BIRD Handbook

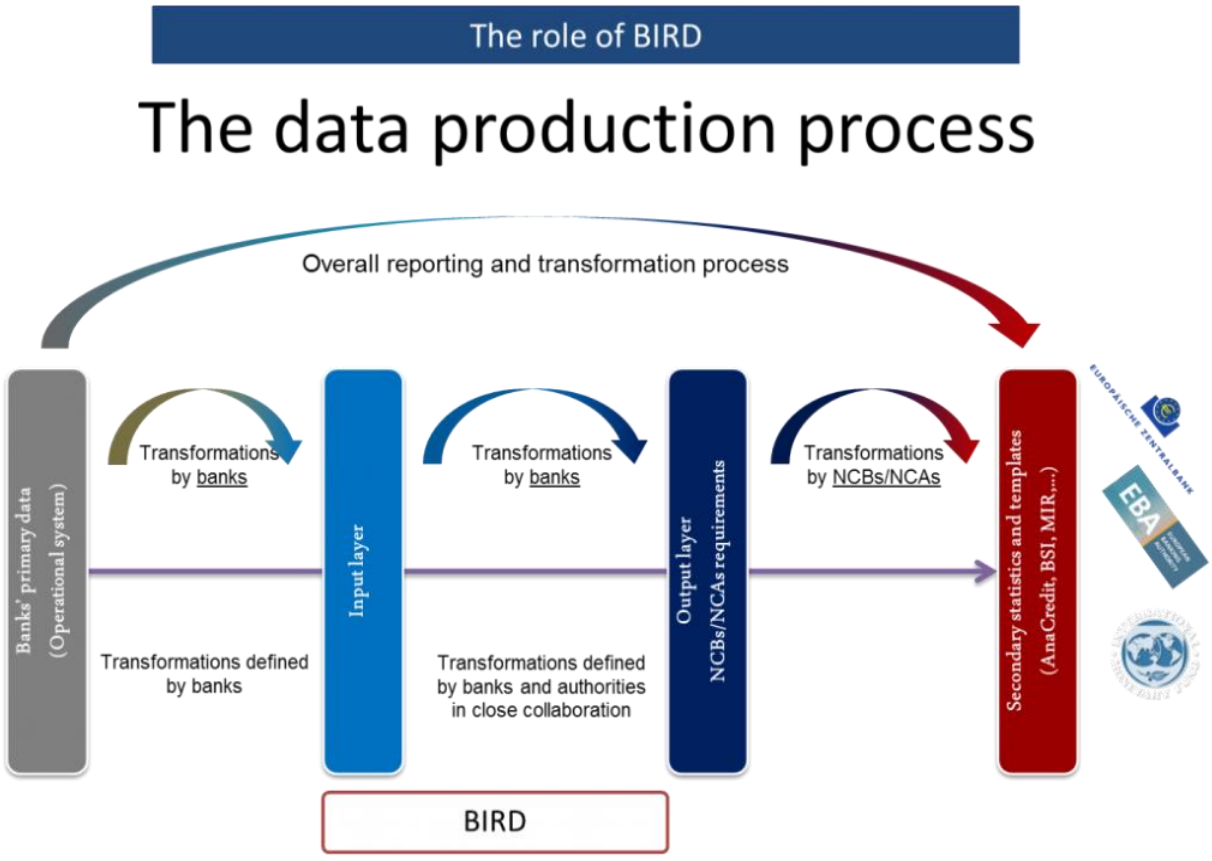
April 2017

Contents

1 About the BIRD

1.1 What is the BIRD

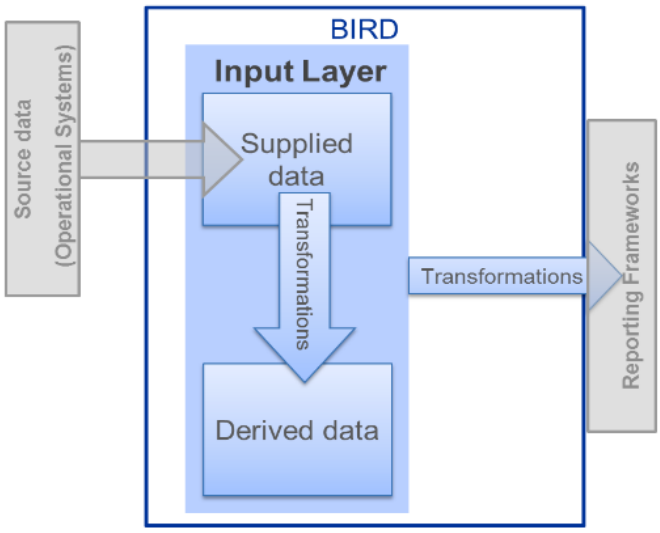
Every time a statistical or supervisory regulatory framework is updated or a new one comes into existence, each bank is left to its own devices to interpret it, extract the data from its internal systems and transform them to derive the final figure asked for in a regulation. It is not always straightforward which source data to use and how to process it to produce the number required in Legislation X, Table Y, Cell Z. The greater the misalignment among banks with regard to the meaning of specific sections within a regulatory standard, the more questionable is the quality of the output data and the more difficult are the comparisons between banks. In-depth studying of revised or new legal acts is a costly and time-consuming process for each bank to carry separately, and one for which cooperation among banks can bring significant efficiency gains.



