Key Architecture Choices in microdata.no

Version 1.0 – June 2022

Contributors:The microdata.no teams in

Sikt - Norwegian Agency for Shared Services in Education and Research (Sikt)
Statistics Norway (SSB)

Abstract

Microdata.no is a self-service data access and research platform for instant[1], informed and interactive access to fully detailed and linkable register data and other types of microdata.[2]

End-users freely compose result data sets and define populations of interest from available source variables. User-composed result data sets can be flexibly merged, transformed, derived, aggregated, disaggregated and analyzed.

A web based integrated development environment (IDE) lets users interactively develop statistical programs in the form of scripts that are executed immediately on the platform, returning analytical results and other relevant output in a confidentiality preserving manner.

Figure 1 - The microdata.no IDE – showing script mode

This paper shows how minimalistic and carefully designed data and metadata models support desired end-user functionality as well as a minimalistic technical architecture.

Here we will show how seemingly orthogonal design aspects related to researcher needs, technical architecture and choices in data-/metadata organization all influenced big and small design decisions and support a coherent and relatively simple overall system architecture.

We discuss how:

- Researcher input and confidentiality requirements influenced user interface design
- Interface design influenced data and metadata models

[1] Without any application process. User accreditations are issued by the researcher's own institution.
[2] For more information, see https://www.microdata.no/en/

- Multi-variable source data sets are decomposed into standalone, single-variable source data sets.
- Single-variable data sets are associated to unit identifiers.
- Single-variable data sets are enriched with temporal information.
- Single-variable data sets are annotated with simple metadata structures inspired by core models from GSIM and DDI metadata models.

The foundations for microdata.no's current data and metadata models are described in detail in the report "RAIRD Information Model".[3]

A close relationship between data and metadata models and desired functionality

Data values are never self-explanatory and cannot be well understood or interpreted without precise knowledge about definitions, coding/encoding, measurement units, measurement periods, etc – in other words, rich and detailed metadata.

A core ambition for the microdata.no IDE is to guide and assist users in making informed decisions and to help them work efficiently and comfortably with available data.

To accomplish this, rich and detailed metadata document all variables and simultaneously drive system behavior. *Informational* metadata such as variable definitions as well as relevant *technical* metadata such as validity periods and labeled code lists, exist as an integral part of the IDE, driving and encapsulating user interactions when working with data.

Therefore, data and metadata models designed for microdata.no are at the heart of the platform, enabling

- IDE functionality such as interactivity, lookup of definition, encoding and measurement units, data discovery, feedback, error handling, suggestions, auto-completion
- Automated Statistical Disclosure Control (SDC) decisions[4]
- Data versioning

Here we describe microdata.no's data and metadata models and how they are designed to support the following user-empowering ambitions:

Users can freely compose result data sets and populations from all available data.

1. Automated SDC decisions keep data outputs confidential.
2. The IDE explicitly supports temporal aspects of data (event data, panel data, etc)
3. The IDE continuously visualizes structure and core properties of data sets as they are created, transformed, and analyzed.
4. The IDE's scripting language abstraction level is designed to allow users to express their intended actions rather than typing low-level steps to accomplish the intention.

[3] https://statswiki.unece.org/display/gsim/RAIRD+Information+Model+RIM+v1_0
[4] For more details on microdata.no's Statistical Disclosure Control mechanisms, see http://www.unece.org/fileadmin/DAM/stats/documents/ece/ces/ge.46/2019/mtg1/SDC2019_S7_Norway_Heldal_AD.pdf

5. The IDE has good defaults capturing common user actions through concise statements and interactions.
6. The IDE supports more advanced actions through additional options and commands.
7. The IDE blocks logically invalid operations and gives helpful feedback in the process.
8. The IDE blocks operations that exceed SDC thresholds and gives helpful feedback in the process.
9. Reproducibility: analytical output is a function of the script and nothing but the script.[5]

Good data- and metadata models are necessary, but obviously not sufficient to reach these ambitions. The following additional foundations are designed in close relation with the data and metadata models, but are out of scope for this paper:

- Grammar development for the IDE scripting language
- Language modules for auto-completion, semantic error messages
- IDE architecture and implementation
- Complete system architecture for microdata.no
- SDC rules, mechanisms and decision points
- Overall security architecture
- Data versioning system

Analyses of requirements for a system architecture that could fulfill the previously stated ambitions hinted at simplistic data and metadata models with certain desirable characteristics.

The next chapters explain the remaining rationale behind several key architecture choices for microdata.no.

Key choice 1:
A command/script-based web IDE

Early inputs from the research community made it evident that typical researcher workflows when working with register data is dominated by data transformations, creation of derived variables and population definitions, involving complex boolean logic and selection criteria. Frequently, data transformations would constitute several hundred lines of program code in statistical analysis tools such as SAS, Stata, SPSS, R or Python.

It quickly became clear that no *graphical* user-interface would let users express complex data transformation logic in an efficient manner. An approach based on programmatic commands and scripts (collection of commands) was chosen for the main user interface in microdata.no.

