TOWARDS A GENERIC METADATA WAREHOUSE
Ivan Pankov, Pavel Saratchev, Filip Filchev, Paul Fatacean
Technical university – Sofia, 8 St.Kliment Ohridski Boulevard, Sofia 1756, Bulgaria

Abstract
This document describes the creation of a generic metadata model prototype, combining the principles of Generic Modeling and Data Vault architectures. Based on this metadata model a concept for standardization and automation of the data warehouse processes will be designed and implemented.

Key words: Generic, Metadata, Modeling, Data Warehouse, Mapping, Workflow, Automation, Standardization

1. MOTIVATION
Metadata is an essential component for the correct operation and further development of a BI/DWH-system. Despite this fact, most organizations do not have centralized processing and management of metadata. The main reasons for this fact are as follows:

- No established standards - every software vendor has its approach to the preservation and management of metadata.
- Additional resources and investments - resulting from a complex concept and implementation for centralized management of metadata.
- Lack of clear understanding about the benefits which a centralized management of the metadata could bring.

For the reasons above, the goal of this paper is to design and implement a generic metadata-driven prototype for the creation and management of data warehouses with the following characteristics:

- Universal metadata model which can accept any type of metadata and so give a single point of view over all objects, business rules, processes and their dependencies throughout the systems in the organization.
- Optimized and flexible structure with relative small amount of tables, which will facilitate the maintenance and reduce the costs of ownership.
- Standardized and optimized structure of the ETL-processes.
- Metadata-driven generation of ETL-processes and data models.

2. CONCEPTION
Meta data is often defined as „data about data”. This definition is rather abstract and can be interpreted differently in different context. But at this ambiguous aspect of the definition its accuracy and content are expressed. This is data that emerge from the modeling of certain objects and relations between them in a given abstraction level and context. The model in every subsequent abstraction level is the meta model of the previous one. On fig. 1 is presented the dependence of models and metadata at various abstraction levels:

---

1 Metadata standards: DDI, ISO 19115, ISO 19506, ISO 23081, MARC, CWM and many others.
The real objects exist in the real world of abstract level 0. The model of the real object is abstract (simplified) image for a particular purpose\(^2\) in a given context. With the modeling of real objects in the first abstraction level a model 1 was created, which consists of metadata about the real object. A consequent modeling of the model 1 at the second level is a model 2, which in turn consists of a metadata from model 1. In the context of a real object, model 2 is its meta model and model 3 its meta meta model, which in turn is a model of a model 2 and meta model of model 1. Each model in fig. 1 has its own convention and language model\(^3\). By building a meta model on a higher abstraction level we can combine or consolidate different models having different conventions, semantics and metadata.

Metadata that arise from the modeling of objects and related processes are defined as structural metadata. Depending on the area of its origin, the data is classified as technical and business metadata. For example, technical metadata can serve as a description of a table that consists of columns of certain specific format. Example of business metadata could be the definition of a customer - which is a natural or legal person with certain attributes that has purchased a given product or service. The structural metadata gives the user the ability to navigate through its structure and in this way to get an understanding of the real objects without ever having touched them.

Structural metadata is a relatively static data, dependent on how often the underlying object or its model changes. Related objects however undergo changes mainly in the long run. In contrast, metadata derived from the interaction\(^4\) of the objects is dynamic in nature and is known as operational metadata. Operational metadata could be statistics that describe events and processes related to the real objects.

Depending on the area of its origin, metadata is also classified as technical or business. On fig. 2 is presented a classification of technical/business and structural/operational metadata:\(^5\)

---
\(^2\) The purpose of modeling can be simulation, explanation, etc.
\(^3\) The various conventions and model language are represented by the different forms of fig. 1.
\(^4\) In this case the interaction between the objects is modeled.
\(^5\) For other examples of metadata ref. bibl. (Marco, 2000)
In order to manage metadata with different origin and classification type the corresponding model has to be created in a high abstraction level with generic architecture. This will allow the creation of one single generic model for every type of metadata.