The initiative entitled "BIRD" aims to foster such cooperation in the field of regulatory reporting thus alleviating the reporting burden for the banks and improving the quality of data reported to the authorities. “BIRD” stands for **"Banks’ Integrated Reporting Dictionary".** Its contents, published on the BIRD website, are based on a **harmonised data model** describing precisely the data which should be extracted from the banks’ internal IT systems to derive reports required by authorities. In addition to this, there are clearly defined transformation rules to be applied to the data extracted from the banks’ internal IT systems in order to produce a specific final regulatory figure. The univocal categories of the data to be extracted from the banks’ internal IT systems, the so-called **"input layer"**, together with the **transformation rules** make up the BIRD.

The purpose of the BIRD is to provide a service to the banks. The BIRD is available, as **a "public good"**, to banks and all interested parties (e.g. software houses that develop application packages for financial reporting). The adoption of the BIRD by banks is fully voluntary. It can be used as additional documentation (with respect to regulations and guidelines) or as "active dictionary" for procedures developed by banks. The BIRD represents an "input approach" because it doesn't stop at the regulatory requirements, but goes all the way back to the data in the banks' internal systems. It should be emphasised, however, that banks remain responsible for the organisation of their internal reporting system.

Figure 1: BIRD = Input Layer (Harmonised Data Model) + Transformation Rules.



1.2 What is out of the BIRD's scope?

The BIRD is **not an IT tool** itself nor is the initiative expected to provide such tool. The BIRD does not make any changes to the banks' internal IT systems. The scope of the BIRD does not cover the mappings of the data from the banks' internal IT systems to the input layer.

The BIRD is **not a regulatory act** and it is **not a new rule**. It is a transposition of the legal requirements at a more operational level. In other words, the BIRD provides a formalised representation of how the requirements set in reporting regulations may be satisfied. Its application is fully **voluntary**.

1.3 How will the BIRD be a benefit for the European banks?

The adoption of the BIRD has several **advantages**:

1. Different reports can be produced from a single input layer by applying harmonised algorithms → Lower reporting burden for the banks. Greater consistency and quality of the data. No more need to manage each mandatory data collection in a separate way.

2. Well-defined transformation rules, which include the calculations to obtain certain regulatory figures → A univocal interpretation and clarity of regulations. Enhanced compliance with the regulatory requirements.

3. Decreased time and effort to analyse and satisfy new reporting requirements → Increased efficiency and lower costs.

4. Increased awareness, understanding and interest in what is behind data (what they include, how they are produced) → Improved management and use of data.

1.4 Which statistical and supervisory regulatory frameworks are (will be) covered by the BIRD?

The intention is that several statistical and supervisory reporting requirements will be covered by the BIRD: ECB's collection of granular credit data and credit risk data (AnaCredit), ECB's Securities Holdings Statistics (SHS), ECB's Monetary Financial Institutions' Balance Sheet Items Statistics (BSI), ECB's Monetary Financial Institutions' Interest Rate Statistics (MIR), the needs of other statistics, such as the balance of payments and national accounts, the additional requirements of the Single Supervisory Mechanism and EBA's Implementing Technical Standards (ITS), which encompasses Common Reporting (COREP) and Financial Reporting (FINREP).

1.5 Cooperation between the authorities and the banks to develop the BIRD

For reasons of efficiency and effectiveness, the BIRD is carried out and maintained by a number of banks participating in the BIRD group in **close cooperation** with the European Central Bank and National Central Banks. Both banks and authorities have a very strong interest in an appropriate organisation of the information systems of banks, able to produce timely, consistent and high quality data. The production of a meaningful BIRD requires **specific knowledge of reporting agents**, which calls **for joint work and close collaboration** between the banking industry and the authorities.

2 Project Status

In 2013 the Statistics Committee (STC) of the European System of Central Banks has investigated the possibility to promote an integrated approach to supervisory and statistical data.

In 2014 an STC Groupe de Réflexion recommended the development, in close collaboration with the banking industry, of a European “input approach” model which would provide a standardised design of the banks’ internal data warehouses to support the data reporting to the authorities.

In 2015 a workshop with the industry, hosted by the ECB, has indicated as the best approach for launching the implementation of BIRD, the implementation of the new requirements related to the collection of granular credit and credit risk data (hereafter AnaCredit) as defined in Regulation (EU) 2016/867), and data on holdings of securities (hereafter SHS) as defined in Regulation (EU) No 1011/2012 as amended by Regulation (EU) 2016/1384.

Considering the limited time for implementing the AnaCredit and SHS requirements, it was decided to start with a light short term approach via a BIRD pilot for AnaCredit and SHS. The BIRD pilot started in 2015 with some volunteer National Central Banks (NCBs) and banks contacted by each participating NCB.

The Bird pilot has been successfully concluded in April 2017, with the publication of a comprehensive and detailed documentation describing in a formal language how the requirements addressed to banks with respect to the ECB Regulations on AnaCredit and Securities Holdings Statistics (for banking groups) may be satisfied by using data from the banks’ internal systems. The documentation is available to banks and all interested parties, as a public good and free of charge.

Based on the results of the BIRD pilot, it will be discussed the future of the BIRD and how to proceed with the initiative.

3 BIRD methodology

3.1 Introduction

The BIRD methodology refers to the technique used to provide a formal documentation of the BIRD proposed data model.