Direct usage of any of the previously mentioned languages/packages would however not be possible. The main ambition of instant analytical access to all available register data requires that end-users (e.g., researchers) *cannot see nor infer personal identifiable information* through output from the system; all output would have to preserve confidentiality of the data subjects.

[5] Read more about reproducibility and data versioning in the paper FAIR Data Versioning in Microdata.no https://www.microdata.no/en/the-architecture-behind-data-versioning/

Existing languages/platforms that support direct programmatic access to data are not designed to be safely constrained to support microdata.no's confidentiality-ambition and automated SDC mechanisms.

To sum up:

- Researchers needed a command/script-based user interface
- Existing command/script-based solutions could not be used directly

The logical choice was therefore to create a language and command-/script based user interface that would let users work efficiently and in familiar ways, while maintaining control over the executional context for these commands and scripts and thereby operations on data.

As stated earlier, it is beyond the scope of this paper to describe language design and overall system architecture in detail. However, a brief explanation of script execution is in order:

Scripts written in the microdata.no-language get interpreted in the web IDE, and each command gets compiled into structured API calls that are sent to the server stack. There, API calls invoke Python programs that operate directly on data. These Python programs combine use of the excellent data processing libraries available in the Python ecosystem with SDC-algorithms developed for microdata.no, ensuring that responses to the web IDE are confidential.

Language design opportunities

The choice to design and implement a concise language that enables researchers to work comfortably and efficiently with large collections of microdata was regarded demanding but necessary.

It also presented an opportunity to design a language that could:

- Support user autonomy in composition of bespoke data sets from source variables
- Have core support and constructions for managing temporal aspects in data
- Include support for tasks and workflows typically associated with register data
- Integrate well with metadata (suggestions, command validation, error handing, etc) in the IDE
- Have an abstraction level that lets users focus on data processing and analysis rather than low-level details
- Be extended to support most known analytical methods
- Be extended to support new and emerging analytical methods

These opportunities show the intimate relationship between design of the microdata.no-language and the overall data-/metadata architecture. User interactions rely on language constructions that again rely on support from data and metadata models and highly structured metadata content.

Key choice 2:
Decomposition of data into single-variable data sets

A variable is a set of attributes (e.g., sex, age, income, height) that describe a set of units (e.g., persons, vehicles, municipalities) at a given point in time or over a period of time.

Researchers investigate the relationship between different variables to generate new knowledge, using a range of statistical methods.

For practical and historical reasons, variables have frequently been organized into two-dimensional multi-variable data sets, where each row typically corresponds to a unit (e.g., a person) and each column to a variable:

Unit identifier	Sex	Age	Income	Height
1	"1"	45	400 000	187
2	"2"	34	500 000	165
3	"1"	67	450 000	173
…	…	…	…	…

Table 1 - Data set with 4 variables and a unit identifier

For survey data and other types of data designed for research and collected at certain points in time, a multi-variable data set layout is preferable not only for data publisher, but also for data users. Such variables have been deliberately collected to answer a set of research questions specified prior to data collection, and the unit identifiers typically do not support direct linkage with other/external data sources.

Register data, on the other hand, are *not* originally designed for research. Instead, register data exist as results of registration of unit-related information in administrative registries, including:

- Tax registries
- Vehicle registries
- Education registries
- Housing and address registries
- Health registries
- Criminal justice registries
- Etc

Again, for historical and practical reasons, register data have also traditionally been organized as multi-variable data sets. Commonly, a given multi-variable data set was managed and maintained by a specific organizational unit or curation team – with specific goals, purposes and mandates related to data curation, data quality, data release, etc.

Researchers however have quite different goals, purposes and mandates than register data curators have. During the initial phase of microdata.no, a multi-variable data set organization was therefore identified as a major hurdle for our ambition to enable researchers to freely define and investigate research questions and to utilize the inherent linkage possibilities present in Norwegian register data.

Below we will discuss some of the issues we identified in the multi-variable data set approach, and our approach to mitigate these problems in microdata.no

Problems with multi-variable data sets for register data

Inherent linkage potential across sources – national unit identifiers

In Norway, a national identity number[6] is issued for all residents and citizens. Similar national identifiers exist for vehicles, businesses and several other types of units.

In register data, the unit identifier variables are, or can be related to one or more of such national unit identifier series[7]. This is a fairly unique property of Norwegian register data, enabling unparalleled linkage possibilities between sources.

In short, researchers could link data from *all* available register data sources that uses national unit identifiers. From a research perspective, the possibilities are virtually endless. Moreover, it is clear how these possibilities transcend the scope of any given organizational unit of data curators.

Our ambition was for researchers to be able to fully utilize the linkage potential of Norwegian register data. We saw traditionally organized multi-variable data sets as a complicating and sometimes confusing factor in achieving this goal, making it harder than necessary for data users to compose succinct, manageable and comprehensible result data sets.

Temporal aspects, event data, complex or sparse data sets

Another important property of Norwegian register data is the existence of event history data.

A classic example of an event history variable is marital status; each unit (e.g., person) may only have one state at a given point in time – but the state may vary over time.