The purpose of the Generic Modeling (Fowler, 1995) is that some parts of the model or the whole model could be reused in other models without any or with minor local changes. In addition to this requirement changes of the data model should be minimized or avoided if possible, even with changing business rules. To achieve this goal the business model is divided to business rules and independent from them generic data model. In order for the model to be generic, the architect should follow the principles of generic design: (West, 1996)

1. Candidate attributes should be treated as representing relationships to other entity types.
2. Entities should have a local identifier within a database or exchange file. These should be artificial and managed to be unique. Relationships should not be used as part of the local identifier.
3. Activities, associations and event-effects should be represented by entity types (not relationships or attributes).
4. Relationships (in the entity/relationship sense) should only be used to express the involvement of entity types with activities or associations.
5. Entity types should represent, and be named after, the underlying nature of an object, not the role it plays in a particular context.
6. Entity types should be part of a subtype/super type hierarchy of generic entity types in order to define a universal context for the model.

Figure 2: Classification of Metadata
The model produced with the Generic Modeling architecture is capable to acquiesce with the changes of business rules without or with small modifications. Its standardization (ISO, 2007; ISO, 2012) allows it to be combined with other generic models and to be exchanged (ISO, 1994) between different organizations. Data is presented very consistently without any denormalization, which leads to the elimination of any anomalies.

Despite the advantages of this approach there are also some drawbacks. Mostly, they are reflected in the complexity of the data model. Large number of entity classes and their relations leads to extremely large and complex data models that are practically incomprehensible in scope and can be managed only with a specialized software. The extensive normalization of the generic models leads to significantly lower performance of the queries.

To avoid these disadvantages we'll apply the Generic Modeling principles to the Data Vault (DV) (Linstedt, et al., 2009; Linstedt, 2011) architecture. The DV-architecture is optimized for storage and handling of the so called raw data and consists of three main types of entities: Hubs, Satellites and Links.

Hubs consist of lists of unique business keys. Example of hub can be the table of the entity Product which will contain all ProductNr (product numbers) of products that are known in the system as well as several technical attributes which are recorded by the ETL-processes and can be used for control such as: SQN, LOAD_DTS, EXTRACT_DTS and REC_SRC - sequence, load date timestamp, extract date timestamp and record source.

Satellites consist of descriptive data for business keys and/or their associations. They are built in the form of slowly changing dimensions (SCD 2) (Kimball, 1996) and contain all historized information about the hubs or connections including the technical attributes listed above and LOAD_END_DTS - load end date timestamp, which contains the date on which the data set got a new version in its history.

Links in their turn consist of unique lists of associations which are representing the relations between two or more business keys. They define the interactions between business objects (hubs).

With the separation of the attributes in hubs, satellites and links, DV presents a highly flexible architecture with the following characteristics:

- Changes in the data model are mostly carried out by adding additional links and satellites.
- Different models can be just put together with creation of additional links and satellites.
- Historization is done only within the highly denormalized satellites, which allows effective management of their life cycle.
- Classification of the tables makes easier the navigation and orientation through the model and facilitates the standardization of the ETL-processes.
- Model is optimized for storing the data but not for querying the data. The advantages of this architecture however outweighed this disadvantage.

By combining the principles of Generic Modeling with the Data Vault structure a model can be created, which is generic enough to contain all kind of metadata and still be manageable.

---

6 For ISO 15926 there are more than 10,000 predefined entity classes.
7 Analog to the attribute VALID_TO
3. MODELING OF METADATA

To achieve our goal (s. ch. 1) we'll start the modeling in a high abstraction level. Every single object, which can exist alone will be presented as an object within the H_OBJECT and will have certain attributes respectively in the hub H_ATTRIBUTE. Objects and attributes will have certain types H_OBJECT_TYPE and H_ATTRIBUTE_TYPE. On fig. 4 is presented the first draft of the model:

Every object can have any number of attributes, and each attribute can be owned by any number of objects. Relationship between the entity’s objects and attributes will have cardinality (n:m) and be implemented through an additional entity L_OBJECT_ATTRIBUTE. In the same way the relationships between all other entities will
be expressed. Activities and associations are to be modeled with separate entities which are the links, thus the third and fourth principles of Generic Modeling are observed. Relations between the attribute type and attribute \text{L\_ATTRIBUTE\_ATTRIBUTE\_TYPE} and object with object type \text{L\_OBJECT\_OBJECT\_TYPE} are modeled on the same principle. Object types can also have attributes, which is modeled by their relation with the link \text{L\_ATTRIBUTE\_OBJECT\_TYPE}.