The description of the BIRD proposed data model is composed of:

- Description of datasets
- Transformation process
- Technical guidelines on how to populate the datasets and implement transformations.

The BIRD is documenting a dynamic data process, where some data are provided as input, validated and subsequently transformed into other datasets until the final datasets, which are those that need to be reported to the authorities, are generated. Section 2 of this document describes the high-level BIRD data process.

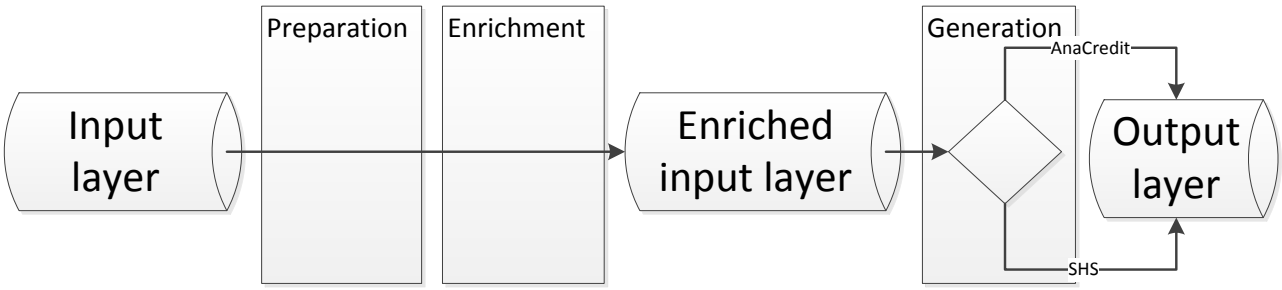
The BIRD provides a formal description of both the datasets and the validation and transformation rules. Section 3 introduces the methodology for the formal description of the BIRD datasets, while section 4 deals with the validation and transformation language.

The technical guidelines on how to populate datasets, which include the conceptual model of the relation between the cubes in the form of an Entity Relationship Model, constitute a separate document, which is included in the BIRD website.

3.2 High-level BIRD data process

A system following the BIRD process would normally start by feeding the input cubes from banks' internal IT systems, following the structure of the input cubes defined by the BIRD. The system would then run the validations of the input cubes, and depending on the output of the validations, it would create enriched cubes. The enriched cubes contain the relevant information for generating the final reports, including input information but also new information derived from that input. From the enriched cubes, all reporting requirements would be generated. The validation, enrichment and generation would be guided by the transformation rules (further explained in Section "Transformation rules in the BIRD model"). The reporting requirements are also part of the BIRD documentation, in the form of output cubes, although their definition is out of the scope of the BIRD; the BIRD simply includes the translation of the output cubes into its data model.

The process described in the BIRD is divided in three phases, of which two are common to all output frameworks to be generated, and one will be specific for each reporting framework.



The enrichment phase finishes when writing the enriched input layer. The enriched input layer is also described in the BIRD dictionary as a group of datasets.

The detailed sub-phases of the transformation process are further described in Annex IV.

3.3 Description of the datasets

The description of the datasets in the BIRD is provided following the SMCube methodology, which is a methodology developed with the objectives of:

- Enabling the description within one dictionary of all types of datasets (registries, dynamic data for supervisory purposes, for monetary purposes…)
- Facilitating the integration of dictionaries developed with different methodologies (like DPM/XBRL or SDMX) in one single dictionary by keeping a high degree of compatibility with the other methodologies

The version 1.0 of the SMCube methodology is finalised. The SMCube information model will be provided as soon as possible. A non-technical introduction to the main concepts of the methodology is provided in Annex I of this handbook.

The BIRD pilot is using a simplified version of the methodology, where some artefacts (like those in the rendering and provisioning packages) and features (like historicizing and extensibility) have been removed, since for the time being they are not going to be used. The formal description of the BIRD datasets is provided in an Access database, which can be downloaded in the BIRD website.

3.4 Transformation rules in the BIRD model

Transformation rules represent a description of logical operations applied to objects of the BIRD database to create new / additional information.

Transformations rules are organised, following the VTL model, consisting of Transformation schemes, Functions, Rulesets and procedures. One Transformation scheme can contain one or many transformations.

Transformations constitute the basic element of the calculations, and consist of a statement assigning the outcome of an expression to a VTL element. Transformations have thus (except for procedure calls) the following structure:

```
transformationResult := expression
```

Transformations are grouped into transformation schemes, which are sets of transformations aimed at obtaining some meaningful results for the user. Transformation schemes group transformations that share some functional or business purpose. An example of transformation scheme is the derivation of enterprise size. This transformation scheme is formed by several different transformations aimed to give as a result the final enterprise size.

Transformations are defined in verbal and formal form. The verbal description ("Natural language") represents the business perspective trying to avoid specifications regarding the technical implementation. The formal description is based on the "Validation and Transformation Language (VTL)" with minor adaptions to improve usability / readability by the end user and aims at a technical description of the transformation rules independent of the technical implementation.

Regarding the formal description, please note that due to the ongoing developments in the Task force for the Validation and Transformation Language (VTL) and the publication of VTL version 1.1 at the end of June 2017, changes to the syntax are possible. Currently the VTL task force is reviewing the comments received before the final release.