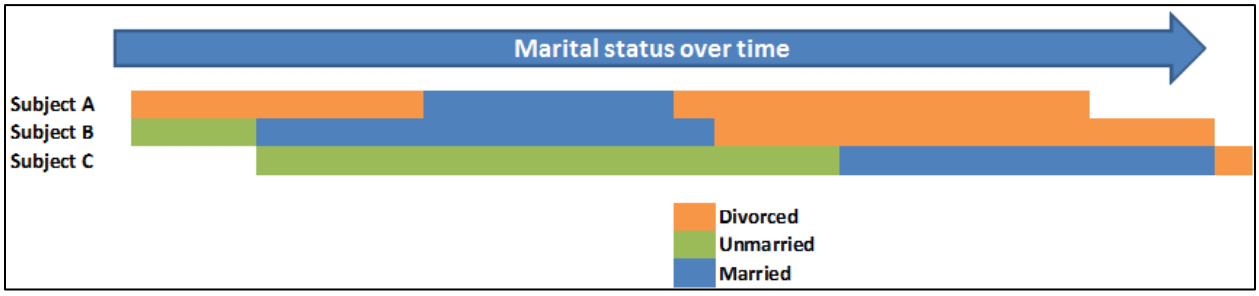

Figure 2 - Illustration of marital status event histories for 3 persons (subjects)

With event history data, it becomes possible to study the state of a population for a given point in time, for a series of time points – as well as state-changes for continuous time periods. From a research perspective, event history data constitute rich analytical possibilities,

[6] https://en.wikipedia.org/wiki/National_identity_number_(Norway)
[7] Various pseudonymization regimes and techniques apply. A further discussion of pseudonymization is kept out of scope in this paper for clarity reasons, but is a central part of microdata.no.

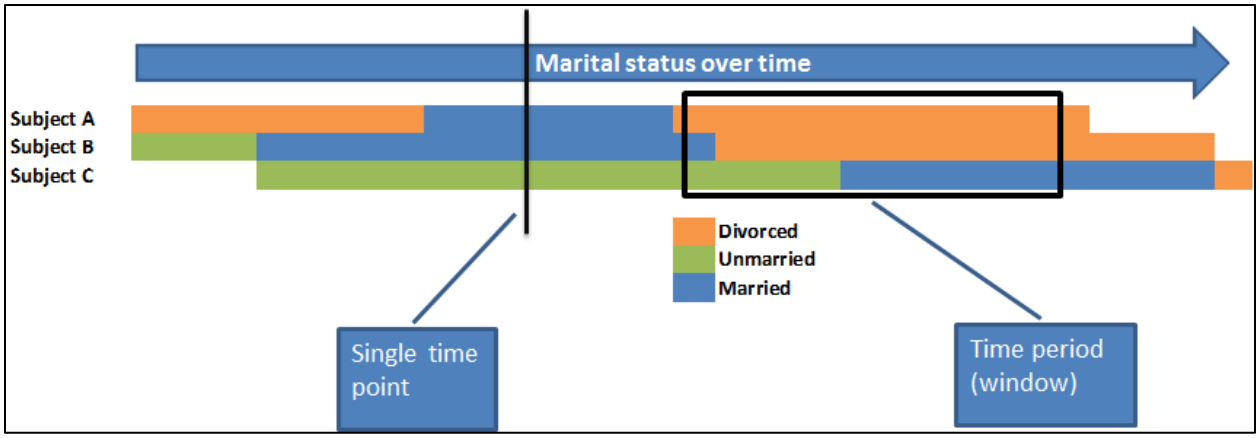

Figure 3 - Possibilities when working with event history data

It was always clear that microdata.no would need to support event history data in ways that would honor the great potential in this type of data. However, the logical structure of event history data revealed additional problems with multi-variable data set organization.

Event history data consist of state-changes for a given variable (e.g., marital status or area of residence) for a set of units (e.g., persons). The state changes for each unit (e.g., a person gets divorced) happen independently of state changes for all other units.

But – a state change in the variable marital status for one specific person is not coordinated with state changes in other variables (e.g., area of residence) for the same person. As we demonstrate below, the phenomenon of independent event history variables is not compatible with multi-variable data set organization.

To illustrate the problem, we have below created two short and simplified examples showing the logical structure of two different event histories for the same set of 3 persons: event histories for marital status and for area of residence, respectively:

Unit identifier	Marital status	Marital status start date	Marital status stop date
1	"1"	2010-01-10	2012-03-25
1	"2"	2012-03-26	
2	"2"	2002-05-25	
3	"1"	1996-01-31	2001-11-12
3	"2"	2001-11-13	2016-04-06
3	"1"	2016-04-07	
…	…	…	…

Table 2 - Marital status event histories for 3 persons

Unit identifier	Area of residence	Area of residence start date	Area of residence stop date
1	"0301"	2009-01-04	2011-06-06
1	"1505"	2011-06-07	2015-06-30
1	"0301"	2015-07-01	
2	"1201"	1994-05-23	1996-01-15
2	"0301"	1996-01-16	

3	“1406”	1986-08-14	1999-09-03
3	“1201”	1999-09-04	
…	…	…	…

Table 3 - Area of residence event histories for 3 persons

Obviously, start and stop dates for marital status do not correspond to start and stop dates for area of residence for the persons in the example, nor for persons in general.