All of the entities are named after their main character, which is the requirement of the fifth generic principle. The prefixes H\_HUB and L\_LINK imply the type of the table, which is a hub and a link respectively. Every entity model will have an artificial primary key (technical key), which is the second principle of the Generic Modeling. Because of the fact that technical keys do not change\(^8\) modeling with hub and links results in one extremely stable and yet flexible architecture. Changes in the structure of the model are implemented mainly by adding new hubs. They interact with other entities using links without changing the consistency in the rest of the model.

The sixth principle of the Generic Modeling demands that the entities have to be a part of sub/super type hierarchy to determine the universal context of their design. To satisfy this principle and extend the meaning of the relation in the model the entities H\_RELATION and type of relation H\_RELATION\_TYPE will be added, as well as the links \text{L\_OBJECT\_RELATION\_TYPE}, \text{L\_OBJECT\_RELATION} and \text{L\_OBJECT\_RELATION\_TYPE} representing the relations, shown on fig. 5.

The hubs H\_RELATION\_TYPE and H\_RELATION are used to classify all possible kinds of references - hierarchies, associations, relations, connections, etc. between the entities H\_OBJECT and H\_OBJECT\_TYPE. The relations will be provided by the hub H\_RELATION and the links \text{L\_OBJECT\_RELATION} and \text{L\_OBJECT\_RELATION\_TYPE}, which will be done by double referencing the hub with H\_OBJECT and H\_OBJECT\_TYPE. The double reference will be done by the usage of two foreign keys in the link, both of which will reference the primary key of the corresponding hub. With the use of hub H\_RELATION the reference will be performed in a certain context, which satisfies the requirement of universal context of design.

![Figure 5: Second Draft of the Generic Metadata Model](image)

A detailed presentation of the link \text{L\_OBJECT\_RELATION} with the hubs H\_OBJECT and H\_RELATION is shown on fig. 6.\(^9\)

\(^8\) The business keys should not be changed, because they are used to uniquely indentify the objects. Their change can compromise the consistency of the entire model.

\(^9\) The link \text{L\_OBJECT\_RELATION\_TYPE} is modeled in the same way.
Figure 6: Hubs H_OBJECT, H_RELATION and Link L_OBJECT_RELATION

Hub H_OBJECT has an artificial primary key H_OBJECT_ID and one composite business key consisting of SOURCE_KEY and OBJECT_KEY. SOURCE_KEY is the key of the source from which the object keys have been imported. It is composite because the business keys of different objects from different sources may have the same values. H_OBJECT is the only hub with a composite business key. All other hubs have one artificial primary key, one business key and two technical attributes LOAD_DTS and RECORD_SOURCE, which are common attributes in the Data Vault modeling representing the date on load and source of the data.

Link L_OBJECT_RELATION has one attribute L_OBJECT_RELATION_ID which plays the role of an artificial primary key and a combination of three foreign keys, one to the primary key of hub H_RELATION and two to the primary key of hub H_OBJECT (H_OBJECT_ID and REL_H_OBJECT_ID). This modeling technique could be applied to model a hierarchy with unlimited depth and structure. To demonstrate this approach let's model an organization with the following characteristics:

- The organization O1 is divided into two main departments D1 and D2.
- The business activity of department D2 take place in two regions, which are managed by the units U1 and U2.
- With expanding the organization in the future an emergence of new departments and units is expected.
- Both new and old departments and units may also have their subdivisions.
- The organization's structure should remain flexible, allowing reformation of departments, units, divisions and subdivisions.

The model of the described above structure is a hierarchy with levels of certain elements that can be represented as follows:

The metadata of the model shown on fig. 6 will be inserted in link L_OBJECT_RELATION as well as in hubs H_OBJECT and H_RELATION, which is presented on tab. 1. In hub H_OBJECT the metadata of the organization, departments and units is inserted. The primary keys are double referenced in the link L_OBJECT_RELATION as foreign keys, modeling a hierarchy between them. In H_OBJECT_ID are placed the keys of the parent element, while the REL_H_OBJECT_ID these of their direct descendants. A reference to H_RELATION classifies the relationship between the object and its related object as “parent-child”. By expanding the organization’s new departments and units will be added at the appropriate level and their metadata will be loaded accordingly.

