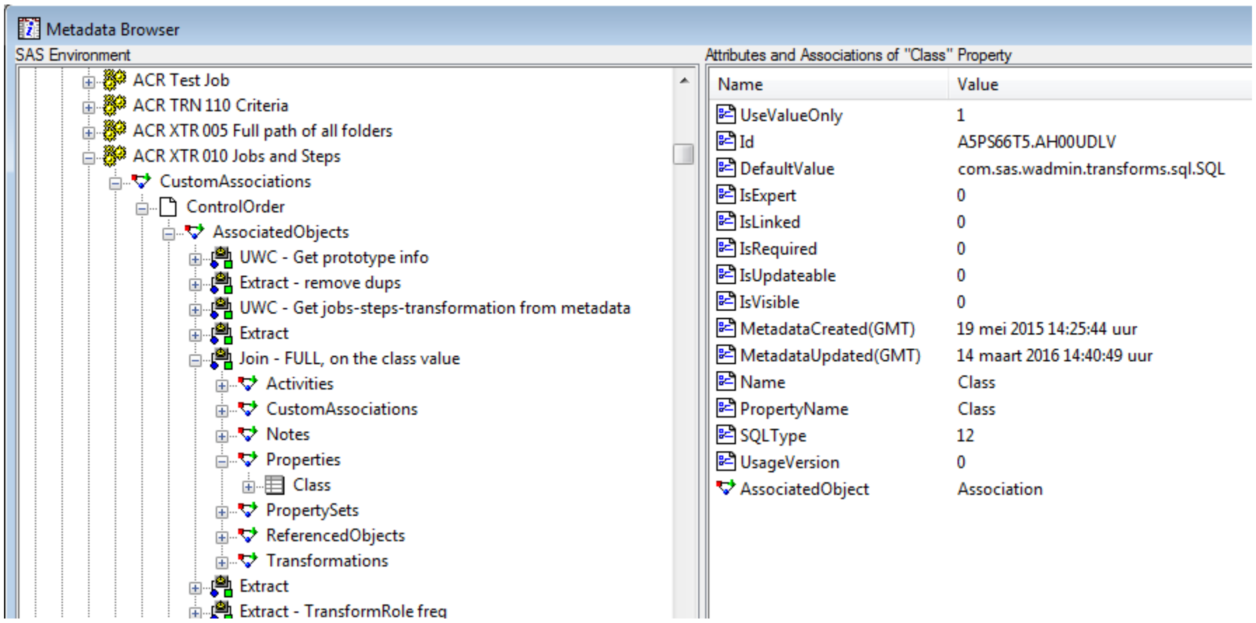
TOOLS TO INSPECT THE METADATA

In the following paragraphs three tools are introduced. The screenshots illustrating each tool all show more or less the same information. This is information about the relation between a job, a step and the transformation used in that step. This example will also be used in the paragraph “Relevant metadata objects and their associations” later in this paper.

Metabrowse in a SAS Foundation session

In an ‘old fashioned’ SAS Foundation session there is the ‘metabrowse’ command. This will start a window with a graphical view of the metadata in a metadata server you can connect to. There one can inspect any metadata object and its attributes, its associations and then move to an associated object. This way one can traverse the whole metadata environment – or run in endless circles…

If you have this available this is a fine tool to discover how the objects you are interested are linked together. Figure 1 gives an example of this window, showing one of the jobs we use to extract the metadata on the jobs and its steps.



Metadata Browser

SAS Environment

- ACR Test Job
- ACR TRN 110 Criteria
- ACR XTR 005 Full path of all folders
- ACR XTR 010 Jobs and Steps
  - CustomAssociations
    - ControlOrder
      - AssociatedObjects
        - UWC - Get prototype info
        - Extract - remove dups
        - UWC - Get jobs-steps-transformation from metadata
        - Extract
        - Join - FULL, on the class value
          - Activities
          - CustomAssociations
          - Notes
          - Properties
            - Class
          - PropertySets
          - ReferencedObjects
          - Transformations
        - Extract
        - Extract - TransformRole freq

Attributes and Associations of "Class" Property

Name	Value
UseValueOnly	1
Id	A5PS66T5.AH00UDLV
DefaultValue	com.sas.wadmin.transforms.sql.SQL
IsExpert	0
IsLinked	0
IsRequired	0
IsUpdateable	0
IsVisible	0
MetadataCreated(GMT)	19 mei 2015 14:25:44 uur
MetadataUpdated(GMT)	14 maart 2016 14:40:49 uur
Name	Class
PropertyName	Class
SQLType	12
UsageVersion	0
AssociatedObject	Association

Figure 1. The metabrowse window

Unfortunately a SAS Foundation session has become less and less available for many users. If you don’t have access you have to resort to one of the other options.

