Introducing MapLan to Map Banking Survey Data into a Time Series Database

Dr. Manuel Günter
Swiss National Bank
+41 (0)44 631 3514
Manuel.Guenter@snb.ch

ABSTRACT

In order to fulfill its monetary policy function, the Swiss National Bank (SNB) collects statistical data on the economy. The SNB stores results of the regularly held surveys in a specialized database (primary), ordered by surveys and survey forms. After validation the data has to be transferred in another specialized database (secondary) where it can be accessed by economists. The secondary database keeps the data in time series that are hierarchically arranged by statistical taxonomies. The data transfer from the primary to the secondary database feeds 1.5 million time series. Mapping and transformation logic was hard-coded in legacy programs. They were cumbersome to manage and intransparent to the economists in charge. In this paper we describe a novel approach called MapLan, a Java-based data mapping system featuring a domain specific language. The MapLan system not only performs the data transformation and mapping, it also produces complete data lineage information. This paper shows in practice that domain specific languages are an efficient tool to solve two pressing data mapping and transformation problems of statistical databases. One problem is that of mapping the large and heterogeneous schemas of statistical databases in an efficient and manageable way. The other problem is the business need for complete data lineage of the target time series.

1. INTRODUCTION

The Swiss National Bank conducts 30 statistical surveys on a regular basis (monthly, quarterly, and yearly). Each survey consists of one or more forms. All in all SNB surveys use roughly 150 forms [1]. The layout of a form is usually a table. Each form holds up to several thousand positions. More than 300 banks, 2000 companies, and 300 collective capital investment companies deliver millions of positions (usually numbers) per year. Incoming survey data is stored in a specialized primary database that stores it in the survey structure. A typical data locator in the primary database thus consists of the ID of the sender (aka subject), the date of the survey, the form name, the row number, and the column number.

The secondary database is a separate system. This makes sense because it has to meet different requirements. The economists and data analysts that work with the statistical data expect validated figures ordered into categories of economic domain (taxonomy). They work with time series so that they can identify trends and calculate prognosis. Moreover, the Swiss National Bank produces publications that also show the statistical data in the form of time series (See for example [3]). In order to anonymize the data, only aggregated time series are published. E.g. the sum of the assets of all Swiss regional banks or cantonal banks is published but not the assets of a given single bank. So to meet all these requirements by the economists and the publication process, the SNB uses a specialized time series database (secondary database). In order to transfer the data from the primary to the secondary database on a regular basis, each survey needs a transfer program. The program fetches the data (ordered by form, row, and column), aggregates it, maps it to the taxonomy of the time series database, and performs further survey specific transformations (currency conversion, calculation of totals and residuals, net calculation, etc.).

Section 2 of this paper delves into the problems of the lecacy mapping programs: they were hard to maintain, and they hard-coded the mapping and transformation without any support for handling data lineage information. Section 3 describes the transfer system MapLan, a novel approach that alleviates these problems. Throughout the paper the SNB's securities holdings survey is cited as a real world example. Section 4 briefly presents related work. Finally, section 5 discusses the resulting benefits of the novel approach.

2. PROBLEMS WITH THE LEGACY MAPPING PROGRAMS

Since the transfer from primary to secondary database is carried out on a regular basis and since it moves big chunks of data it needs to be automated by programs. Yet, the mapping (including aggregations and transformations) holds information that is necessary to interpret, to back-track, and to reproduce the time series that economist work with and that are published. In other words, there is a need for data provenance and lineage information [7]. This information was hard-coded in the legacy transfer programs. The transfer programs were mingled with data fetch, data preparation, and data output code. This problem was aggravated by the fact that the two databases use very different data ordering metaphors (object-oriented and form based DB vs. hierarchical time series DB). The transfer programs were procedural. The algorithmic steps (e.g. nested loops) obfuscated the sometimes simple mapping logic that stems from the statistical systematic of the survey. Moreover, the procedural structure of a transfer program reflected the programming style of the individual programmer. A typical business case is that a reader of a statistical publication (e.g. a journalist) asks for further information about a figure published by the SNB. The figure can easily be found in the secondary database, but then the back tracing became very cumbersome because of the missing data lineage information that would lead to the appropriate entries in the primary database. With the legacy system, the case often ended up with the programmer studying his/her code and retracing the mappings and transformations behind it. Obviously, this was a tedious, time-consuming, and error-prone process. Another business case concerned upcoming changes in surveys. The data owners wanted to inform the data consumers about the time series affected by a given change. But when you consider that 1.5 million time series are fed by transfer programs, it was impractical to provide that information. To solve these cases we needed to solve the data lineage problem.