Generally, as all register data curators know, event history data sets for different variables cannot be combined sensibly in a two-dimensional multi-variable data set without duplicating unit identifiers and creating a high ratio of blank/invalid cells and “sparse” data matrices. Consequently, event history data typically reside in single-variable data sets.

For microdata.no it was clear that first-class support for event history data would involve single-variable data set layout for this type of variables. Moreover, because of the previously mentioned points regarding realization of linkage potential, it was decided to choose single-variable data set layout for *all* types of variables, and not just for event history variables. Key choice 5 below goes through the logical layout of single-variable data sets.

We will later demonstrate how this decision enabled simplistic data- and system architectures and resulted in several benefits across many dimensions.

Key choice 3:
A typology for temporality

An early investigation of SSB’s register data holdings, revealed 4 common temporality types.

- Fixed data—data that do not change over time for a given unit (e.g., area of birth, country of origin)
- Event history data—each unit may only have one state at a given point in time – but the state may vary over time (e.g., marital status, area of residence).
- Cross section data – data for each unit is reported at specific time points, without knowledge of the state between time points (e.g., number of children in the family, code for current main job)
- Accumulated data – periodically recorded data that covers the entire period (e.g., yearly income, yearly debt)

These 4 temporality types have turned out to be sufficient for microdata.no-purposes since the initial release for the service in 2018. New temporality types will be evaluated and integrated when needs emerge.

The 4 existing temporalty types have been assigned reserved keywords in a controlled vocabulary that is used in the metadata documents for single-variable data sets:

- FIXED
- EVENT
- STATUS
- ACCUMULATED

Key choice 4:
Differentiated handling of unit types

As mentioned above, a variable is a set of attributes (e.g., sex, age, income, height) that describe a set of units (e.g., persons, vehicles, municipalities) at a given point in time or over a time period. Register data and microdata in general cover an unbounded range of different unit types.

Some unit types however deserve special attention, since they are directly associated with data protection and confidentiality legislation (e.g., persons and businesses), whereas others are not (e.g., municipalities).

Many unit types are unproblematic in isolation, but may at the same time support data aggregation into the level of protected unit types (e.g., vehicle data may be aggregated pr. person and subsequently linked with person level data).

In microdata.no, variables that measure unit types that require special attention get tagged in metadata with keywords from a controlled vocabulary of special unit types.

Linkage variables that support aggregation/disaggregation involving protected unit types are also tagged with relevant keywords from the same vocabulary. An example of the latter is the variable linking a vehicle (unit type: VEHICLE) to its owner (unit type: PERSON). This variable exists for the vehicle unit type, but is additionally tagged with unit type: PERSON in a special metadata field that drives aggregation and linkage functionality.

Leveraging these special unit types furthermore allows SDC mechanisms to continuously monitor the population of protected units across aggregation and disaggregation between unit types, enabling the application of sound confidentiality measures without needlessly interfering with user workflows.

Key choice 5:
Logical data layout for single-variable data sets

The rationale for using single-variable data sets is given earlier in this paper.

The following effects of the choice should be noted:

- A single-variable data set contains data for one unit type only
- A single-variable data set contains data with one temporality type only
- Data validity requirements can vary between temporality types

A single-variable data set has two or more columns of the following nature:

- **Unit identifier**. Mandatory.
- **Measure** (the actual data values). Mandatory.
- **Start date**. Mandatory for temporality types other than FIXED
- **Stop date**. Mandatory for temporality types other than FIXED

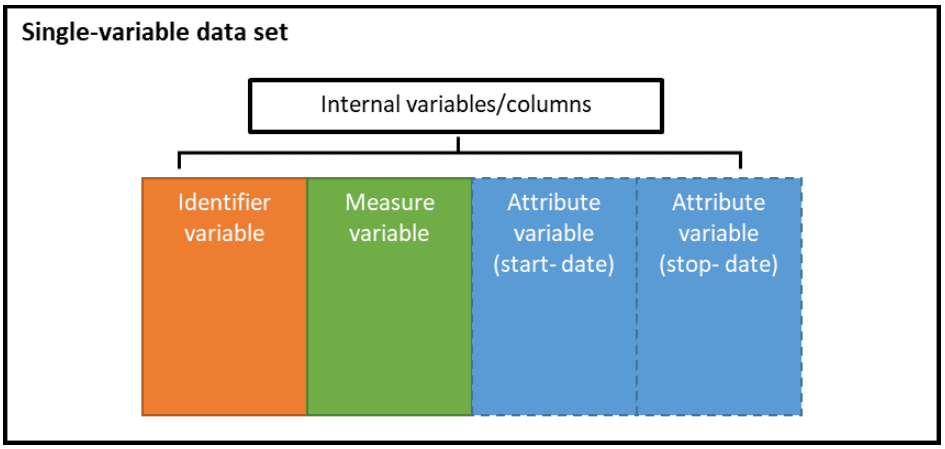

Figure 4 - Illustration of the inner structure of a single-variable data set

Examples below.