Transformation schemes are classified in the following four types:

i. **Validation rules:** Transformations that evaluate a logical condition that data should comply with. Validations are subdivided in:
    a. Consistency: Check the consistency among variables (for instance, the inception date has to be earlier than the reference date)
    b. Completeness: Check that all related information exists (for instance, for each loan there should be at least one debtor).
    c. Integrity: Check that, for one instance of one object, there is a related instance in another object. This includes referential integrity validations (for instance, if there is an issuer of a security, this issuer has to exist in the *Counterparties* cube).
    d. Uniqueness: Checks that identifiers used in separate cubes are unique (for instance, that an *Instrument unique identifier* can only be used for one instrument).
ii. **Derivation rules:** Transformations that create new variables from existing data.
iii. **Generation rules:** Transformations focused on the data preparation based on the formalities described for each output framework.
iv. **Technical rules:** Transformations required for technical purposes, in order to have a complete description of the transformation process, but are not relevant for business users.

Annex II provides a technical description of the model used in the database to represent VTL as well as an overview of the most important VTL functionality used in BIRD transformations.

Annex I: Non-technical introduction to SMCube methodology

SMCube is a methodology for defining metadata to describe datasets. Its pivotal role is in defining cubes, which define the structure of a dataset, intended as a set of data organised as a table with fields (columns) and records (rows).

Table 1 provides an example of a dataset with information related to securities.

Table 1: Securities dataset.

ISIN	Type of instrument	Inception date	Legal final maturity date	Subordinated debt	Nominal value	Fair value	Accrued interest	Currency
YY7365 923618	F_32	16/06/2015	16/06/2035	1	1.000	1.020	10	EUR
ZZ3941 829354	F_511	05/10/2015			6.000	6.050		USD

The minimum information required to describe the structure of the dataset can be summarised with the following three questions:

1. What are the fields (columns) of the dataset?

In the SMCube methodology, the fields of a cube are called variables. The variables are defined independently of the cube, so that a cube will refer to as many variables as it needs; but the same variable may be referred by many cubes, thus allowing for reusability of concepts.

Referring to Table 1, the variables are: “ISIN”, “Type of instrument”, “Inception date”, “Legal final maturity date”, “Subordinated debt”, “Nominal value”, “Fair value”, “Accrued interest” and “Currency”.

2. What are the allowed values for each field?

In the SMCube methodology, each variable is defined on a domain, which states the allowed values for the variables. Domains can be enumerated, if they provide a list with the allowed values, or non-enumerated, if they provide a data type.

In the example dataset, some variables are defined on non-enumerated domains, like the “ISIN” (string domain), the “Inception date” (date domain) or the “Fair value” (monetary domain). Three variables are defined on enumerated domains: “Type of security”, “Subordinated debt” and “Currency”; for each of these variables, it is necessary to provide the list of allowed values. The possible values for an enumerated domain are called members in the SMCube methodology. For each member, the dictionary shall provide additional information, like the description of the concept (e.g. what does F_32 mean?)

But one variable can have different allowed values in different datasets. For instance; the example dataset refers to securities, so the allowed values for “Type of instrument” should only refer to securities, but in a dataset covering more instruments the list of allowed values could be enlarged. In order to allow reusing the variables, the domain will contain all the possible members for variables.

The same holds true for non-enumerated domains, where the subdomain will serve to constrain the allowed values, even if they are not enumerated. For instance, the variable “Fair value” is a monetary variable, but in the context of the securities dataset, its allowed values may be limited to only positive amounts.

In the context of a cube, one variable has to be associated to a concrete subdomain, i.e. the subset of the domain that is allowed in the context of the cube.

3. What is the role of one field within one dataset?

One of the most relevant aspects of the structure of the dataset is what the identifier of the record is or, in other words, what combination of fields makes a record unique. In the example dataset, if nothing is said regarding the structure, applying some business knowledge one may conclude that each record is uniquely identified by the ISIN. But it is also true that one security with one ISIN may be denominated in more than one currency, so if the variable currency is referred to the denomination of the security, it could be the case that more than one record per ISIN is allowed, as long as the currency in each record is different. Thus in order to get a thorough understanding of the described dataset, the role of the variables has to be explicit. In the SMCube methodology, variables that serve as identifiers of the records take the dimension role.

Let’s suppose that in the example the only dimension is the ISIN. The rest of variables may have also different roles, depending on the variables to which they add information. Let’s take again the currency variable as an example, once it is known that it is not a dimension. Without additional information, a user could have two different interpretations of it: It may be adding information to the ISIN (i.e. to the combination of dimensions, in this case only one) specifying the currency in which the currency is nominated, or it may be referred to the monetary amounts, specifying the currency in which the amounts are nominated. In the SMcube methodology, the variables that add information to the combination of dimensions take the observation value role, while the variables that add information only to one variable of the dataset take the attribute role.

It is worth highlighting that one variable can take different roles in different datasets. For instance, the example dataset is disaggregated to the level of security. Another dataset may contain, for one reporting institution, the total securities broken down by instrument.

Table 2: Aggregated dataset.

Reporting institution	Type of instrument	Nominal value	Fair value
ABC	F_32	1.000	1.020
ABC	F_511	6.000	6.050

In the context of this dataset, the type of instrument is a dimension, since it is part of the identifier (one record is uniquely identified by the combination of “Reporting institution” and “Type of instrument”).