Presented at this stage entity types are the hub and the link that contain only business keys of real objects and the relations between them. Metadata however comprises not only business keys, but many additional attributes. Typical for these attributes is that they describe the real objects and their references and change over time. In the Data Vault modeling all attributes of this type are placed in the satellites, which on their side are referencing a hub or connection.

10The combination of SOURCE_KEY and OBJECT_KEY must be unique.
11LOAD_DTS and RECORD_SOURCE are used in all hubs, links and satellites in the model.
12The other links in the model have the same standardized structure similar to L_OBJECT_RELATION.
Figure 7: Organization’s Structure: Organization, Department, Unit

Table 1: Metadata of Organization: Organization, Department, Unit

On fig. 8 the corresponding satellites for the hub H_OBJECT and link L_OBJECT_RELATION are shown:
The cardinality of hubs and links to their satellites is (1:n), every primary key has at least one reference key in its satellite. The satellites are built on the principle of slowly changing dimension (SCD 2) and contains the entire history of the hub or connection. With the satellites to their corresponding hubs and links the model can be presented in its final shape, which is shown on fig. 9.

The presented satellite's attributes can be extended for the different requirements and needs. In addition there could be more satellites for one hub or link, containing completely different business attributes. This approach is often applied to data which comes from different sources.

The structure of the presented metadata model could be used for other data models. The generic and standardized tables allow the standardization of the ETL-processes, which on itself would bring a lot of benefits in the conception, implementation and administration of the BI/DWH-system.
4. STANDARDIZATION OF ETL-PROCESS

The processes of Extraction Transformation Load (ETL) include all activities from procurement data of different sources, processing and cleaning to the final loading into the database. Their concept and structure is based on the business model of the organization and mostly on the built data models. The implementation of the ETL-process is performed using programming languages such as C, JAVA, PL / SQL, etc., or with specialized software such as OWB, KETTLE, Informatica etc. but most often by a combination of the two alternatives. The main principles for the development of ETL-processes consist of:

- **Completeness** - the correct design and implementation of all relevant processes for the data processing.
- **Adaptability** - ETL-processes must be easily adaptable to any changes in the data model or business logic.
- **Stability** - ETL-processes must be stable at high load and volume of data. The loading time should not increase exponentially with the increase of the data volume.
- **Simplicity** - the simplified structure of the ETL-process helps their clarity. Complex structure automatically lead to complex and expensive implementation and administration of the entire system.
- **Consistency** - ETL-processes should be designed to maintain the data consistency. It is essential to save automatically the original integrity of the data in case of intentional or unintentional process interruptions caused by inconsistent data, system crash or other reasons.
- **Standardization** - ETL-processes should have standardized structure which helps to reduce the risk of implementation errors, clarity and maintenance of the entire system.

The cost of conception, implementation and maintenance of ETL-processes often exceed 50% of the entire budget. Therefore the compliance with above described principles is crucial for optimizing the costs for development and administration of the system. For this purpose we'll present a high-level design (HLD), which aims to establish the main characteristics of ETL-processes in the form of pseudo code and diagrams. The ETL-processes consist primarily of two main parts: mapping and workflow.

The mapping is a directed graph, which comprises of the following types of building blocks: (Informatica, 2012)

- **Sources** - the starting point from the mapping, where the data is read from like: flat files, XML-files, tables, queues, etc.
- **Transformations** - the processing elements of the mapping like: filters, lookups, expressions, etc.
- **Mapplets** - partial mappings, which encapsulate a particular logic and algorithm and can be used inside other mappings.
- **Targets** - the end point of the mapping, where the data is written after the processing could be: flat files, XML-files, tables, queues, etc.

