An Approach to Evolution Management in Integrated Heterogeneous Data Sources

Darja Solodovnikova(✉), Laila Niedrite, and Lauma Svilpe

Faculty of Computing, University of Latvia, Riga, Latvia
{darja.solodovnikova,laila.niedrite}@lu.lv

Abstract. In this paper we target the current problem of evolution of heterogeneous data sources of a data warehouse. Evolution may be caused by changes in the structure of data sources that are often independent from a data warehouse as well as by changes in information requirements. The solution we introduce in this paper is based on the architecture of a data analysis system that apart from a data highway that collects and transforms data also employs a metadata repository and various tools that provide different kinds of analysis of stored data. The unique feature of our solution is an adaptation component that incorporates mechanisms for automatic discovery of changes in the structure of integrated data sets and propagation of these changes in a data warehouse and other components of a data analysis system. In addition to the presentation of our approach, we give details of approbation of our software prototype in the case study system.

Keywords: Evolution · Data warehouse · Change propagation · Metadata · Heterogeneous data

1 Introduction

Data warehouses have been used for decades to support the analysis of integrated data. However, before recently, mainly structured data stored in relational databases have been used to populate data warehouses. Due to new technological developments, currently data that should be analyzed in the decision-making process are becoming more and more enormous and heterogeneous and traditional solutions based on relational databases have become unusable to process all these data volumes.

Besides, changes in the structure of large heterogeneous and often independent data sources occur more frequently, but finding a solution to problems caused by this evolution is a more challenging task for several reasons. On one hand, there is currently no standard architecture that is commonly used to support the analysis of heterogeneous data sources. On the other hand, data sources we are working with are often semi-structured and unstructured and it is a complex task to detect and process changes in such sources. And finally, in modern systems data may be generated at a higher rate, and this means that changes should be also handled somehow immediately after they occurred.

The goal of our study is to develop a solution to collect, store and analyze data from multiple heterogeneous data sources as efficiently as possible, while also processing changes in the structure of data that occur as a result of evolution. In our approach presented in this paper we use a well-known data warehouse paradigm and extend it with the processing of semi-structured and unstructured data sets as well as mechanisms for discovery and automatic or semi-automatic propagation of changes in these data sets.

The present paper is an extended version of our paper [20]. In this paper, we provide details of the implementation of the change discovery and propagation tool and results of the approbation of the proposed approach in the case study system. In addition to that, this paper includes the statistical analysis of the supported change adaptation scenarios which demonstrates how well our approach allows to reduce human participation in the evolution management process.

The rest of this paper is organized as follows. In Sect. 2 we review related work. In Sect. 3 we give details of our proposed approach to evolution management. In Sect. 4, the case study system and examples of real-world changes are discussed. In addition to experiments, the overall statistical evaluation of the developed solution is presented in Sect. 5. The paper ends with conclusions drawn based on the evaluation of the proposed approach presented in Sect. 6.

2 Related Work

A great research effort has been devoted to studying the problem of schema evolution in relational databases. The offered solutions to evolution problems in this domain can be classified into two categories: schema adaptation and schema versioning. The goal of approaches in the former category [1] is to adapt just the existing data warehouse schema or ETL processes [2] without keeping the history of changes, while approaches in the latter category [3–5] maintain multiple versions of schema that are valid during some period of time. Another related research papers [6,7] deal with a formal description of information requirements and their influence on the evolution changes. All the above-mentioned approaches target data warehouses implemented in relational database environments, thus, they cannot be utilized directly to perform adaptation of data warehouses that integrate big data sources.

Several articles reviewing current research directions and challenges in the fields of data warehousing and Big Data mention also evolution problems. The authors in [8] mention dynamic design challenges for Big Data applications, which include data expansion that occurs when data becomes more detailed. A review paper [9] indicates research directions in the field of data warehousing and OLAP. Among others, the authors mention the problem of designing OLAP cubes according to user requirements. Another recent vision paper [10] discusses the variety of big data stored in the multi-model polystore architectures and suggests that efficient management of schema evolution and propagation of schema changes to affected parts of the system is a complex task and one of the topical issues.