There were also operational requirements. Changes in the transfer programs occur frequently either because the survey is adapted to new requirements or because the time series database is reordered (e.g. based on new requirements by the economists or publications). These changes were a significant maintenance burden to the IT staff. So the new system was supposed to alleviate standard mapping changes. Further requirements to our new data mapping system were:

- Replacement of the outdated and costly programming language (license fees).
- High expressive power, so that all current and future mapping needs can be addressed.
- High performance, so that the large quantity of data can be processed in a timely fashion.

2.1 Example: Securities holdings survey

The "securities holdings in bank custody accounts" survey [2] consists of seven forms (WB51 to WB63) which are identical in structure. Each form surveys a different investment currency. The rows of the forms represent a breakdown by origin of the issuer (resident or non-resident) and by category of securities (in particular money market paper, medium-term bank-issued notes, bonds, shares, structured products etc.). The columns of the forms show a breakdown by custody of account holders and by economic sector (e.g. financial, public). 326 bank offices have to turn in the securities holdings forms (roughly 4000 form positions) on an annual basis. A subset of about 70 large banks has to turn in the forms monthly. The input data has to be mapped to about 63'000 time series representing the securities holdings survey in the secondary database. We use this survey as example because it is fairly typical in size and features relatively simple mappings and transformations.

The hierarchical structure of the keys of the survey's time series is illustrated by the following key example: SNB3A.NA2.SNB.A.B.T.A.D.CHF.M. This is an address of a time series. The key parts, separated by dots, reflect the hierarchy in the time series database. In the GUI of the database, you would follow the key from left to right to navigate to the series data, just as you would navigate through directories of a file system. The following table shows the names and the business meaning of the hierarchical levels and the key codes and its business meaning of the given example key.

Hierachy level name	Level meaning	Key code	Code meaning
DB	Statistics family	SNB3A	Banking statistics
Area	Survey	NA2	Securities holdings
Segment	Aggregate / subject	SNB	Swiss national bank
Item	Domicile of the custody	A	Residents
S1 (Suffix 1)	Sector	B	Financial institutions
S2	Sub-sector	T	Total
S3	Origin of issuer	A	Resident
S4	Category of securities	D	Shares
S5	Currency	CHF	Swiss francs
S6	Frequency	M	Monthly

The first four key parts are mandatory. The further key parts are called suffices. They vary in numbers. The hierarchy of the key reflects the statistical breakdown of the data.

We picked this key as example, because it has a particularly simple mapping: the time series is fed by only one survey form position, namely by form WB51 (securities in Swiss francs), row 6 (Shares issued by residents), column 3 (resident financial institutions). This example features no aggregation because the data of the SNB itself[1] is not made anonymous.

Note, that the aggregated figures of the securities holdings survey are published in the Statistical monthly bulletin [3], tables D5.

3. SOLUTION: A DSL EMBEDDED IN A JAVA MAPPING FRAMEWORK

Our solution rests on two main pillars: (1) a mapping framework and (2) a domain specific language (DSL) [5].

1) The MapLan Java framework builds and processes generic mapping lists of explicit formulae at runtime. Each formula describes in terms of generic database addresses how the data stored in the target database has to be calculated by data in the source database. Since a list holds explicitly instantiated runtime objects, it can be stored as data lineage metadata in a mapping database in order to make the mapping transparent and reusable to end-users and other applications. 2) We embedded a declarative domain specific language (DSL) [5] into the framework. The DSL acts as a formula factory. In case of strongly systematic mappings, few and simple statements of the language suffice to set up a mapping for a large number of database addresses. Pure Java formula factories and interfaces ensure that even very complicated and fragmented mappings can be implemented.