On fig. 10 is demonstrated a universal and standardized HLD, which structure will be used for the implementation of all mappings for loading the tables of type hubs and satellites:

In the first step of fig. 10 the data is read from the staging table object type STG_OBJ_TYPE. In the staging table data is stored temporally when reading it for the first time from its source. In the next step the data is transferred to the expression *delta detection*. Here the data is marked as:

- **New data** - this data cannot be found in the table H_OBJ_TYPE and S_OBJ_TYPE so it will be distributed by the filter FIL/ROUTER to expression EXP1_NEW_HS. Afterwards it is loaded into the respective hub and satellite tables.
- **Changed data from type SCD1 or SCD2** - such data is found in the table H_OBJ_TYPE but not in S_OBJ_TYPE. If we are interested in saving the history, then it will be forwarded to EXP2_NEW_S. The new version will be saved and the old one terminated. In case we are not interested in saving the history the new data will be moved to expression EXP3_UPD_S and the old data will be overwritten.
- **Old data** - this data will be filtered out and ignored from the further processing.
Figure 10: High Level Design for Mapping

The workflow is also a directed graph, which consists of one or several mappings and additional control elements which may be performed sequentially or in parallel. The concrete functions of the workflows differ by the different vendors, but generally they can be described as: (Informatica, 2012)

- Summary element that can combine mappings, sub-processes and other control elements on order to perform the desired logic.
- Additional abstraction layer for physical attributes from the mappings such as: a commit interval, log-in, schema, database, time schedule, etc.

On fig. 11 a universal and standardized HLD, which structure will be used in the implementation of all ETL-processes is presented:

Figure 11: High Level Design for Workflow

In the first step of fig. 11 the start of the process START_ETL_PROCESS is modeled. Any process that changes the data must ensure its consistent state, even in case of (un)intentional interruption of the process. This will be achieved with the generic sub-process SEC_CONS_STATUS, which will maintain the original status of the data. After successful saving of the consistent state the sub-process, which loads new or changing existing data
INS_UPD_DATA will be started. The following three states of the process termination are possible, depending on the processed data, business rules for loading and stability of the infrastructure:

- **NOT OK** - the sub-process INS_UPD_DATA is interrupted or terminated with errors. For recovering of the consistent state the generic sub process RECOV_CONS_STATUS will be started. After its completion the responsible specialists will be informed with the task or sub-process INFORM.

- **WARN** - the sub-process INS_UPD_DATA is completed with warnings. In this case, the experts must determine whether or not someone hast to be informed and/or the next process can be started.

- **OK** - the sub-process INS_UPD_DATA has completed successfully and the next process can be started.

The presented standard HLDs for mappings and workflows can be described and defined with metadata. In the next chapter an application with graphical user interface (GUI) will be presented, which allows the simplified management of the metadata and automatic generation of the mappings and workflows.

5. **METADATA MANAGEMENT AND ETL-AUTOMATION**

5.1 Metadata Manager Architecture

Having a generic metadata model brings a lot of advantages, but also complexity with it. As by just looking at the stored data, it is not so obvious what are the different entities in the model and how do they correlate to each other. The user can write complex and long SQLs to find out the information and to manage the data, but this would be quite cumbersome and tedious. The definition and the design of the ETL-processes for the DWH can on itself be also very complex, challenging and time consuming task. These problems can be overcome to large extend, by the creation of an application with graphical user interface (GUI) in this case the Metadata Manager (MM). Such a GUI will bring transparency and will reduce significantly the complexity of definition and management of the entities and their relations in the metadata model.

The reduced complexity will allow the DWH-architects to concentrate on the DWH-system and process definition and not on how to define it. As a consequence, the time and costs needed for the conception and implementation of the ETL-processes will decrease drastically. Being standardized, the structure of the generic metadata model is not a matter of change. This enables the standardization of the GUI as well, which in its turn allows creating it once, and reusing it for any DWH conception and implementation. Having the standardized metadata model, a set of operations that the UI shall provide to the end user can as well be defined.

On fig. 12 a use case diagram depicts the actions necessary for metadata definition, which are available for execution via the GUI. As it can be seen Create, Read, Update and Delete (CRUD) functionality is provided for each of the tables in the model. This will give the DWH-architect the possibility to monitor existing entities’ definitions, as well as to create new ones.

As it can be seen on the diagram, the DWH architect triggers any of the CRUD operations via the Metadata Manager GUI. Based on this, via HTTP Request, the business logic is invoked. The Metadata Manager performs a CRUD against the database (DB), with the help of DB specific ODBC driver, which eventually manipulates the data stored in the Schema according to the triggered operation and given context. The corresponding execution status and updated data set is then returned back the chain to the end user (not depicted on the diagram).