Various recent studies have been devoted to solving the evolution problems in the Big Data context. In the paper [11], we summarized the research made in the field of

Big Data architectures and analyzed available approaches with the purpose to identify the most appropriate solution for the evolution problems. The most relevant studies that deal with evolution problems are also discussed in this section.

We have also found several studies that deal with evolution problems in systems aimed at Big Data storage and analysis. An architecture that exploits Big Data technologies for large-scale OLAP analytics is presented in the paper [12]. The architecture supports source data evolution by means of maintaining a schema registry and enforcing the schema to remain the same or compatible with the desired structure.

Another study that considers evolution is presented in the paper [13]. The author proposes a data warehouse solution for Big Data analysis that is implemented using MapReduce paradigm. The system supports two kinds of changes. Slowly changing dimensions are managed with methods proposed in [14] and fact table changes are handled by schema versions in metadata. Unlike our proposal, the system does not process changes in heterogeneous data sources.

A solution to handling data source evolution in the integration field was presented in the paper [15]. The authors propose the Big Data integration ontology for the definition of integrated schema, source schemata, their versions and local-as-view mappings between them. When a change at a data source occurs, the ontology is supplemented with a new release that reflects the change. Our approach differs in that the proposed architecture is OLAP-oriented and is capable of handling not only changes in data sources, but also information requirements.

There is also the latest study presented in [16] dedicated to evolution problems in heterogeneous integrated data sources. The authors propose to use deep learning to automatically deal with schema changes in such sources.

A tool for evolution management in multi-model databases is presented in the paper [17]. The solution includes a multi-model engine that is a mediator between a user and databases of various formats. The engine accepts commands of a special schema evolution language and propagates them to affected entities in multi-model databases. The integration of multiple databases is based on a special abstract model. In contract to the solution presented in the paper [17], we use different architecture that physically integrates data at different levels, as well as employs a data warehouse which allows to perform OLAP operations.

In the paper [18] we analyzed studies dedicated to metadata employed to describe heterogeneous data sources of data lakes and we concluded that none of the examined metadata models reflect evolution. For our solution, we adapted the metadata model proposed in [19] to describe data sources of a data lake. The authors distinguish three types of metadata: structure metadata that describe schemata of data sources, metadata properties and semantic metadata that contain annotations of source elements. In our approach, we extended the model with metadata necessary for evolution support.

3 The Proposed Approach to Evolution Management

In this section, let us concentrate on the description of our solution to topical evolution problems. Our approach allows to perform OLAP operations and conduct other types of analysis on integrated data from multiple heterogeneous sources as well as detects evolution in these sources and facilitates evolution management.

3.1 Data Warehouse Architecture

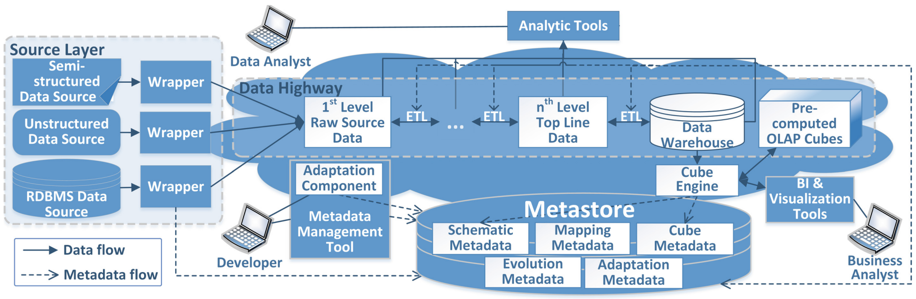


Fig. 1. Data warehouse architecture for evolution management.

Our approach is based on the system architecture that is composed of various components that provide data flow and processing from the source level to the data stored in the data warehouse. The interaction of these components is shown in Fig. 1. A more detailed description of the architecture is provided in the paper [21].