3.1 Mapping lists with explicit formulae: the basic structure

In order to set up the mapping between different kinds of databases and data metaphors, we came up with the following generic formalism: (1) Mapping describes how an address in a target namespace can be filled by a calculation of data addresses in a source namespace. (2) Namespaces (addresses in databases) consist of an arbitrary number of dimension names (strings). An address in a namespace consists of an assignment of an identifier (a string) to each dimension name. Thus, namespaces can be seen as n-dimensional hypercubes, where dimensions have names. Addresses are points in the hypercube. (3) A mapping is a list of entries of type $y=F(x_1, .., x_n)$. Each formula F expresses how data to be found under addresses (points) $x_1, .., x_n$ of the source namespace X has to be computed in order to produce the data under a given address (point) y in the target namespace Y. Silently we assume there is a function class $W_N($ a) that delivers the data value under address a in the namespace N. Remember that our namespaces represent databases, so the function W simply represents the retrieval of a data element given its address in the

[1] The SNB has to deliver the survey as well. It turns it in to itself.

database. So more precisely an entry in the mapping list represents the assignment $W_Y(y) := F(W_X(x_1), .., W_X(x_n))$. Yet, it is more practical to represent F() as a pure address calculus: $y = F(x_1, .., x_n)$ in order to store it as data lineage information. $W_N()$ is only necessary to evaluate the formula at the end and store the calculated result under the address y. Our implementation supports only functions F() consisting of the basic operations (addition, subtraction, multiplication, division) and nested combinations thereof. This turned out to be sufficient to model all the necessary survey mappings, but it could easily be extended.

Section 2.1 showed the mapping of one time series of the securities holdings survey. Let us now represent this example as MapLan mapping entry. The source namespace is the address namespace of the primary database; the target namespace is the one of the time series database. So a given formula operand can be seen as an address in the namespace of the primary database. The namespace holds 4 named dimensions: Subj (the subject that delivers the data), Form (the form identifier), R (the form row), and C (the form column). So the form entry we mentioned in the example can be written as {Subj:SNB, Form:WB51, R:6, C:3}. The time series namespace in the example holds ten dimensions (see table 1). The names of the dimensions are given by convention and reflect the hierarchical nature of the database: (DB, Area, Seg(ment), Item, S1, S2, S3, S4, S5, S6). When we put this together, we can formulate our first, exemplary mapping list entry. It consists of one entry in the target namespace (left side), and its (trivial) formula consisting of one address in the source namespace without any calculation: {DB: SNB3A, Area:NA2, Seg:SNB, Item:A, S1:B, S2:T, S3:A, S4:D, S5:CHF, S6:M} := {Subj: SNB, Form: WB51, R:6, C:3}.

Here is an example of a mapping entry featuring a formula: {DB: SNB3A, Area:NA2, Seg:*AV3*, Item:A, S1:B, S2:T, S3:A, S4:D, S5:CHF, S6:M} := {Subj: *UBS*, Form: WB51, R:6, C:3} + { Subj: *CS*, Form: WB51, R:6, C:3}

This is an example of an aggregate, namely the big banks. There are only two banks in this aggregate, so the data behind the same form position of two banks (see Subj dimension in namespace) is added together. The key to be fed differs from the previous example in the assignment of the Segment dimension (now AV3 – "Big banks") Note that the most complex formula in a single mapping entry currently consists of over 8000 operands.

We said that the base of our mapping system works with a list of entries of the type $y = F(x_1, .. , x_n)$, where F consists of the basic arithmetic operations. Formula, operators, operands, and result are runtime objects. Then there is an evaluation mechanism (resolver) that visits the formula, resolves the addresses of x_i, performs the calculations of that data, and stores the result under address y. The complete data transfer for a survey is performed by repeating this process for the whole mapping list.

3.2 Building the Mapping List with the MapLan System

Each entry in the mapping list describes how a target database key has to be fed with source data. Note that it also represents the complete data lineage. But to build the huge mapping lists necessary to transfer the data of a survey in an efficient way, we needed more tools. We came up with the MapLan DSL, a declarative mapping language that generates the list at runtime and uses the systematic inherent in the statistical survey. We use the fact that mapping in statistical databases is not arbitrary. It is highly systematic because there is an underlying economic domain and both representations of the survey (forms and time series store) embody the structure of that domain to some extent.