Summary

With the SMCube methodology, one cube serves to define the structure of a dataset. One cube is a set of variables, for which the allowed values are specified by a subdomain, and that have a role in the context of the cube. The SMcube definition of the dataset 1 could be summarised in the following table:

Table 3: Cube describing the securites dataset.

Variable	Subdomain	Role
ISIN	12-character alpha-numerical code	Dimension
Type of instrument	Types of instruments for dataset 1	Observation value
Inception date	All dates	Observation value
Legal final maturity date	All dates	Observation value
Subordinated debt	Boolean including not applicable	Observation value
Nominal value	Positive monetary amounts	Observation value
Fair value	Non-negative monetary amounts	Observation value
Accrued interest	Non-negative monetary amounts	Observation value
Currency	ISO 4217 codes	Observation value

Annex II: Transformations

The representation of transformations in the BIRD database is based on the SDMX information model (see section II, 13.2 Model – Inheritance View, 13.2.1 Class Diagram).

According to the VTL mode, a Transformation scheme is an ordered list of transformations. Therefore such a transformation scheme contains one or many transformations (i.e. one line of valid VTL code). The SDMX model specifies that transformations can contain one or many transformation nodes (i.e. the components of this line of valid VTL code). Therefore a transformation element is a constant, an operation or a BIRD model object (i.e. a variable, cube, etc.). In case the transformation element represents an operation such a transformation element itself can have a relation to one or many transformation elements.

The BIRD database contains the complete information about transformation schemes in the sense that not only the decomposition of each transformation scheme into its transformations but also the decomposition of transformations into its transformation nodes according to the SDMX information model is stored in the database.

The next sections (“Example”, “Representation in the database”) explain the relation between transformation schemes, transformations and transformation nodes and their representation in the database. In the section “New VTL artefacts” we also describe functionality of VTL that is used regularly in various transformation schemes.

Example

The following example will try to clarify the current status of the representation of transformations in the BIRD database:

Let’s assume we have a (database-)table named “coordinates” containing the columns (i.e. variables) x and y which (clearly) relate to some coordinate system. Our transformation scheme’s goal is to derive a new variable distance for all records where x and y are greater than or equal to 0 defined in the following way:

$distance = sqrt(x * x + y * y)$.

Using VTL syntax we would write the following lines:

```
/*extract all records from the (database-)table coordinates and store
the result in a dataset named "coordinates"*/

coordinates := get("coordinates");

/*extract all records from the dataset "coordinates" where x and y are
greater or equal to zero, keep only x and y and store the result in a
dataset named "result"*/
```

```
result := coordinates [filter (x >= 0 and y >= 0), keep (x, y)];

/*apply a calculation on the dataset "result" which takes the square
root of the sum of x squared and y squared and stores the result in a
new column (i.e. variable) named "distance"*/

finalResult := result [calc sqrt (x * x + y * y) as "distance"];
```

The tree structure with respect to the second line can be illustrated as follows:

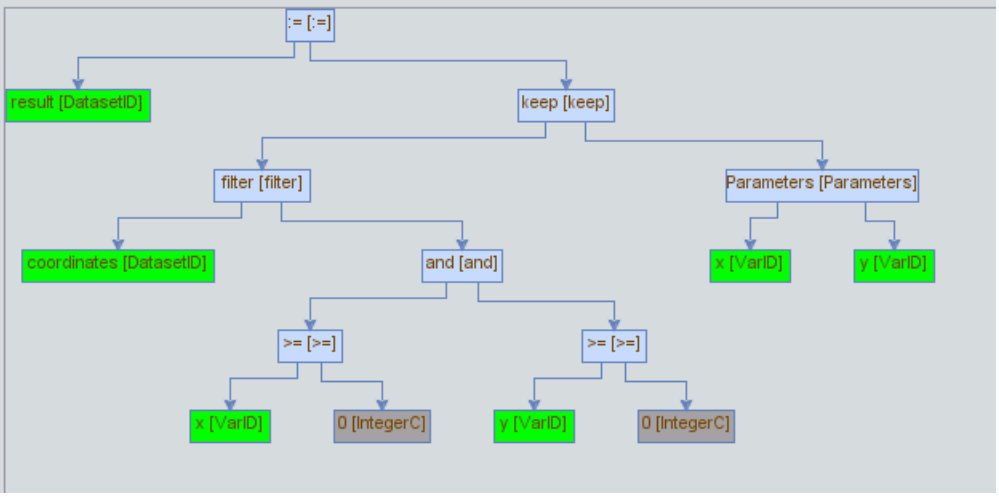


FIG 1: tree structure representation of *result := coordinates [filter (x >= 0 and y >= 0), keep (x, y)];*

The term written in brackets is the type of transformation element which can be used to identify constants and BIRD model objects (i.e. variables, cubes, etc.).

Please note that the Boolean condition applied to the filter operator (i.e. “x>=0 and y>=0”) is completely decomposed into its components (i.e. transformation elements) in a structured way in the sense that the Boolean condition can be reengineered from this tree structure.