The central component of the architecture is a data processing pipeline that we call data highway. We followed the idea and the concept of the data highway first presented in [14]. At the source level, data is obtained from various heterogeneous sources (including big data sources) and loaded into the first raw data level of the data highway in its original format. Different types of data sources are supported in our approach: structured (database tables), semi-structured (such as XML, JSON, CSV, etc.), and unstructured data (text or PDF files, photos, videos). Then, data for each subsequent data highway level is obtained from the previous level by performing transformations, aggregations and integrating separate data sets. The number of levels, their contents and the frequency of their updating are determined by the requirements of a particular system. The final level of the highway is a data warehouse which stores structured aggregated multidimensional data. Various types of analysis are provided on pre-calculated OLAP cubes introduced to improve query performance as well as data at various levels of the data highway.

Another essential component of the architecture is the metastore that incorporates five types of interconnected metadata necessary for the operation of various parts of the architecture. Schematic metadata describe schemata of data sets stored at different levels of the highway. Mapping metadata define the logic of ETL processes. Information about changes in data sources and data highway levels is accumulated in the evolution metadata. Cube metadata describe schemata of pre-computed cubes. Adaptation metadata accumulate proposed changes in the data warehouse schema as well as additional information provided by the developer required for change propagation. We give an overview of the metadata that we maintain in the metastore to support the adaptation of the system after changes in data sources and information requirements in Subsects. 3.3 and 3.4.

Metadata at the metastore are maintained via the metadata management tool that integrates the unique feature of the architecture - the adaptation component that is aimed at handling changes in data sources or other levels of the data highway. Change handling is performed in the following steps:

- Changes are detected by the change discovery algorithm of the adaptation component and information about them is recorded in the evolution metadata.
- Change handling mechanism of the adaptation component determines possible scenarios for each change propagation. The information about possible scenarios for each change type is stored in the adaptation metadata.
- Scenarios that may be applied to handle changes are provided to a data warehouse developer who chooses the most appropriate ones.
- The adaptation component leads the implementation of the chosen scenarios. Since certain scenarios may require additional data from the developer, such data are entered by the developer via the metadata management tool and stored in the adaptation metadata after the respective scenario has been chosen.

3.2 Atomic Change Types

Various kinds of changes to data sets employed in each level of the data highway must be handled by the adaptation component. Based on possible operations that may be applied to various elements of the schematic metadata model, we defined a set of atomic change types that must be supported in our proposed solution. These types classified according to the part of the metadata model they affect follows:

- *Schematic changes*: addition of a data source, deletion of a data source, addition of a data highway level, deletion of a data highway level, addition of a data set, deletion of a data set, change of data set format, renaming a data set, addition of a data item, change of a data item type, renaming a data item, deletion of a data item from a data set, change of a data item type, addition of a relationship, deletion of a relationship, addition of a mapping, deletion of a mapping;
- *Changes in metadata properties*: addition of a metadata property, deletion of a metadata property, update of a value of a metadata property.

3.3 Schematic, Mappings and Evolution Metadata

To accumulate metadata about the structure of data sources as well as data sets included at various levels of the data highway and to maintain information about changes that occur in them, we use the metadata model presented in Fig. 2.

The class *Data Set* is used to represent a collection of *Data Items* that are individual pieces of data. The class *Data Set* is split into three sub-classes structured, semi-structured and unstructured data set, according to the type and format. A data set may be obtained from a *Data Source* or it may be a part of a *Data Highway Level*. Relationships between data items in the same data set or across different data sets, for example, foreign key, composition, predicate or equality, are implemented by means of an association class *Relationship*.