Example 1: Temporality type: FIXED

Unit identifier	Measure: Region of birth
1	"0301"
2	"1201"
3	"1406"
4	"1505"
5	"0101"
6	"1606"
7	"0301"
…	…

Constraints for data with temporality type: FIXED:

- One record pr. unit
- No date columns

Example 2: Temporality type: EVENT

Unit identifier	Measure: Marital status	Marital status start date	Marital status stop date
1	"1"	2010-01-10	2012-03-25
1	"2"	2012-03-26	
2	"2"	2002-05-25	
3	"1"	1996-01-31	2001-11-12
3	"2"	2001-11-13	2016-04-06
3	"1"	2016-04-07	
…	…	…	…

Constraints for data with temporality type: EVENT:

- More than one record pr. unit is allowed
- Mandatory columns for start end stop dates
- States for a given unit cannot overlap in time
- Start dates cannot be blank
- Blank stop date associated with a state means state is still valid

Example 3: Temporality type: STATUS

Unit identifier	Measure: Number of children in the family	Number of children in the family start date	Number of children in the family stop date
1	1	2010-01-10	2010-01-10
2	4	2010-01-10	2010-01-10
3	2	2010-01-10	2010-01-10
1	1	2011-09-30	2011-09-30
2	5	2011-09-30	2011-09-30
3	3	2011-09-30	2011-09-30
…	…	…	…

Constraints for data with temporality type: STATUS:

- More than one record pr. unit is allowed
- Mandatory columns for start and stop dates
- Start date and stop date are identical
- Start dates and stop date cannot be blank

Example 4: Temporality type: ACCUMULATED

Unit identifier	Measure: Yearly income	Yearly income start date	Yearly income stop date
1	450 000	2010-01-01	2010-12-31
2	500 000	2010-01-01	2010-12-31
3	550 000	2010-01-01	2010-12-31
1	470 000	2011-01-01	2011-12-31
2	375 000	2011-01-01	2011-12-31
3	660 000	2011-01-01	2011-12-31
…	…	…	…

Constraints for data with temporality type: ACCUMULATED:

- More than one record pr. unit is allowed
- Mandatory columns for start and stop dates

- Start date signifies start of accumulation period
- Stop date signifies stop date of accumulation period
- Start dates and stop date cannot be blank

Key choice 6:
Logical data layouts for user-composed result data sets

While users never modify single-variable single-variable source data sets, they create, modify and manage *result* data sets themselves. All analyses are executed on user-composed result data sets.

The single-variable data sets serve as source data from where users compose bespoke result data sets using commands from the `import*` family of commands.