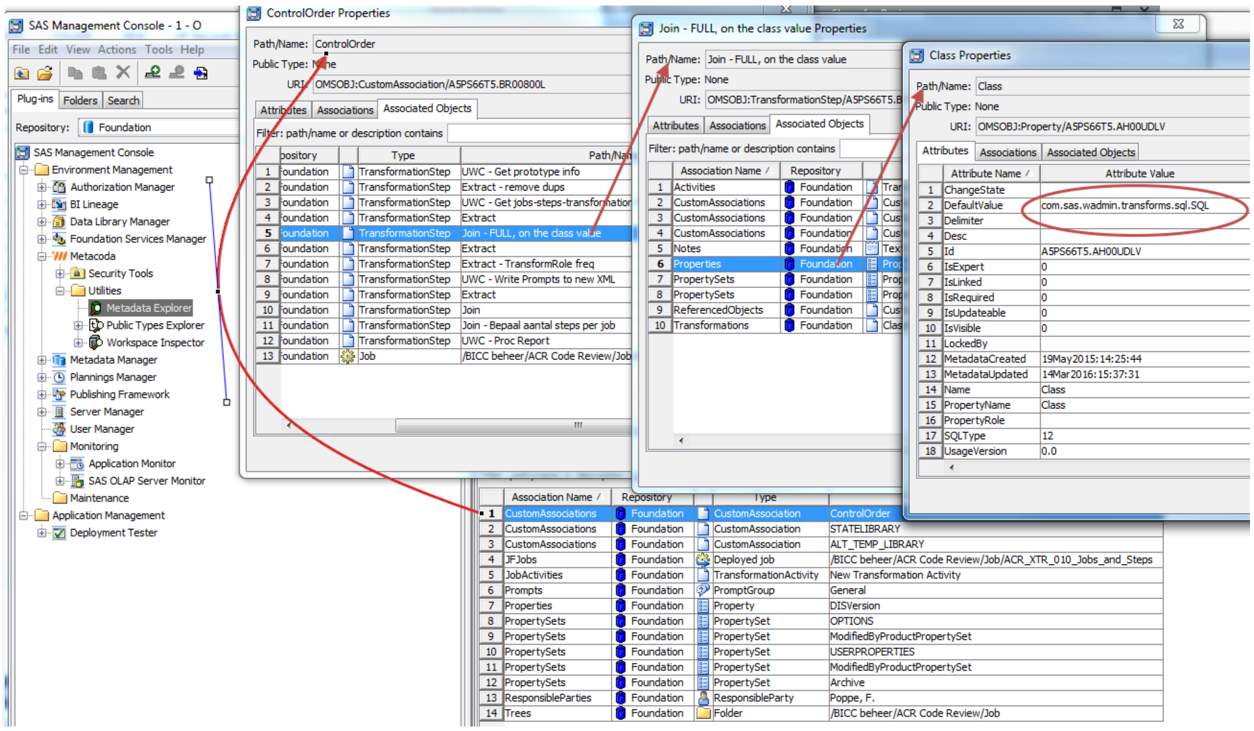
The Metacoda metadata explorer

Metacoda, a software company providing utility tools for SAS products, offers a freely available metadata exploration plug-in for the SAS Management Console (see reference [3] for more information). It functions in the same way as the metabrowse window described in the previous paragraph, but each object opens in a new window. This makes it easier to lose track of the path you are following. The screenshot in Figure 2 shows the information on the same objects as in Figure 1.

A web interface using Stored Processes

A previous release of SAS (9.1.3) included a set of Stored Processes that together formed the web-based *SAS Metadata Explorer*. It was not delivered with the next release (9.2), but was available upon request. We haven’t been able to get an up to date version for 9.3, which we are currently running. Rumor has it (from ‘a little bird’) that an updated version *does* exist, but apparently is it not available for customers.

However, writing such Stored Processes is no rocket science, and there are several crude versions around. The screenshot in Figure 3 is from such an application, which we occasionally use.



ControlOrder Properties

Path/Name: ControlOrder
Public Type: None
URI: OMSOBJ:CustomAssociation/A5PS66T5.BR00800L

Attributes | Associations | Associated Objects

Filter: path/name or description contains

	pository	Type	Path/Na
1	oundation	TransformationStep	UWC - Get prototype info
2	oundation	TransformationStep	Extract - remove dups
3	oundation	TransformationStep	UWC - Get jobs-steps-transformation
4	oundation	TransformationStep	Extract
5	oundation	TransformationStep	Join - FULL, on the class value
6	oundation	TransformationStep	Extract
7	oundation	TransformationStep	Extract - TransformRole freq
8	oundation	TransformationStep	UWC - Write Prompts to new XML
9	oundation	TransformationStep	Extract
10	oundation	TransformationStep	Join
11	oundation	TransformationStep	Join - Bepaal aantal steps per job
12	oundation	TransformationStep	UWC - Proc Report
13	oundation	Job	/BICC beheer/ACR Code Review/Job

Join - FULL, on the class value Properties

Path/Name: Join - FULL, on the class value
Public Type: None
URI: OMSOBJ:TransformationStep/A5PS66T5.B

Attributes | Associations | Associated Objects

Filter: path/name or description contains

	Association Name	Repository	
1	Activities	Foundation	Tran
2	CustomAssociations	Foundation	Cus
3	CustomAssociations	Foundation	Cus
4	CustomAssociations	Foundation	Cus
5	Notes	Foundation	Tex
6	Properties	Foundation	Prop
7	PropertySets	Foundation	Prop
8	PropertySets	Foundation	Prop
9	ReferencedObjects	Foundation	Cus
10	Transformations	Foundation	Clas

Class Properties

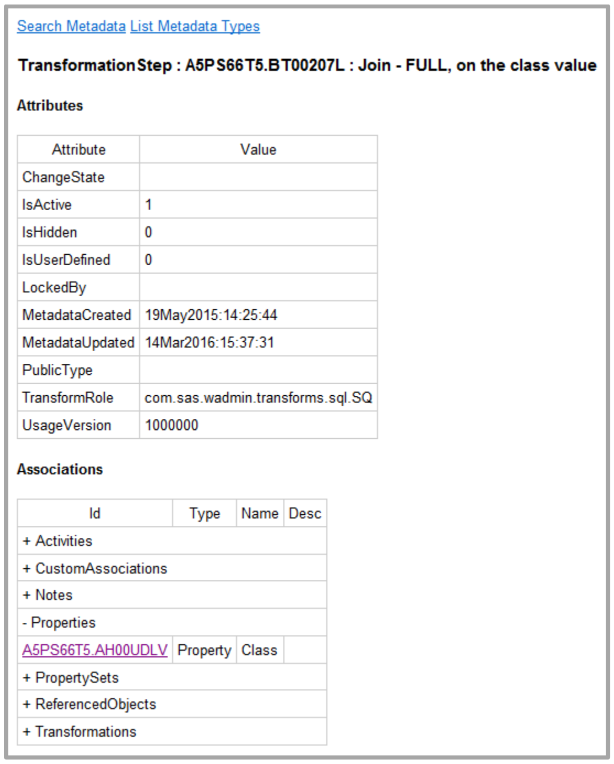
Path/Name: Class
Public Type: None
URI: OMSOBJ:Property/A5PS66T5.AH00UDLV