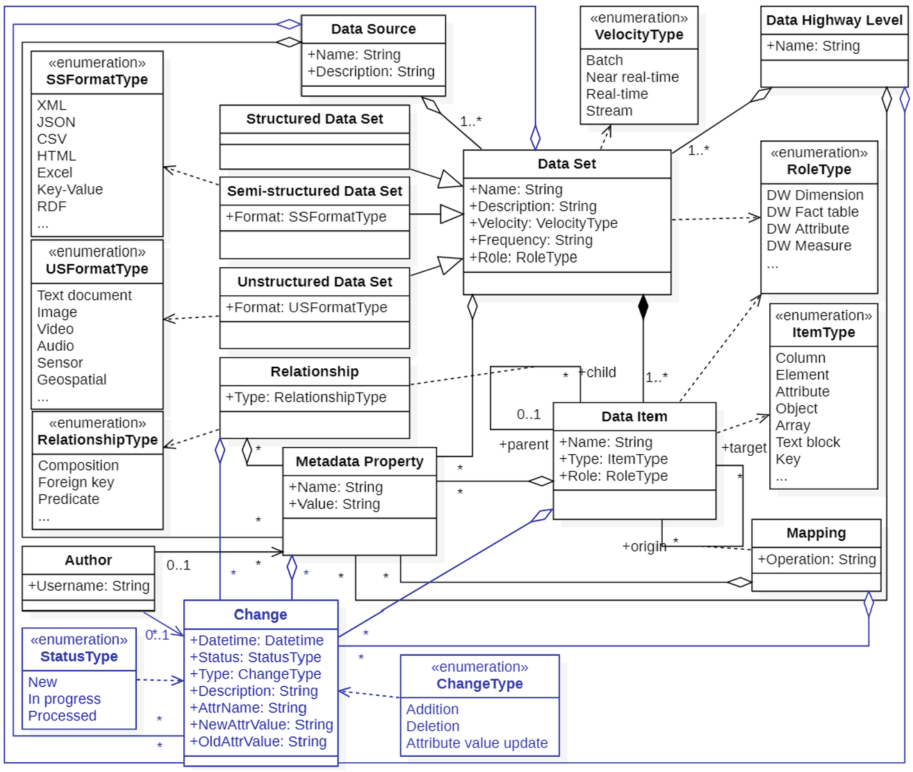


Fig. 2. Schematic and evolution metadata model [18].

We introduced an association class *Mapping* to make it possible to follow the lineage of data sets. A mapping defines a way how a target data item is derived from origin data items by a transformation function stored in the attribute *Operation* of the association class *Mapping*.

To represent other characteristics of data apart from their structure, we included a class *Metadata Property*. Examples of metadata properties are file or table name, size, character set, data type, length, precision, scale, or even mechanism used to retrieve data from a data source. Each property is represented by a name:value pair to allow for some flexibility as metadata properties of different elements may vary considerably. If a property has been entered manually by a user, we associate such property with an *Author* who recorded it.

Finally, a class *Change* is included in the model to store information about evolution. Each instance of the class *Change* is associated with one of the classes in the model which determines the element of the model that was affected by the change. In the metadata, we store the date and time when the change took place, type of the change and status that determines whether that change is new, already propagated or being currently processed. If a change was performed manually by a known user, the corresponding *Author* is associated with it.

3.4 Adaptation Metadata

The metadata presented previously allow to describe the structure of data highway levels and properties of data sets and to store data on discovered changes. The metadata essential for change propagation are demonstrated in Fig. 3. The metadata model incorporates a class *Change* from the evolution metadata. Based on data in this class, it is possible to determine the atomic change type that is used to select possible scenarios for change processing.