The decomposition of transformation into its transformation elements supports specific implementations of these transformations. For example in case of a SQL implementation we could apply the following mappings:

- := → CREATE VIEW _____ AS
- Filter → WHERE
- Keep → SELECT

Walk the tree and create the corresponding line of SQL code:

```
CREATE VIEW result AS SELECT x, y FROM coordinates WHERE x >= 0 AND
y >= 0;
```

Please note that – in order to generate such an SQL statement – one must additionally rearrange the nodes of the tree according to the SQL syntax. Please also note that the elements after each keyword (i.e. CREATE VIEW, SELECT, FROM, WHERE) are similar to the elements represented in the tree structure.

For the sake of completeness you find the tree representation of the first and second line here:

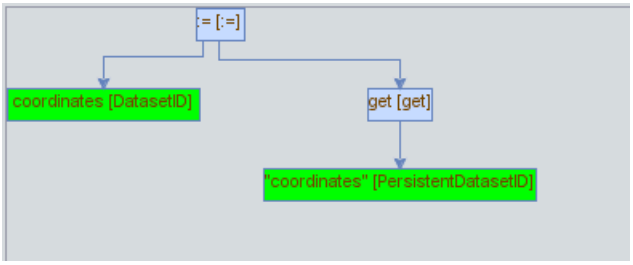


FIG 2: tree structure representation of *coordinates := get("coordinates");*

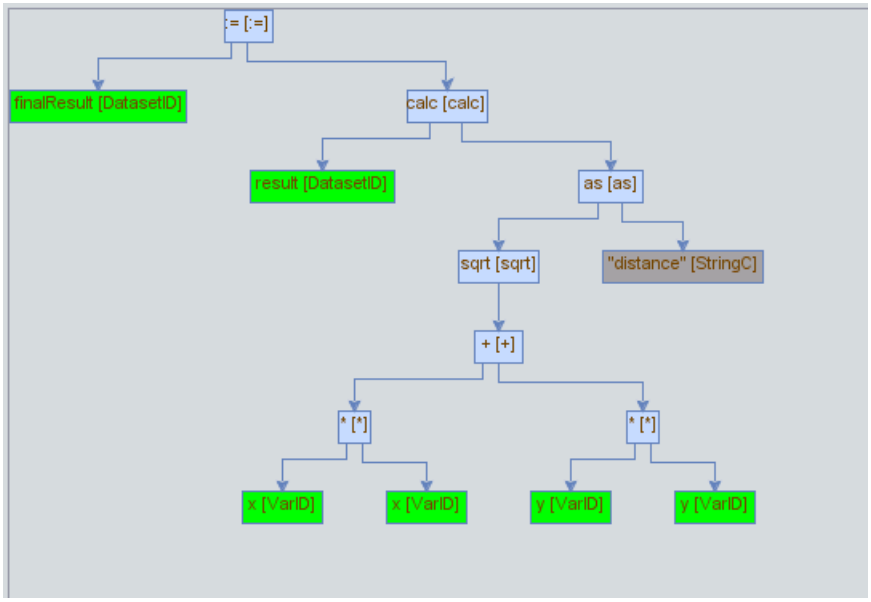


FIG 3: tree structure representation of *finalResult := result [calc sqrt(x * x + y * y) as "distance"];*

Representation in the database

The transformation scheme is stored in the table TRANSFORMATION_SCHEME:

SCHEME_ID	EXPRESSION	DESCRIPTION	NATURAL_LANGUAGE

TEST_SCHEME	/*extract all records from the (database-)table coordinates and store the result in a dataset named "coordinates"*/
	coordinates := get("coordinates");
	/*extract all records from the dataset "coordinates" where x and y are greater or equal to zero, keep only x and y and store the result in a dataset named "result"*/
	result := coordinates [filter (x >= 0 and y >= 0), keep (x, y)];
	/*apply a calculation on the dataset "result" which takes the square root of the sum of x squared and y squared and stores the result in a new column (i.e. variable) named "distance"*/
	finalResult := result [calc sqrt (x * x + y * y) as "distance"];
		Transformation scheme example	Transformation scheme deriving the distance between x and y

Please note that this is a reduced version of the original table, presented for illustrative purposes.

Each individual transformation is stored in the TRANSFORMATION table:

TRANSFORMATION_ID	EXPRESSION	SCHEME_ID	ORDER
156430	/*extract all records from the (database-)table coordinates and store the result in a dataset named "coordinates"*/ coordinates := get("coordinates");	TEST_SCHEME	0
156435	/*extract all records from the dataset "coordinates" where x and y are greater or equal to zero, keep only x and y and store the result in a dataset named "result"*/ result := coordinates [filter (x >= 0 and y >= 0), keep (x, y)];	TEST_SCHEME	1
156451	/*apply a calculation on the dataset "result" which takes the square root of the sum of x squared and y squared and stores the result in a new column (i.e. variable) named "distance"*/ finalResult := result [calc sqrt (x * x + y * y) as "distance"];	TEST_SCHEME	2

Using the SCHEME_ID we can connect these transformations with the related transformation scheme (which is similar to stating that “these transformations are children of the transformation scheme with SCHEME_ID ”TEST_SCHEME”).