Attributes | Associations | Associated Objects

	Attribute Name	Attribute Value
1	ChangeState	
2	DefaultValue	com.sas.wadmin.transforms.sql.SQL
3	Delimiter	
4	Desc	
5	Id	A5PS66T5.AH00UDLV
6	IsExpert	0
7	IsLinked	0
8	IsRequired	0
9	IsUpdateable	0
10	IsVisible	0
11	LockedBy	
12	MetadataCreated	19May2015:14:25:44
13	MetadataUpdated	14Mar2016:15:37:31
14	Name	Class
15	PropertyName	Class
16	PropertyRole	
17	SQLType	12
18	UsageVersion	0.0

	Association Name	Repository	Type	
1	CustomAssociations	Foundation	CustomAssociation	ControlOrder
2	CustomAssociations	Foundation	CustomAssociation	STATELIBRARY
3	CustomAssociations	Foundation	CustomAssociation	ALT_TEMP_LIBRARY
4	JFJobs	Foundation	Deployed job	/BICC beheer/ACR Code Review/Job/ACR_XTR_010_Jobs_and_Steps
5	JobActivities	Foundation	TransformationActivity	New Transformation Activity
6	Prompts	Foundation	PromptGroup	General
7	Properties	Foundation	Property	DISVersion
8	PropertySets	Foundation	PropertySet	OPTIONS
9	PropertySets	Foundation	PropertySet	ModifiedByProductPropertySet
10	PropertySets	Foundation	PropertySet	USERPROPERTIES
11	PropertySets	Foundation	PropertySet	ModifiedByProductPropertySet
12	PropertySets	Foundation	PropertySet	Archive
13	ResponsibleParties	Foundation	ResponsibleParty	Poppe, F.
14	Trees	Foundation	Folder	/BICC beheer/ACR Code Review/Job

Figure 2. The Metacoda Metadata Explorer windows



Search Metadata List Metadata Types

TransformationStep : A5PS66T5.BT00207L : Join - FULL, on the class value

Attributes

Attribute	Value
ChangeState	
IsActive	1
IsHidden	0
IsUserDefined	0
LockedBy	
MetadataCreated	19May2015:14:25:44
MetadataUpdated	14Mar2016:15:37:31
PublicType	
TransformRole	com.sas.wadmin.transforms.sql.SQ
UsageVersion	1000000

Associations

Id	Type	Name	Desc
+ Activities			
+ CustomAssociations			
+ Notes			
- Properties			
A5PS66T5.AH00UDLV	Property	Class	
+ PropertySets			
+ ReferencedObjects			
+ Transformations			

Figure 3. Stored Process showing information from the metadata

Because the window refreshes with each switch from any object to the next object only the current objects and its associations can be visible at any given time. This means that also here one easily can lose track of the path one has navigated.

It should be possible to develop a more advanced version of the web interface. The interface of such a version should give a better view of the navigation path, by displaying 'breadcrumbs' or by keeping all steps in view like in the SAS Metabrowse window.

RELEVANT METADATA OBJECTS AND THEIR ASSOCIATIONS

Using the tools described in the previous paragraphs we determined how to navigate the metadata in order to collect the information we were looking for.

The schema in Figure 4 shows the path from a job to its steps, and then to the actual transformation they use.