MapLan allows the programmer to split the mapping into partial mappings that only reflect the mapping of a subset of dimensions. Subsets should be chosen, so that they are orthogonal to each other. Consider the securities holdings example. The survey features a breakdown by currency. In the primary database, this is reflected by the dimension "Form". In the time series database this is reflected by the dimension "S5" (suffix 5 – see table 1). If you want to know how a given time series (of securities holdings) has to be filled, and you now that suffix 5 of that time series is "CHF" then you know that you need the data from the survey form corresponding to the currency "CHF" (which is WB51). So suffix 5 determines the survey form no matter how suffix 1 or Area etc. are assigned. This dependency is therefore orthogonal to the other dimensions. Partial mapping is the MapLan construct to describe such dependencies between subsets of target and source dimensions. Other dimensions of the mapping example are orthogonal as well. Consider the breakdown by category of securities. In the forms, this is mapped to row numbers. In the time series database, it is reflected in suffix 4 (S4). Yet, in the forms the breakdown by origin of the issuer is also mapped to the rows. First come all the categories of securities issued by residents then all the categories issued by non-residents. The time series namespace reflects the origin of the issuer in dimension S3 (see table 1). Given S3:A ("Resident") and S4:D ("Shares"), you can conclude (independently of the assignments of other dimensions) that you need to get the data from row 6 in the forms. Partial mappings help the programmer to describe such dependencies between subsets of the dimensions. MapLan represents partial mappings as lists of partial mapping entries. Such a list entry looks the same as a complete mapping entry: $y = F(x_1, .., x_n)$, but y can be an incomplete address (for example, as just seen, only the dimensions S3 and S4 are assigned) and the x_i can also be incomplete (for example only the row dimension is assigned).

Composition of Mappings by Means of Partial Mappings. The programmer builds a MapLan mapping by defining a set of orthogonal partial mappings that covers all dimensions of the source and target namespace. The MapLan resolution engine will then generate the mapping list by combining all entries of the partial mappings with each other. So each entry of every partial mapping list is combined with all the entries of the other lists. Every combination results in a mapping entry that is complete (all dimensions are assigned). It is easy to see that, due to the many combinations, relatively small partial mapping lists cover a large mapping range. Therefore, mappings that can be split into several partial mappings can be defined very efficiently. If a mapping is split into n partial mappings, with a list size of pl_n, then its definition consists of $\sum_{i=1}^{n} pl_i$ partial formulae, but it will cover a mapping list with $\prod_{i=1}^{n} pl_i$ complete formulae. On the other hand, splitting is not always possible. Key to the success of designing efficient partial mappings is whether there is orthogonality in the mapping that stems from the business domain (the statistical survey – in our example the different orthogonal break downs of the data). Yet, the expressive power of MapLan is not limited by this challenge. In extreme cases a mapping can be defined with one single partial mapping. This is always possible. It is equal to declaring the complete mapping list explicitly. We never had to resort to this approach when implementing the SNB surveys. Nevertheless, the statistical domain data was often fragmented, consisting not only of one hypercube but of several sub-cubes.

Therefore, we introduced further composition mechanisms to MapLan: mappings can be linked. Linked mappings produce intermediary results. A survey can also be transferred using several independent mappings bundled as one. To a given intermediary namespace several mappings can be linked.

3.3 MapLan DSL definition of the securities holdings survey

Here we show a code snippet of the domain specific language (DSL) of MapLan. It shall demonstrate the expressive power of MapLan and give a flavor of the syntax. We will model the partial mappings of the securities holdings survey described earlier. Note that this example covers the mapping of over 63'000 time series, including the two mapping entries we used as examples in section 3.1.

```
PM_Currency_Anchor:
  Form -> {DB,    Area, S5,  S6}:
  WB51 -> {SNB3A, NA2,  CHF, M},# Currency
                  # in S5 maps to form
  WB52 -> {SNB3A, NA2,  USD, M},
                  # Rest of dims constant
  ..
  WB63 -> {SNB3A, NA2,  Z,   M};


PM_DomicileCustody_Sector_SubSector:
  C -> {Item, S1, S2}:
  1 -> {A, A, Z}, # Resident, non-
                  # financial inst., tot.
  3 -> {A, B, T}, # Resident,
                  # financial inst., tot.
  5 -> {A, B, A1},# Resident,
                  # financial inst.,
       # Collective investm. institutions
  ..
  18-> {B, T, Z},
  19-> {B, T, F}; # Non-resident, tot.,
                  # of which lent


PM_OriginIssuer_Category:
  R  -> {S3, S4}:
  1  -> {A, A},   # Resident, money market
                  # instruments
  2  -> {A, B},
  3  -> {A, C},   # Resident, bonds
  4  -> {A, C1},
  5  -> {A, C1A},
  6  -> {A, D},   # Resident, shares
  ..
  38 -> {B, F2},
  39 -> {B, F3},
  40 -> {B, F4},
  41 -> {B, FZ};  # Non-resident, residual
                  # structured products
```

```
PM_Intitute:
  Subj     -> Segment:
  SNB      -> SNB,
  ..
  [CS+UBS] -> AV3;# Aggregate of two
                  # big banks
```