All Transformation elements are stored in the table TRANSFORMATION_NODE:

TRANSFORMATION_ID	NODE_ID	EXPRESSION	TYPE_OF_NODE	LEVEL	PARENT	ORDER
156430	156431	:=	OperatorNode	0		
156430	156432	coordinates	ReferenceNode	1	156431	0
156430	156433	get	OperatorNode	1	156431	1
156430	156434	"coordinates"	ReferenceNode	2	156433	0
156435	156436	:=	OperatorNode	0		
156435	156437	result	ReferenceNode	1	156436	0
156435	156438	keep	OperatorNode	1	156436	1
156435	156439	filter	OperatorNode	2	156438	0
156435	156440	coordinates	ReferenceNode	3	156439	0
156435	156441	and	OperatorNode	3	156439	1
156435	156442	>=	OperatorNode	4	156441	0
156435	156443	x	ReferenceNode	5	156442	0
156435	156444	0	ConstantNode	5	156442	1
156435	156445	>=	OperatorNode	4	156441	1
156435	156446	y	ReferenceNode	5	156445	0
156435	156447	0	ConstantNode	5	156445	1

156435	156448	Parameters	OperatorNode	2	156438	1
156435	156449	x	ReferenceNode	3	156448	0
156435	156450	y	ReferenceNode	3	156448	1
156451	156452	:=	OperatorNode	0		
156451	156453	finalResult	ReferenceNode	1	156452	0
156451	156454	calc	OperatorNode	1	156452	1
156451	156455	result	ReferenceNode	2	156454	0
156451	156456	as	OperatorNode	2	156454	1
156451	156457	sqrt	OperatorNode	3	156456	0
156451	156458	+	OperatorNode	4	156457	0
156451	156459	*	OperatorNode	5	156458	0
156451	156460	x	ReferenceNode	6	156459	0
156451	156461	x	ReferenceNode	6	156459	1
156451	156462	*	OperatorNode	5	156458	1
156451	156463	y	ReferenceNode	6	156462	0
156451	156464	y	ReferenceNode	6	156462	1
156451	156465	"distance"	ConstantNode	3	156456	1

This structure supports easy access to the components of each Transformation. If, for example, we are interested in the operators that are used in the second line (*result := coordinates [filter (x >= 0 and y >= 0), keep (x, y)];* compare FIG 1) we simply select all rows where the TRANSFORMATION_ID equals 156435 and restrict the result to those records where the TYPE_OF_NODE equals OperatorNode. The result reflects the blue squares in FIG 1.

Please note that not only Transformations are represented in this structure but also Functions, Datasets and Procedures.

New VTL artefacts

Procedures

Procedures are aimed at automating common processing tasks, and can be used as a means for shortening the code by replacing common processing tasks with a procedure call.

Procedures take input and output arguments and describe the set of transformations performed with those arguments.

The following example shows a procedure for checking the identifiers that are present in a cube (FRGN_CB) but are not present in another cube (PRMRY_CB).

```
define procedure PRCDR_RFRNTL_INTGRTY(input FRGN_CB as dataset, input
FRGN_VRBL as string, input PRMRY_CB as dataset, input PRMRY_VRBL as string,
input VLDTN_ID as string, output RSLT as dataset) {
```

```
        /*extract a set of Foreign variable (FRGN_VRBL), rename to ID, store
        in dataset Foreign identifiers (FRGN_IDS)*/

        FRGN_IDS := FRGN_CB[keep (FRGN_VRBL), rename FGN_VRBL as "ID" ];

        /*extract a set of Primary variable (PRMRY_VRBL), rename to ID, store
        in dataset Primary identifiers (PRMRY_IDS)*/

        PRMRY_IDS := PRMRY_CB[keep (PRMRY_VRBL), rename PRMRY_VRBL as "ID" ];

        /*calculate the set difference between Foreign identifiers (FRGN_IDS)
        and Primary identifiers (PRMRY_IDS)*/

        RSLT :=  setdiff (FRGN_IDS, PRMRY_IDS);

        /*rename ID to FRGN_IDS, make VLDTN_ID a measure variable*/

        RSLT := RSLT [rename (ID as "FRGN_IDS", VLDTN_ID role Measure)];

}
```

The procedure can be called afterwards with concrete arguments:

```
call PRCDR_RFRNTL_INTGRTY(TRNSCTNS_CNTRPRTS, CNTRPRTY_ID, CNTRPRTS,
CNTRPRTY_ID, V_TRNSCTNS_CNTRPRTS_ID);
```

Functions

VTL allows extending the available operators by defining functions. Functions take as input some variables, and give as a result a predefined calculation. Currently the BIRD uses functions as linear maps n:1 maps creating one output value while taking into account n input parameters.

The following example shows a function to calculate the carrying amount from the required input variables:

```
/*map: (Accounting classification, Fair value, Gross carrying amount
excluding accrued interest, Accrued interest, Fair value changes due to
hedge accounting, Accumulated impairment) → Carrying amount*/

create function D_CRRYNG_AMNT(ACCNTNG_CLSSFCTN, FV,
GRSS_CRRYNG_AMNT_E_INTRST, ACCRD_INTRST, FV_CHNG_HDG_ACCNTNG,
ACCMLTD_IMPRMNT)  {

returns

if (ACCNTNG_CLSSFCTN in ("2", "4", "8", "41")) then FV

elseif (ACCNTNG_CLSSFCTN in ("6", "14")) then (GRSS_CRRYNG_AMNT_E_INTRST +
```

```
ACCRD_INTRST -  ACCMLTD_IMPRMNT + FV_CHNG_HDG_ACCNTNG)

else null

as integer}
```

This function can be then used to derive new data:

```
RESULT := CUBE [calc D_CRRYNG_AMNT(ACCNTNG_CLSFCTN, FV,
GRSS_CRRYNG_AMNT_E_INTRST, ACCRD_INTRST, FV_CHNGS_HDG_ACCNTNG,
ACCMLTD_IMPRMNT) as "CRRYNG_AMNT" role Measure];
```

The line illustrated above adds a column named “CRRYNG_AMNT” to the dataset CUBE, where the value of this new column for each row is determined by the function D_CRRNG_AMNT, and stores the result in a dataset named RESULT.