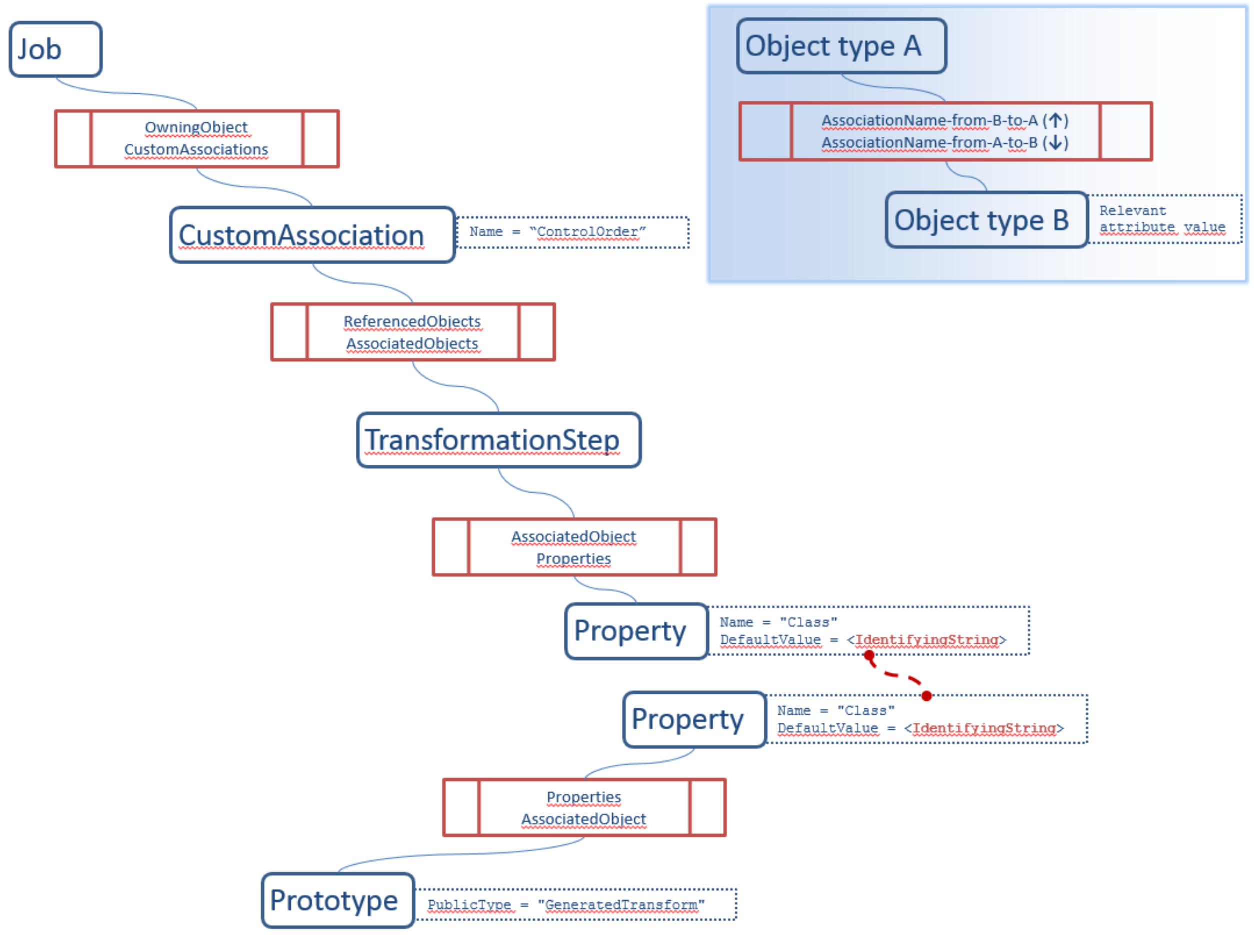


Figure 4. Relation between Jobs, Steps and Transformations

The route from the <Job> object through the <CustomAssociation> object returns the <TransformationStep> objects (the 'blocks' on the canvas in DI Studio) in the order they are executed.

However, not all relations can be found through associations in the metadata – rather surprisingly.

As can be seen in Figure 4 the <TransformationStep> objects have an association to a <Property> object named "Class". The value of the *DefaultValue* attribute is a string that refers to the actual transformation being used. This works in either of two ways:

- If this value is of the form `"com.sas.wadmin.visuals.SASSort"` (i.e. words separated by dots) it is a transformation delivered with DI Studio, with a Java-developed interface. These transformation have their own designated icon in the Transformation pane in DI Studio. We have not investigated where and how these transformations can be found in the metadata (if at all), nor how Java-developed transformation developed by third parties show up.

- If the value is of the form `"0171b302-ac10-926b-01de-0c0901dd3b66"` (i.e. codes separated by dashes) it refers to transformations developed through the *New Transformation* action. These transformations all have the same icon in the Transformations pane: . This can be either a transformation delivered with DI Studio, or a transformation locally developed transformation. These transformations are stored as in a <Prototype> object, together with several associated objects that store the information on source code, parameters, options, etcetera. There will always be an associated <Property> object, with a *DefaultValue* matching that of the <Property> object associated with the <TransformationStep>.

The screenshots of Figure 1 to Figure 3 show, each in their own way, the <TransformationStep> and <Property> objects.
In the last one the value for *DefaultValue* can be seen in a truncated form as the value of the *TransformRole* attribute of the <TransformationStep>. This however is not always the case, and we haven't used this attribute.

A similar schema of the relations between objects was developed to extract the information on jobs, their deployed manifestations, and the flows they are being used in. A complicating factor here was that flows can be used in flows. That meant that a complete view only could be found through an iterative analysis of the relations.

COLLECTING THE INFORMATION FROM THE METADATA

All information from the metadata is collected by calls to METADATA Procedure. (In the very beginning we started with the DATA Step Metadata functions, because then it is much easier to define the query. But as soon as one tries to extract information on several objects using do loops, the amount of time grew exponentially.)
The following XML-query collects information on a subset of jobs. In this case the subset is defined, through the select-clause at the end of the query, as all jobs residing in the 'ACR Code Review' metadata subtree (this is the subtree where this job itself resides).
The circled digits refer to the explanations below - although this is by no means a full explanation of the syntax of such queries! The other queries are similarly built, but are not given here in detail. If you are interested, contact the authors.
The queries is as follows:

```
<GetMetadataObjects>
  <ReposId>$METAREPOSITORY</ReposId>
  <Type>Job</Type> ①
  <Objects/>    ②
  <Ns>SAS</Ns>  ③
  <Flags>2436</Flags> ④
  <Options>
    <Templates>  ⑤
      <Job Id="" Name="" Desc="" IsActive="" IsHidden=""
          IsUserDefined="" >
        <CustomAssociations
          search="CustomAssociation[@Name='ControlOrder']"/>
        <Prompts />
        <Trees />
      </Job>
      <CustomAssociation Id="" Name="" >  ⑥
        <AssociatedObjects/>
      </CustomAssociation>
      <TransformationStep Id="" Name="" Desc="" TransformRole=""
```

```
        IsActive="" IsHidden="" IsUserDefined=""> ⑦
         <Transformations/>
         <Properties search="Property[@Name='Class']" />
      </TransformationStep>
      <Property Id="" Name="" DefaultValue=""/>  ⑧
      <ClassifierMap Id="" Name="" Desc="" TransformRole="" >
        <SourceCode/>
      </ClassifierMap>
      <Select Id="" Name=""  />
      <PromptGroup Id="" GroupInfo="" GroupType="" />
      <TextStore Id="" StoredText="" TextRole="" TextType="" />
      <Tree Id="" Name="">
        <ParentTree />
      </Tree>
    </Templates>
    <XMLSelect search="Job[Trees/*[@Name='ACR Code Review']]" />  ⑨
  </Options>
</GetMetadataObjects>
```