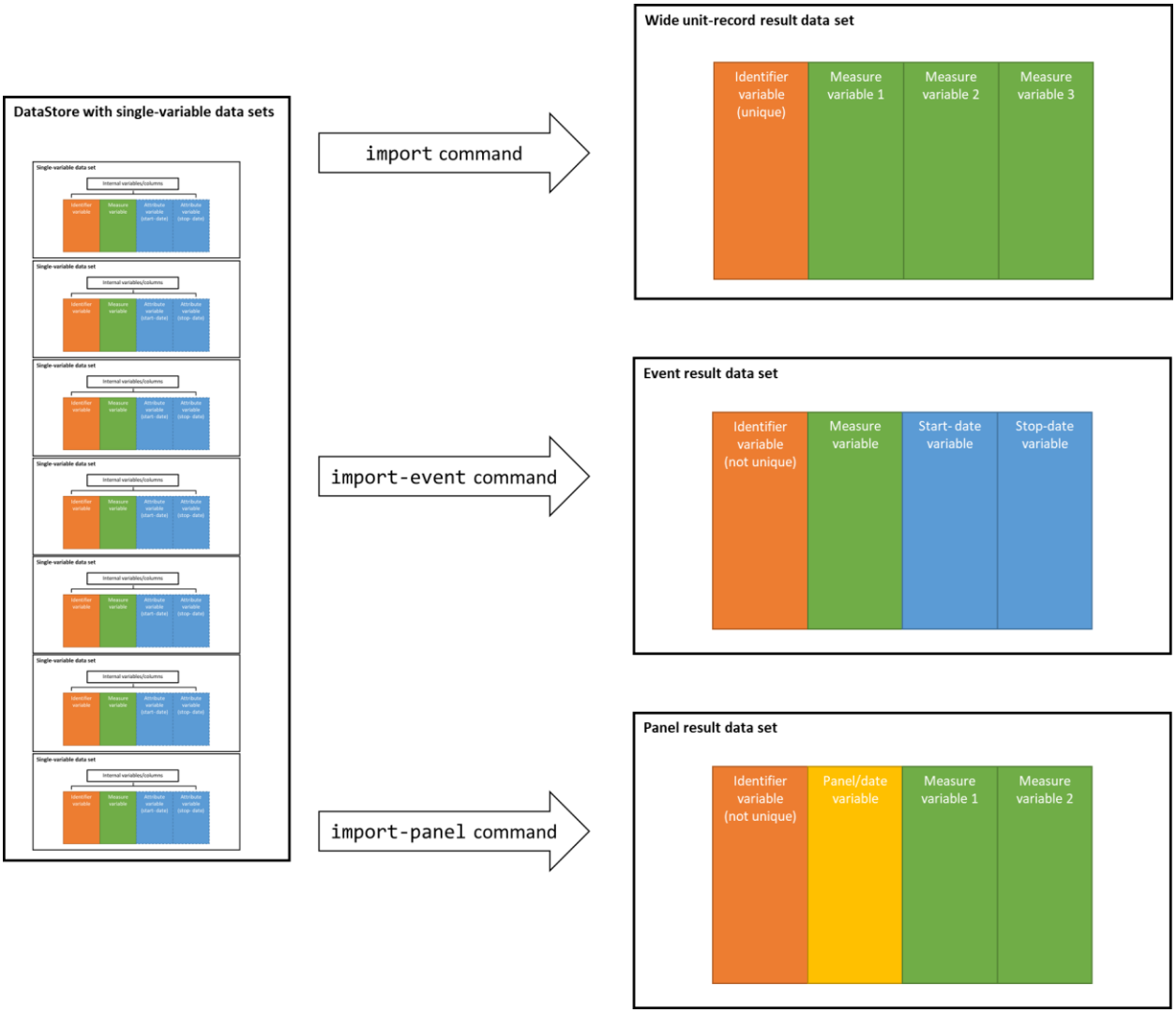

Figure 5 - Three types of user-composed result data sets

The illustration above shows the three different types of result data sets currently supported in microdata.no:

- Wide, unit-record result data set with one row pr. unit (e.g., person) and measure variables linked to the unit identifier. No explicit, system-managed temporal component.

- Event result data set with event history data for one measure variable and a given time-period. Start- and stop-dates exist as explicit variables.

- Panel result data set with repeated measurements for the same units (e.g., persons) for several time points. Measurement date exists as explicit panel/date variable.

The different result data set types support different forms of statistical analyses. Certain analyses require event history layout, others require panel data layout.

The need for analytical flexibility, combined with the fact that SDC-techniques also need to cover event- and panel data in a sound manner, called for explicit modeling of the different types of result data sets.

An alternative approach could have been to model all types of data as generic row/column structures, an approach that would have prevented high-level support for temporal data management, and complicated SDC-systems considerably.

Language design played an important role in the result data set modeling; users write different commands to compose the different types of result data sets:

- The `import` command copies data from the single-variable source data sets into wide, unit-record data sets. This command supports a “snapshot date” argument to produce result variables for a given point in time (e.g., marital status for 2015-01-01) for the entire population present in the source. The import command also works for FIXED source variables, and in these cases no date argument is required/permitted.
  Linkage based on unit identifiers is handled automatically by the system.

- The `import-event` command takes *two* dates as arguments (a from-date and a to-date), and creates a result data set with event histories for a time *period* for a given source variable.

- The `import-panel` command lets users specify a set of source variables as well as a set of dates, and produces a panel data layout, with the set of dates appearing in a special panel-date column in the result data set.
  Linkage based on unit identifiers is handled automatically by the system.

Note that, when logically viable, users can transform data between the three types of result data sets, e.g.,:

- Event result data sets may be aggregated to and merged with unit-record data sets.
- Unit-record data sets with several snapshot date variables can be transformed to panel structure using the `reshape-to-panel` command
- Panel data sets can be transformed into wide, unit record data sets via the `reshape-from-panel` command

Final remarks:
A user may compose as many result data sets as desirable. There is no limitation on the number of measure variables (imported or derived) that can be present in a result data set. Linkage is handled automatically/implicitly by the system whenever possible.

Users can merge, aggregate or use specially designed commands (e.g. reshape-to-panel) to move data between different result data sets, and combine data for different unit types (e.g., person, vehicles, jobs, etc) through aggregation, disaggregation and merging in order to create result data sets suitable for analysis.

SDC-systems monitor and manage confidentiality across these flexible transformations, always maintaining clear and explicit provenance trails from result data/derived data back to source data.

Key choice 7:
Metadata model

The logical data layout consists of 2 or more “columns”, often referred to as “internal variables” in the single-variable data set. Each such column/internal variable has dedicated metadata associated with it.

The metadata model has three main levels:

1. The DataStore (the versioned[8] collection of single-variable data sets)
2. The single-variable data set
3. Internal columns/variables within a single-variable data set

Metadata on the DataStore level is mainly of administrative nature, including the name of the DataStore in reverse domain name notation[9], e.g., “no.ssb.fdb”, a description, etc.

The metadata model gets more sophisticated and detailed for levels 2 and 3.