Having defined what needs to be done, it is now time to dive a bit deeper in the technical details of how this can be achieved. Basically at this point a decision has to be made of what DB will be used for storing the model, which programming language will be used for programming the Metadata Manager and what API’s will be provided to access the application. When deciding on this, several points must be considered:

- **License costs** – most of the commercial tools have free nonprofit version, they can become quite expensive when used for profit purposes.

- **OS and DB independence** – as most companies would not switch the DB or OS they are using.

- **Security** – it is an essential part of every corporate application. The application needs to therefore meet the enterprise’s standards.
Figure 12: Use Case Diagram for Metadata Management

On the fig. 13 a high level overview of the MDM application’s architecture which can which can be adopted by any enterprise is provided.

The metadata model is stored in a relational database\(^\text{13}\). As the application should be platform and DB independent, Java is a good option for realizing the implementation. The database access is established via Open Database Connectivity (ODBC) driver and Java Persistence API (JPA). The application's functionality is available via HTTP and Web Services (WS), which is the reason for employing a web application server like Tomcat. The user interface itself is implemented with JavaServer Faces (JSF). Any of the available frameworks which provide automation of the construction of JSF UI (e.g. Prime Faces, webMethods etc.) can be employed for speeding up the development process. By exposing the business logic as WS, the application becomes Service-Oriented Architecture (SOA) enabled. This provides the possibility for easy integration of the software in the company’s environment as well as in 3-rd party frameworks and products.

The entities and relations of the standardized metadata model can be easily mapped to Objects (in terms of programming languages e.g. Java, C++ etc.). On the following fig. 14 a class diagram is presented, which depicts the metadata model’s entities in terms of Java classes. As it can be seen the entities of the model correspond one to one to the classes defined for the Java implementation.

---
\(^{13}\) In this case Oracle is used, but basically any DB can be used for storing the model.
Once the metadata is defined and stored, the ETL-processes can be implemented. The conventional way of doing this is taking an ETL tool (e.g. Informatica) and manually defining the processes with it. Having a standardized generic model with metadata and predefined ETL-processes flows, is a necessary prerequisite for the implementation and usage of an automated ETL-process generation mechanism.
5.2 Automatic Generation of ETL-Processes

An automated mechanism for ETL-process generation allows the quick realization and simplified adjustment of new and existing processes in a DWH, thus reducing the costs and time to value drastically. Another requirement that has to be fulfilled is having an ETL tool that exposes its functionality as API. As one of the most popular ETL tools for large DWH-systems, and as its API is exposed and documented, Informatica is used for the automated ETL process definition. Having all the prerequisites met, an additional module is implemented and deployed to a Tomcat server. That module reuses the database access functionality that has already been developed for managing the metadata. Having access to the model definition and using an ETL-API, the module enables the generation of the ETL-processes, which are predefined as metadata. The use case for this functionality is presented on the following use case diagram.

![Use Case Diagram for Generation of ETL-Processes](image)

Having defined the necessary artefacts for the ETL processes, the automated generation can be triggered via the Metadata Manager UI, as displayed on the diagram. After receiving the Request from the UI, the Metadata Manager triggers the creation of the ETL processes in Informatica, using its Java API.

After the completion of the procedure, the generated ETL processes can be seen and verified in Informatica. The usage of the automated generation mechanism simplifies the adjustments and redesign of large number of processes – this is where the real power of this approach becomes visible and can be evaluated.

6. CONCLUDING REMARKS

By adopting of the presented generic metadata repository the organizations may lay down the foundation of a centralized and efficient management of all types of models and metadata. Thus enabling the cataloging of technical and business processes in the organization, monitoring the structural dependences and restoring their old versions and changes. The presented model has been designed to be generic and can be extended or adapted to the requirements of any organization. Standardized structure of the entities contributes to rapid orientation in the model and allows the establishment of standard methods and processes for their loading and querying. This in turn reduces the cost of implementation and administration and contributes to optimal usage of available resources.

Generic concept of modeling is suitable not only for metadata repositories, but also for models that will be created for transactional and master data. The concept of the generic data model is enhanced by a standardized GUI that facilitates the overview and management of the metadata. This contributes to better productivity during the definition of the model and higher quality of its loading routines. Providing the automated mechanism for ETL process generation is the last crucial part for making this solution of great value for any company that opts for using it.
REFERENCES