The following remarks explain some of the elements:

1. Primarily we are interested in Job objects.
2. The <Objects> element is required, but we do not use it to select objects, instead we use the <Templates> and <XMLSelect> elements.
3. Then <Namespace> element is required, and for these queries should specify “SAS”.
4. With the <Flags> element one stipulates the behavior of the query. It is a combination of bitwise flags. The value 2436 is the result of adding 4 + 128 + 256 + 2048. These flags have the following meaning:
    - 4: A <Templates> element will be used.
    - 128: A <XMLSelect> element will be used.
    - 256: Execute a GetMetadata call for each object found.
    - 2048: Return only attributes that have a non-null value.
5. In the <Templates> element one can specify which object and which associations one is interested in, and which attributes should be returned. The first element specifies is the <Job> object, and within that element one specifies the associations that should be followed:
    - CustomAssocations; but only it if leads to a CustomAssociation object for which the attribute Name has the value “ControlOrder”;
    - Prompts (which indicate the job is parameterized);
    - Trees (which will lead to the metadata folder for the job).
6. Following the <Job> element are the objects that will be found through the associations that were specified, starting with the <CustomAssociation> (obviously from the CustomAssociations). Within the <CustomAssociation> element we request the AssociatedObjects association.
7. The CustomAssociation will give us the TransformationStep object, which is described here.
8. Some other objects and associations are requested with further information.
9. The <XMLSelect> element gives the opportunity to limit the search to some objects, or, as in this case, to a single object (although theoretically one might have more jobs with the same name).

TIMING INFORMATION ON THE JOBS

At DNB all scheduled flows that are run in batch have the same standard job as the last one. This last job is added by a Stored Process that *must* be used for every flow. Using this Stored Process also ensures that there exists an up to date table with information on the jobs, extracted from the metadata. This includes information on the steps. The history of this information is retained.

This last job uses that table, checks the log files for all the jobs of the flows, extracts timing information from those logs and adds that to another table, building historical information on the runs of all flows, jobs and steps. Because also the history on the metadata of the job and its steps is available the development over time can take both into account. One can check for instance of a change in the amount of CPU time used coincides with a change in the job.

This system of Stored Processes and tables in use at DNB is called *“Continuous Monitoring”*.

Because also in *Continuous Monitoring* the information on flows, jobs and steps is identified by the metadata-id, it can be combined with the information described in the previous paragraphs.

ANALYSIS OF THE JOBS AND STEPS

SOME GENERAL RESULTS

Although this project was started to analyze the jobs, there were also some results that helped clean up the Development environment, and raised some questions on the consistency of the metadata.

There were several User Generated Transformation that were not being used at all. These may be leftovers from experiments that may have had their benefits, but now they can at least moved to a folder named “Obsolete”, or completely deleted.

We also found a set of jobs that did not have a folder in the metadata. This means that they cannot be found through the Folders tab in DI Studio, but they do turn up using the Search utility. We are not sure how this could originate – perhaps an *import metadata* action gone wrong.

TERMINOLOGY: DEPLOYING, FLOWS AND SCHEDULING

Before launching into the next paragraphs we want to define a few terms we use. This is because we feel the way SAS uses these terms in the documentation and in the menus is not very consistent.

By **deploying** a job we refer to generating the code and exporting it to a .sas file in a specified location. In DI Studio this is done by selecting *Scheduling* ► from the context menu and then *Deploy…*
This creates a DeployedJob object in DI Studio which knows the location, the name of the file and the command to run it (important if there are different workspace servers).

When a job has been deployed it can be used in a **flow**, which defines how the jobs are linked together.

The flow can **scheduled**, using the LSF scheduler the SAS Management Console can communicate with, or any other scheduler (we use UC4). The scheduling mechanism can start jobs at certain dates and times, or every few minutes (perhaps only if the previous one has finished), or if a certain file is present on a certain location.
The scheduling itself is beyond the scope of this paper.

DEPLOYMENT

When one looks at the way a job is being used, one normally would expect a job in the Production environment to fall in one of three categories:

- It has been deployed, and the deployed job is part of a scheduled flow.
- It has *not* been deployed, but the job is included in another job.
- The job was meant for initializing new tables, or for converting existing tables, or a task like that, and was meant to be run only once from within DI Studio (if that is allowed in your situation).