The example features a mapping consisting of four partial mappings. Each partial mapping begins with a name (for listing purpose) followed by the declaration of the subsets of dimensions that it covers. Note, that when combining all subsets of the four partial mappings (as the resolver does), we get all four dimensions of a primary database address and all 10 dimensions of a time series address. After the declaration part, partial mapping entries are defined. When indicated by ".." we shortened the lists somewhat for readability reasons. The complete example would use as little as 75 lines. An entry consists of a formula of partial points in the source namespace (form position) on the left side (of the -> sign) and a partial point in the target namespace (time series key) on the right side. Note that the assignments refer to the dimension(s) of the declaration. So in the third partial mapping named PM_OriginIssuer_Category we find an entry that says row 6 of the form corresponds to {S3:A, S4:D}. This is exactly the example showed in section 3.1. Note that when an entry assigns only one dimension then the MapLan syntax allows one to omit the curly brackets. Note also, that in most of the cases the left side is not a formula, but a single (partial) point. This is of course allowed. The last declaration in the last partial mapping shows an example of a formula (left side). It is the aggregation of the data of two banks into one banking group (AV3) we also saw earlier. For clarity, we omitted the aggregation into larger banking groups. In principle, these formulae are sums with as many summands as there are banks in the group. In practice, since the aggregation into banking groups is a standard operation for MapLan applications, there is a factory to generate these formulas on the fly and in a transparent way.

When executing this example, the MapLan resolution mechanism generates all combinations of right side of the entries. This gets us the huge list of time series keys to update. It then resolves the left side (and their formulas) for each entry into a formula with complete primary database keys and evaluates it. The resulting figure is written to the time series hence it is transferred.

3.4 Advanced MapLan Features

This section addresses three further MapLan features without going into detail: handling of reporting dates, changes over time, and reuse of mapping definitions.

Handling reporting dates and changes. Survey data has a reporting date that is used to interpret the data. Obviously, this is also the date to be used for the entry of the target time series we want to update. So mapping the date is straight forward. Therefore, this is hard-wired into our data transfer solution. Yet, MapLan is open to handle time as just another namespace dimension, offering full flexibility.

We said earlier that due to changes in the statistical domain the mapping may change over time. Usually this happens at a given reporting date. For surveys performed for earlier reporting dates we still need to be able to perform data transfers with the old mapping definitions (e.g. when reporting banks send in corrections). We solved this problem with the directory structure for MapLan DSL mapping files. The directory names indicate the first reporting date for which the file is valid. Newer valid entries override older ones. When loading the MapLan DSL mapping files, the system dynamically resolves the appropriate mapping files to use.

Reuse of mapping definitions. Often, several mappings share some partial mappings. Furthermore, most of the time when a mapping changes only some partial mappings change while others stay the same. E.g. when a form gets a new row, it does not necessarily get a new column as well. To simplify the maintenance of mapping declarations, we introduced load commands in the DSL that allow the author of mapping definitions to load partial mappings from separate files instead of listing them in one file (as seen in the example of section 3.3). Partial mappings that were factored out in separate files can then be shared by several mapping definitions. Furthermore, these partial mappings can also profit from the date-relative load mechanism described above. This minimizes declaration duplication due to changes in mappings. Note, that not only partial mapping declarations can be loaded with the DSL but also partial mapping factories (by their java class name). That way, you can access the full expressive power of Java via the DSL.

4. RELATED WORK

We did not find any research result or product that could solve the described problem off the shelf. To our knowledge the MapLan approach (using a declarative DSL to exploit inherent mapping logic and run-time formulae of generic database addresses that serve as complete data lineage) is novel. Yet, our work is related to research in the following areas: data mapping and integration, data provenance and lineage, and domain specific languages.