Before going into further detail, we will review/revisit some of the requirements variable metadata need to support in microdata.no:

- Both numeric and alphanumeric measurement values must be handled

[8] For the data versioning aspects we again refer to the paper FAIR Data Versioning in microdata.no accessible from https://www.microdata.no/en/the-architecture-behind-data-versioning/
[9] See https://en.wikipedia.org/wiki/Reverse_domain_name_notation

- Both discrete and continuous variables must be handled
- End users must have easy access to variable definitions
- Evolving code lists for discrete variables must be handled (e.g., yearly changes in Norwegian municipalities)
- Temporal aspects must be explicit and drive system behavior
- Unit types must be explicit and drive system behavior
- Aggregation and other forms of linkage between unit types must be straightforward but also safe
- To assist variable discovery, thematic/topical tags should be associated with variables

Single-variable dataset layout and associated metadata elements

Figure 6 below illustrates components of the logical layout of a single-variable data set, its internal variables, and metadata elements associated with each logical data layout component.

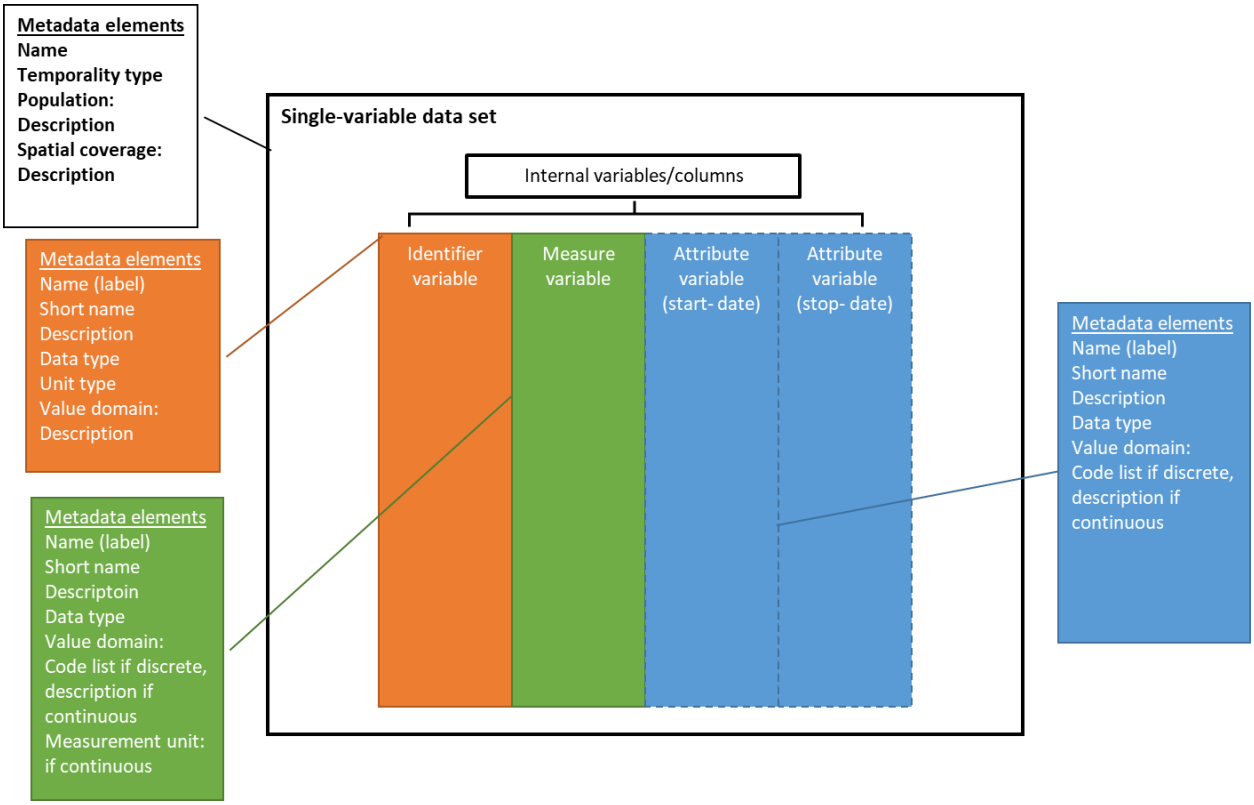

Figure 6 - Illustration of metadata elements for single variable data sets + internal variables

The metadata elements in question are inspired by the core variables models in the GSIM[10]-specification and in the metadata standard DDI (Data Documentation Intitiative), notably the DDI Cross Domain Integration[11] (DDI-CDI).

[10] GSIM = Generic Statistical Information Model. See e.g. https://statswiki.unece.org/display/GSIMclick/Data+Structure

[11] See https://ddialliance.org/ and https://ddialliance.org/Specification/ddi-cdi

SSB has contributed to the GSIM specification, while NSD – Norwegian Centre for Research Data (now a part of Sikt) contributed to DDI from the time the DDI Alliance was founded, and for two decades developed and maintained Nesstar, a reference implementation of the first DDI metadata standard. There has also been considerable collaborative work between the GSIM and DDI communities, and the two specifications/standards align well.

The core data description modules of both GSIM and DDI have been used worldwide and thoroughly tested for many years both by researchers and software developers, and it was therefore considered a safe choice to select microdata.no's core metadata elements.