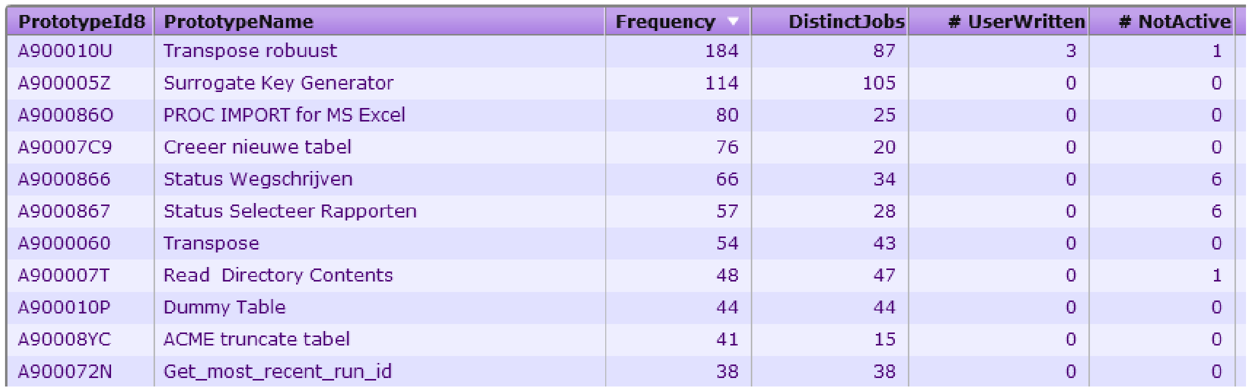
If a job has been deployed twice, that is a reason for further examination.
A complicating factor is that deployed jobs can be part of more than one flow, and flows can be part of other flows. We found some jobs to be part of four different flows, directly or indirectly.

SAS VISUAL ANALYTICS

The data from *Continuous Monitoring* and the data extracted from the metadata, about jobs, steps and transformation, was both loaded in SAS Visual Analytics. A few salient results are shown in the following screenshots.

Transformations: frequency of use

We were interested in the number of times transformations are being used across the jobs in the Production environment. The table in Figure 5 shows the most frequently used transformations:



PrototypeId8	PrototypeName	Frequency	DistinctJobs	# UserWritten	# NotActive
A900010U	Transpose robuust	184	87	3	1
A900005Z	Surrogate Key Generator	114	105	0	0
A900086O	PROC IMPORT for MS Excel	80	25	0	0
A90007C9	Creeer nieuwe tabel	76	20	0	0
A9000866	Status Wegschrijven	66	34	0	6
A9000867	Status Selecteer Rapporten	57	28	0	6
A9000060	Transpose	54	43	0	0
A900007T	Read Directory Contents	48	47	0	1
A900010P	Dummy Table	44	44	0	0
A90008YC	ACME truncate tabel	41	15	0	0
A900072N	Get_most_recent_run_id	38	38	0	0

Figure 5. Most frequently used transformations

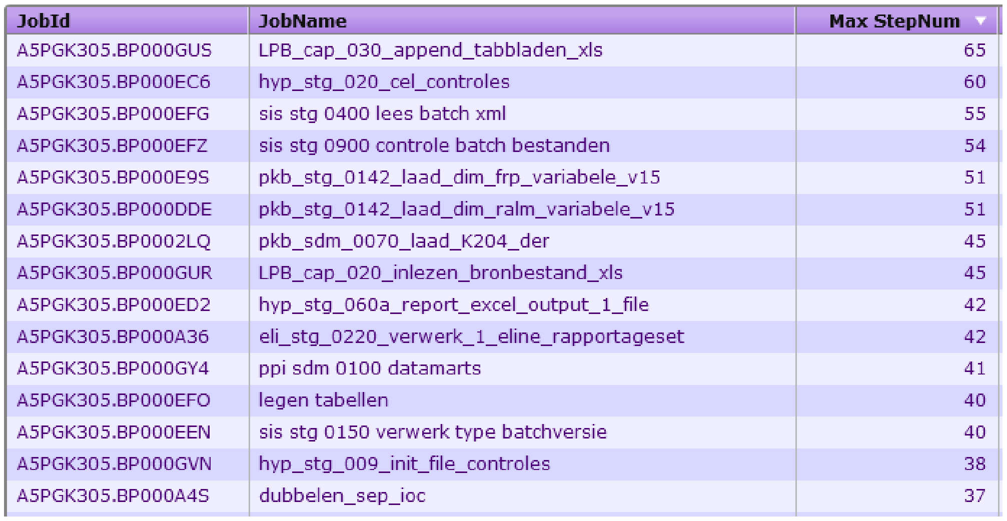
As discussed above the origin of a transformation is a <Prototype> object, hence the identifying columns are called the *PrototypeId8* and *PrototypeName*. The *PrototypeId* gives the last 8 characters of the metadata-id of the object (the first 8 characters are not relevant, as they are the identification of the repository, which is the same for all objects).

The most used transformation, “Transpose robuust”, is a variant of the standard Transpose transformation, trying to make sure that the columns of the output table conform to the metadata. It is applied 184 times, but only in 87 different jobs. In three instances the code was replaced by custom user written code – which is not encouraged. In one case the *Exclude From Run* option was selected – which is also not encouraged in jobs that have been promoted to the Production environment.

Also other transformations are repeatedly made inactive, as can be seen in the 5th and 6th observations. This is a reason for closer inspection: in which jobs had this been done, and why? The collected data makes it easy to pinpoint those jobs.