Data mapping is usually seen as a step of data integration which has been covered by a large body of research. For a survey see [9]. Data mapping research often focuses on mapping data schemas to each other with the ultimate goal to query across multiple heterogeneous data sources. The data is managed by different autonomous authorities. Data mapping occurs between XML schemas or between relational database schemas. Our work differs from other research because we deal with the special issues of statistical databases. The problem is not semantic uncertainty in how to map and transform our data but the very large and non-relational schemas that have to be mapped. Such large schemas (the securities holdings survey alone consists of 63'000 schema fields) are typical for statistical databases. Traditional schema mapping approaches will most certainly lead to maintenance problems. Often, the establishment of views is proposed to integrate data sources. A statically established view, including the transformation operations, would probably not perform reasonably, since it would consume far more memory than the data itself. Instead, we developed a domain specific language to describe mappings. The DSL exploits the inherent systematic of statistical schemas (multi-dimensional hypercubes) and instantiates mapping formulas on the fly.

In the context of data warehouses data provenance can be seen as metadata about the processing history and data lineage. In [7] the authors describe the data lineage problem, namely how to trace warehouse data items back to the original source items from which they were derived. In [8] the authors describe a formal and efficient approach to trace lineage for general database transformations. The formalism introduces transformation classes (dispatchers, aggregators and black-boxes) that can be composed to acyclic transformation graphs. Based on properties of the transformation class, the authors show how and what data lineage can be traced. In the introduction of this paper we showed our business case for data lineage. We wanted to be able to reproduce any given figure of the secondary database, regardless of the mapping, transformation, or aggregation it underwent during its transfer from the primary database. So we need complete lineage information. The approach in [8] is very generic so it deals with the unknown properties of some transformations (black-box). In such an environment, the lineage information becomes sparser. Instead of using a generic lineage tracking algorithm, MapLan uses the fact that at transfer time it builds perfect lineage information (the reified mapping formula). MapLan allows the programmer to build mappings by composition which also leads to acyclic graphs of transformations. Yet, MapLan resolves these graphs into one transformation that is perfectly traceable. The fact that only a limited set of arithmetic operations is used makes this possible.

With MapLan we wanted to exploit the opportunities of DSLs as described by Deursen et al. [5]: The DSL embodies domain knowledge and thus enables the reuse of this knowledge. Mappings can be expressed at the level of abstraction of the problem domain (statistical data taxonomies). The partial mapping lists focus on the relation between survey forms and time series keys. This is very similar to how an end-user would write down the mapping relation. So the DSL is self documenting. DSLs have the potential to enhance productivity, reliability, maintainability, and portability. These were exactly the requirements we wanted to meet. MapLan allows the user to formulate mapping statements that leverage the inherent systematic typical to statistical data. We also needed to keep an eye on the potential disadvantages of DSLs mentioned in [5]. The domain of the DSL needs to be mature. Since we supported the mapping for years we could assure that. The cost of identifying the scope and developing a DSL needs to be justified. MapLan was cost-effective because it replaced an expensive legacy programming language. When the mapping is done in a DSL it may run into performance problems compared to hard-coded, individually forged mapping programs. This was indeed a challenge but we could overcome it by using well established techniques such as caching and efficient data structures such as Java maps.

5. RESULTS

Within four years, all 30 surveys of the SNB were migrated to the MapLan transfer system. By replacing the legacy system, we save a 6 digit CHF amount in licensing fees every year. However, the true benefits of MapLan lie elsewhere, namely: availability of the mapping information (data lineage) and efficiency of the declarative language MapLan in describing the mappings thus reducing the maintenance burden.

5.1 Efficiency of the declarative language

MapLan maps the large schemas of statistical databases very efficiently. Mapping over 1.5 million schema items manually would be infeasible. MapLan offers a way to exploit the intrinsic business logic of the statistical surveys. Logically independent mapping dimensions can be factored out in partial mappings with few statements. In this paper we presented a running example that maps 63'000 schema elements using only 75 lines of DSL code.

Declaring partial mappings allows the programmer to work with simple and intuitive statements ("this form row goes to that key part of the time series"). Fewer statements that are more intuitive simplify the maintenance of the mapping when it changes through time.

The expressive power of the system is guaranteed by composition methods: mappings can be constructed by linking several mappings or by bundling them. If all else fails mappings can be constructed from one large partial mapping. Partial mappings can be generated with factories in a controlled and transparent way, allowing the programmer to use the full expressive power of Java.

5.2 Making use of the data lineage

Internally, MapLan produces mapping lists. A list entry describes how an address in the target namespace has to be fed by addresses in the source namespace. The description is a formula consisting of basic arithmetic operations.