The microdata-validator

In an attempt to increase understanding of the metadata model and the intimate relationship between data representations and metadata documents, the microdata.no-team has made a tool called "microdata-validator" available on Github[12].

Even though microdata-validator's *main* purpose is to support data owner organizations in preparing data and metadata for ingest into microdata.no, the tool and the published examples in the "docs"-folder[13] should also stimulate a general understanding of the models at play.

We recommend a look at the microdata-validator on Github.

Key choice 8:
Statelessness and immutability

Microdata.no is a multi-user environment where many scripts are executed simultaneously server side. True interactivity on the client side relies on decent response times, and a realistic ability for the user to make and correct errors without noticeable delays or other penalties.

Data and metadata volumes in microdata.no are substantial, and pose different challenges to different components of the system:

- Data composition, data merging, data processing and data analysis operate on large data sets server side.
- Analytic output and metadata documents, smaller but still substantial in volume, are transported continuously to the IDE.

In a setting like this, it would be tempting for system designers to attempt to share and reuse state in the form of intermediate data sets and partial results between users to optimize use of resources such as CPU-cycles and memory.

It is, however, known that state management, and in particular management of shared, mutable state is a major source of complexity in software systems[14], and for system robustness, security

[12] https://github.com/statisticsnorway/microdata-validator
[13] https://github.com/statisticsnorway/microdata-validator/tree/master/docs
[14] Moseley, B., & Marks, P. (2006). Out of the tar pit. *Software Practice Advancement (SPA)*, *2006*. https://github.com/papers-we-love/papers-we-love/blob/master/design/out-of-the-tar-pit.pdf

and manageability, it is generally recommended to rely on *stateless* processes[15] and shared-nothing architectures[16].

We will here demonstrate how sharing and reuse of state is in fact a crucial characteristic of the microdata.no-architecture and how this is perhaps counter-intuitively combined with a stateless and shared-nothing approach.

In a naïve stateless approach, each command in a microdata-script would have to re-run all *previous* commands in the script in one go to execute on the correct, user-specified set of data. The data volumes at hand would doubtlessly make this approach unworkable and not at all compatible with ambitions of interactivity and decent response times.

Enter immutability.

In a situation where no source data can change (i.e., where source data are immutable), repeated execution of the same script/operations would produce identical results every time. With immutable source data, it becomes possible to envision a system where partial results may be cached "mechanically" (see definition below), where the lookup key for a partial result is simply the script[17] that was run to produce it the first time. Partial results exist in many forms and in separate parts of the software stack, including:

- Temporary user-created data files server side (large)
- Temporary output data structures with confidential output/results that get transmitted from the server to the client (small)
- Output that is preprocessed and rendered in the web IDE on the end-user-side (small)

Caching will in general be more efficient with smaller partial results, and the closer to the end-user they reside. In microdata.no mechanical caching occurs on all three levels, with a priority on the most efficient, user-near levels first. In a typical multi-hour user-session, IDE caching and output data structure caching will dominate, minimizing response times as much as possible.

When multiple users collaborate using identical or similar scripts, caches on the two lower levels enable reuse of partial data products and output data structures.

Immutable source data is an *essential requirement* for sharing/caching to happen in a stateless, shared-nothing architecture. For more details on the immutable nature of source data in microdata.no, we again refer to the paper FAIR Data Versioning in microdata.no[18].

Below we discuss briefly how our approach to immutable source data and caching simplifies system design and implementation.

[15] https://12factor.net/processes
[16] https://en.wikipedia.org/wiki/Shared-nothing_architecture
[17] Or in fact, the subset of commands that make changes to data in the user-created data sets.
[18] https://www.microdata.no/en/the-architecture-behind-data-versioning/

“Mechanical” caching

Above we use the term “mechanical” caching to contrast our logic from more complicated and perhaps sophisticated or “intelligent” caching approaches.

In microdata.no, and because we rely on immutable source data, entire classes of problems associated with cache creation and more importantly cache *invalidation* are eliminated. There is no need for complex logic for cache invalidation, simply because caches will always be valid in a system with truly immutable characteristics.

Least used caches are simply removed from the system hourly to keep total storage usage manageable.

This means that users who return to microdata.no after a longer period of absence, are likely to notice that their previously cached data files and output data structures are no longer there, and cache will need to be rebuilt command by command. It should be noted that all this is handled by the system, and that users cannot manage caching logic directly.

Simplicity in system design

We view simplicity in system design as a critical feature of microdata.no. Complexity tends to increase as software systems evolve, and many types of risks emerge and increase with it.

Our strategy to prevent complexity increase involves continuous simplification work in all aspects of the system. Immutability is, perhaps along with the concept of single-variable data sets the most noticeable effect of this strategy to this date.

In our view, our focus on simplicity has not only reduced complexity and risk, but also enabled a range of new and novel opportunities that would be difficult to imagine in a more complex architecture.

Further information on immutability in microdata.no

The immutability characteristics of microdata.no was presented at the software conference Booster in 2019:

https://vimeo.com/325144418