Jobs: number of steps

Our guidelines for the developers state, among many other things, that jobs should not be made to complex. As explained in the introduction, the complexity as can be determined from the number of lines of code is not the best measure. The table in Figure 7 makes it possible to look at the number of steps in a job:



JobId	JobName	Max StepNum
A5PGK305.BP000GUS	LPB_cap_030_append_tabbladen_xls	65
A5PGK305.BP000EC6	hyp_stg_020_cel_controles	60
A5PGK305.BP000EFG	sis stg 0400 lees batch xml	55
A5PGK305.BP000EFZ	sis stg 0900 controle batch bestanden	54
A5PGK305.BP000E9S	pkb_stg_0142_laad_dim_frp_variabele_v15	51
A5PGK305.BP000DDE	pkb_stg_0142_laad_dim_ralm_variabele_v15	51
A5PGK305.BP0002LQ	pkb_sdm_0070_laad_K204_der	45
A5PGK305.BP000GUR	LPB_cap_020_inlezen_bronbestand_xls	45
A5PGK305.BP000ED2	hyp_stg_060a_report_excel_output_1_file	42
A5PGK305.BP000A36	eli_stg_0220_verwerk_1_eline_rapportageset	42
A5PGK305.BP000GY4	ppi sdm 0100 datamarts	41
A5PGK305.BP000EFO	legen tabellen	40
A5PGK305.BP000EEN	sis stg 0150 verwerk type batchversie	40
A5PGK305.BP000GVN	hyp_stg_009_init_file_controles	38
A5PGK305.BP000A4S	dubbelen_sep_ioc	37

Figure 6. Jobs with large number of steps

There appears to be a fair number of jobs with a rather large number of steps.

These all are jobs that deserve a closer inspection. The top one e.g. is depicted in Figure 7.

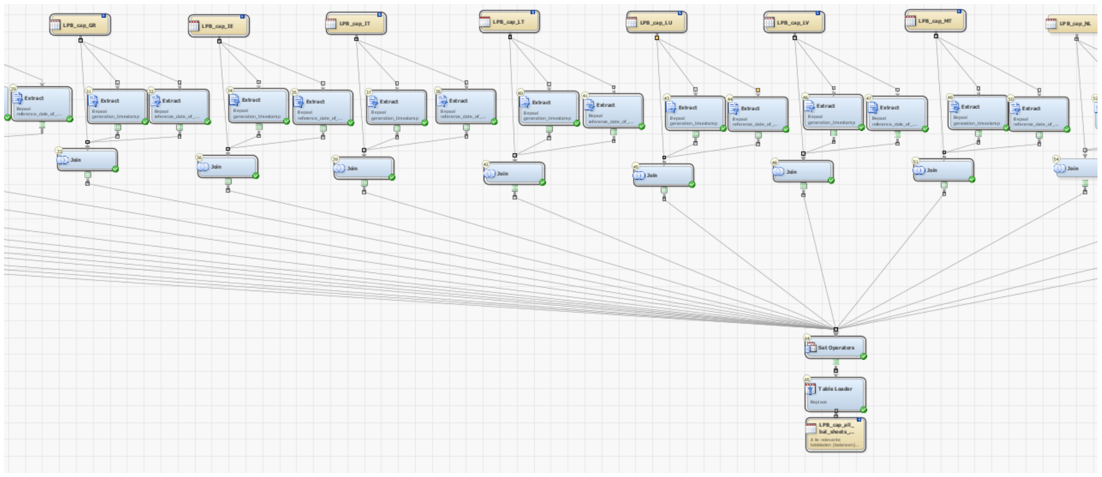


Figure 7. The job with the largest number of steps (partly)

Whether this is the best way to solve the problem at hand, is an open question – and perhaps a matter of taste. What happens is that from a lot of similar tables similar information is extracted, and that information is collected row-wise at the bottom using (at the bottom) a *Set Operators* transformation, containing dozens of *Outer Union* operators.
An alternative solution worth investigating would be a series of jobs that executes an extraction from a source and an append to the same table.
One could also create or a parameterized version of that solution, having the extraction and the append within in a loop transformation, with a table of all the source tables determining the loop. But that would be a complication in itself, and would also remove the lineage information between all the source tables and the single results table.

Average CPU time for flows and jobs

Jobs are organized in flows, and often the handling of a particular data source is separated in a set of flows, for instance:

- Capturing the data from the source;

- Transformation and organizing the data;
- Loading the data warehouse;
- Generating general data marts;
- Generating specific data marts.

Using a hierarchy, going from the set of flows maintaining a particular data stream, to the separate flows and from there to the individual jobs one can select an item for closer inspection.

The picture in Figure 8 shows the average CPU time for all jobs in all the flows that manage the AIFMD data as it developed over time in the last year. (AIFMD stands for Alternative Investment Fund Managers Directive, which has been issued by the ECB. These jobs collect the data and produce the data marts that the supervisors of DNB need to execute this directive.)

Each point is colored according to the highest return code of the jobs for that point in time. Most points are green, indicating a return code of zero, but some are orange, indicating a return code of one to three.