The mapping list is built at run-time with the main purpose to update the target database by evaluating the formulas. Note however, that the mapping list represents complete data lineage for the data in the secondary database. In order to benefit from the lineage information, the MapLan transfer system stores the mapping list in compact form into a meta-database. Now the data lineage can be used to provide new value-added services. Here are some examples.

Enabling the user to trace back data. Providing access to complete data lineage is a key achievement of MapLan, as compared to the legacy system or as far as we know to any mapping system of statistical databases. No longer is the mapping logic hidden in procedural programs. We leverage the mapping metadata by building different web-based queries for that metadatabase. The queries allow our users (economists) to find out what survey (positions) go into what time series, how a given time series is calculated, and even do a drill-down, starting at the time series and getting all involved data from the primary database, plus the mapping formulae. Note that these web-queries solve the business cases mentioned in the introduction.

Automatic anonymity check. The mapping formulae can be visited by other modules before the data transfer is executed. We use this fact to implement the confidentiality options that some surveys need. The confidentiality module analyses the formula (and the underlying data) using the visitor pattern [4]. It marks those figures as confidential where the input of a single bank dominates the resulting figure. We can therefore assure that the aggregates make the data sufficiently anonymous. The module implements a strategy pattern [4] so it is easily adoptable.

Automatic drill down of outliers in aggregates. The aggregated data of some surveys is delivered to international organizations. For example, the Bank of International Settlements (BIS) receives the consolidated banking statistics. In order to ensure the data quality, the BIS performs outlier tests on the data. We accelerated the delivery process by incorporating the test into the data transfer. We find the same outliers in the aggregates at a much earlier stage. Since with MapLan we know the exact data lineage of each aggregate, the system is able to rank the contribution of each bank to the outlier and identify the contributing form positions. Therefore, very early in the process, outliers in the aggregate can be precisely explained or corrected.

Generic mapping approach. MapLan has proven to be applicable in further domains. This is due to the fact that we have ensured the expressive power of the language (composition, fall-back solution to Java etc.). Another enabler is the fact that MapLan works with generic namespaces of arbitrary dimensions. While this approach proved helpful to map between two specific databases, it can also be used to describe a broad range of other mapping scenarios (including relational databases). For example, we used MapLan to generate SDMX [6] data messages from the primary database. As another example, when the primary database was replaced by a completely new system we could easily plug-in the new database even though the data representation had changed. As a fancier example, we used MapLan for consistency checks for data to be published. Since the data is available in a hyper cube (publication document, table id, row, column) we could easily and efficiently express checks like adding up columns and seeing if that ends up with the figure in the "total" column.

6. FUTURE WORK

MapLan is now a mature "working horse" and we do not plan to change much. One idea is to use survey metadata like row/column descriptions to automatically comment the mapping artifacts or even generate skeletons of mapping declarations. More visionary would be to introduce a visual and interactive mapping development environment. However, while the framework would be ready for that, it is questionable if this will raise productivity the way the introduction of MapLan did.

7. REFERENCES

[1] Swiss National Bank (SNB). Surveys Overview. http://www.snb.ch/en/iabout/stat/collect/id/statpub_coll_overview

[2] Swiss National Bank (SNB). Survey documents: Securities holdings. http://www.snb.ch/en/emi/WEBE

[3] Swiss National Bank (SNB). Monthly Statistical Bulletin. ISSN 1661 – 0296. http://www.snb.ch/en/iabout/stat/statpub/statmon/stats/statmon

[4] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.1994. *Design Patterns: Elements of Reusable Object Oriented Software*. Addison-Wesley.

[5] Van Deursen, P. Klint, and J. Visser. June 2000. *Domain-specific languages: An annotated bibliography*. ACM SIGPLAN Notices, 35(6):26–36.

[6] Eurostat et al. *Statistical Data and Metadata Exchange (SDMX)*. http://sdmx.org/

[7] Y. Cui and J. Widom. February 2000. *Practical lineage tracing in data warehouses*. In *Proc. of the Sixteenth International Conference on Data Engineering*.

[8] Y. Cui and J. Widom. 2001 *Lineage Tracing for General Data Warehouse Transformations*. In *Proc. of the 27th VLDB Conference*.

[9] A. Halevy, A. Rajaraman, and J. Ordille. 2006. *Data Integration: The Teenage Years*. In *Proc. of the VLDB Conference.*