Rulesets

Rulesets define validation rules between variables that have to be applied to each individual record of a given dataset. Rulesets take as input the variables to be validated, and contain at least one consistency rule that the validations need to comply with. Each rule has two conditions (introduced by the clauses when and then), and the validation will be satisfied if both conditions are satisfied.

As an example, the following ruleset includes two consistency rules between the variables accounting classification and accumulated changes in the fair value due to credit risk

```
define datapoint ruleset DR_ACCMLTD_IMPRMNT1(ACCNTNG_CLSSFCTN,
ACCMLTD_IMPRMNT) {

RL1:

      when ACCNTNG_CLSSFCTN in ("2", "4", "41")

      then isnull(ACCMLTD_IMPRMNT)

      errorcode("Instruments classified as 'IFRS: Financial assets held for
      trading (2)', 'IFRS: Financial assets designated at fair value through
      profit or loss (4)' or 'IFRS: Non-trading financial assets mandatorily
      at fair value through profit or loss (41)' are not subject to
      impairment");

RL2:

      when ACCNTNG_CLSSFCTN in ("6", "8", "14")

      then not isnull(ACCMLTD_IMPRMNT)

      errorcode("For instruments classified as 'IFRS: Financial assets at
```

```
        amortised cost (6)', 'IFRS: Financial assets at fair value through
        other comprehensive income (8)', 'IFRS: Cash balances at central banks
        and other demand deposits (14)' the 'Accumulated impairment' should
        not be null");

}
```

The ruleset can then be used with the check operator:

```
VALIDATION_RESULT := check (DATASET, DR_ACCMLTD_IMPRMNT1);
```

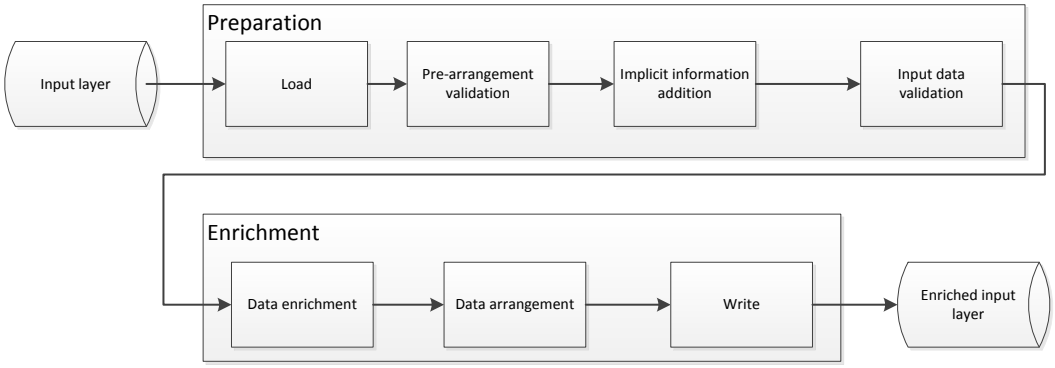
Transformation parser

We developed a parser for VTL in order to visualize tree structures and support the production of the output data model required for the BIRD database. The parser is written in Java, and available in GitHub.

Annex III: Transformation process

Sub-phases of the transformation process

The common phases of the transformation process phases can be further broken down in sub-phases:



- Preparation phase:
  - **Load**: This sub-phase includes the technical transformations aimed to get the data from the input cubes.
  - **Pre-arrangement validation**: Includes the validations that have to be done before creating unions of cubes (those belonging to the entities “Loans” and “Protection received”). Concretely, the validations that check identifiers that have to be unique across more than one cube.
  - **Implicit information addition:** Includes the derivations that add variables that have an implicit value in the input. It also includes the technical transformations for merging all the loans cubes, and all the protection received cubes.
  - **Input data validation**: Includes all the remaining validations referred to the input layer information.
- Enrichment phase:
  - **Data enrichment**: Includes all the derivation rules that serve to create new information that is not implicit.
  - **Data arrangement**: Includes some technical transformations to arrange data into the enriched input layer model.
  - **Write**: Include technical transformations to write the resulting information into the enriched input layer cubes.

Dependencies of transformation schemes

Due to the fact that most of the transformation schemes depend on other schemes in the sense that they use datasets that are generated in other schemes we provide a graphical illustration of such dependencies for each transformation scheme following the header “Scheme dependencies” in the “Natural language” section. These dependencies can be computed from the transformation content in the database (i.e. the tables TRANSFORMATION_SCHEME, TRANSFORMATION, TRANSFORMATION_NODE).

Please note that we implemented some restrictions to the transformation schemes that are taken into account when computing these dependencies; first we do not include validation and put schemes in the dependency tree

and second, such a dependency tree does not contain any duplication of related schemes (although multiple schemes in such a tree may depend on the same transformation scheme).