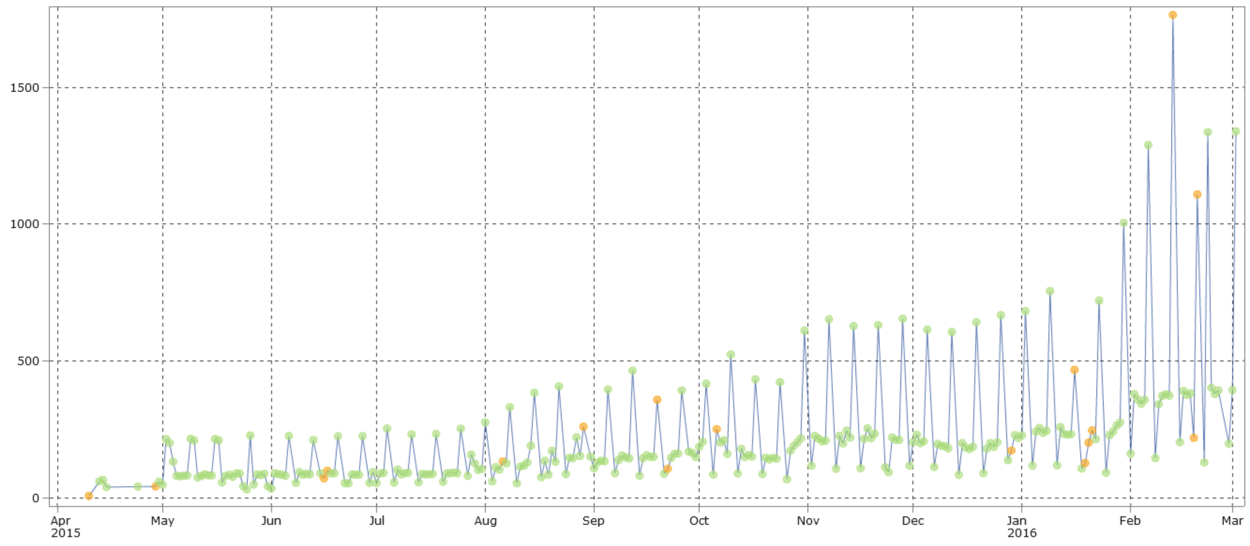


Figure 8. Average CPU time for the jobs in all AIF flows

The saw-tooth pattern invites closer inspection. But when we look at the flow that is responsible for the largest average CPU time this pattern becomes much less prominent. This can be seen in Figure 9.

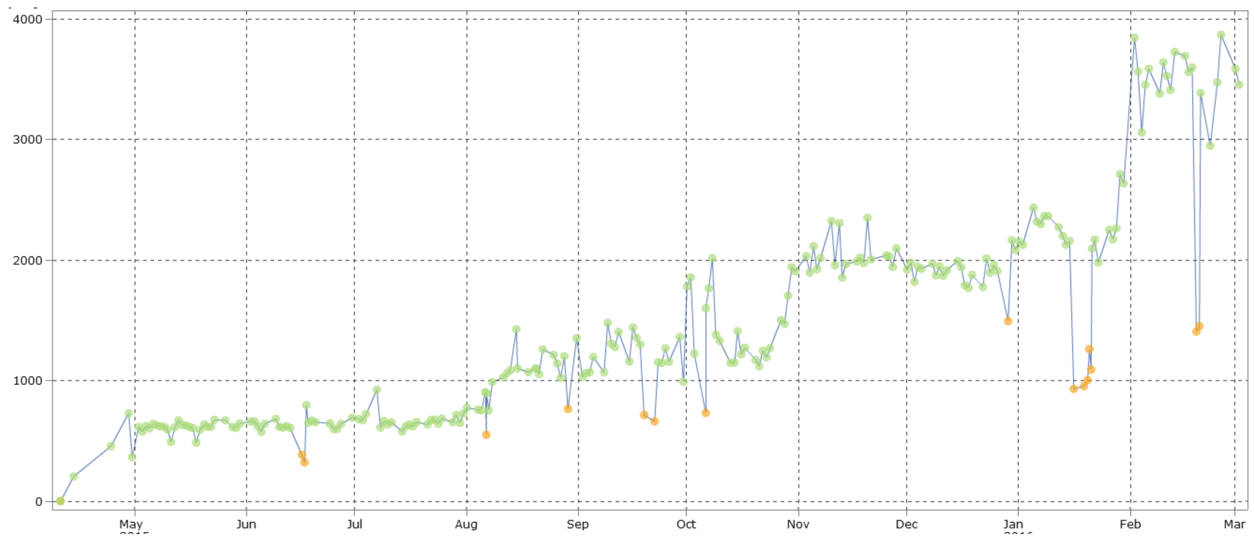


Figure 9. Average CPU time for jobs in the AIF_GDM (general data marts) flow

This picture shows two different phenomena to investigate.

At several occasions the jobs failed half way, resulting in orange dots and significantly lower CPU times. Particularly halfway January the flow aborted for almost a week. The amount of CPU-time is about half of that of a successful run, but that of course only means that the job broke down about half way.

Probably more important, particularly in the long run, are the distinctive steps upward in the amount of CPU time consumed: in the beginning of August, at the end of October and at the end of January. If this development continues this will cause problems.

CONCLUSION

The metadata of a SAS environment is of course essential for all processes. But it also can be seen as a rich source of information on the way several components of those processes are implemented, and can be used to assess the quality of that implementation on different criteria.

Although the metadata is accessible using Proc METADATA, there is no easy way to use interface. This makes that this information is not as widely used as could be.

Our current implementation gives already good pointers to processes and situations that deserve closer inspection.
Over time we hope to learn to discern the ‘false positives’ and to determine combinations of particular transformations and settings that require attention.
In this way we want contribute to the high standards of data quality asked for by De Nederlandsche Bank.

REFERENCES

[1] SAS® 9.3 Metadata Model: Reference. http://support.sas.com/documentation/cdl/en/omamodref/63903/HTML/default/viewer.htm#titlepage.htm

[2] SAS® 9.4 Metadata Model: Reference. http://support.sas.com/documentation/cdl/en/omamodref/67417/HTML/default/viewer.htm#titlepage.htm/.

[3] Metacoda: The Metadata Explorer. http://www.metacoda.com/en/products/utility-plug-ins/metadata-explorer/

ACKNOWLEDGMENTS

Several coworkers at DNB and at PW Consulting contributed to the code described in this paper. Niels Veldt developed most of the *Continuous Monitoring* system. Laurent de Walick wrote the stored processes implementing a web based metadata explorer.

CONTACT INFORMATION

We are interested in any feedback. You can contact us at:

Frank Poppe
PW Consulting
+31 6 2264 0854
Frank.Poppe@PWConsulting.nl
www.pwconsulting.nl

Joris Huijser
De Nederlandsche Bank NV
+31 6 3102 8487
J.W.Huijser@DNB.nl
www.dnb.nl