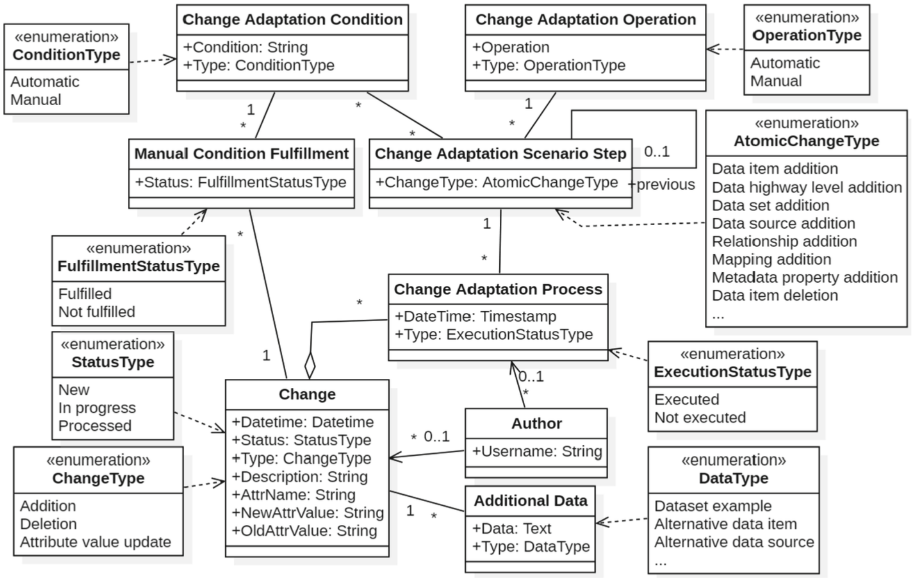


Fig. 3. Adaptation metadata model.

Metadata for Change Adaptation Scenarios. The classes *Change Adaptation Operation*, *Change Adaptation Condition* and *Change Adaptation Scenario Step* are intended for storing information about change adaptation scenarios and their components.

Instances of the class *Change Adaptation Operation* are operations that must be performed to handle a change in the system. An operation is assigned a type that indicates whether it can be performed manually or automatically. In the former case, a textual description of what the developer must do to perform the operation is stored in the attribute *Operation*. In the latter case, the name of the procedure to be executed is stored there.

Although a type of change can be determined at the time of its occurrence, it does not guarantee the existence of an unambiguous change adaptation scenario. There are various conditions under which scenarios can branch out. To store these conditions, the class *Change Adaptation Condition* was introduced. Each instance of this class has two attributes: type of the condition and the condition definition. Manual conditions allow developers choose scenarios that are more suitable for the particular situation if multiple change adaptation scenarios exist. If the condition is executable manually, a textual description of the condition is stored in the attribute *Condition*. If the condition is automatic, the name of the function to be executed is stored there.

Each change adaptation scenario is a series of sequential steps. The steps of each scenario are reflected by the class *Change Adaptation Scenario Step*. Each step of a scenario is associated with an atomic change type, an adaptation operation, a set of conditions that must be fulfilled for the step to be executed, as well as the previous step of the same scenario. Having a link to the previous step maintains the sequence of operations and facilitates adjustments to adaptation scenarios. Multiple adaptation scenarios correspond to each atomic change type.

Change Adaptation Operation and *Change Adaptation Condition* are independent classes and their instances are actually building blocks of scenarios. At the time of the system installation, first, all possible operations and conditions are added to the metadata. Then, scenarios are constructed by selecting appropriate operations and conditions and arranging them in a sequence. This way, the classes *Change Adaptation Operation*, *Change Adaptation Condition* and *Change Adaptation Scenario Step* are pre-filled with data manually before any changes occur. On one hand, such approach allows to easily modify existing scenarios by removing steps or inserting new ones. On the other hand, new instances of these classes may be added later during the usage of the system if new scenarios become necessary.

Metadata for Change Propagation. The classes *Change Adaptation Process*, *Manual Condition Fulfillment* and *Additional Data* were included in the model to support actual change propagation. To store information about the execution of each operation during the change propagation process, we introduced the class *Change Adaptation Process* which reflects the adaptation scenario steps corresponding to each actual change that occurred in the system. Each instance of *Change Adaptation Process* is linked with the adaptation operation via the *Change Adaptation Scenario Step* and is associated with a corresponding *Change*. To keep track of the change propagation process, the status of the operation is also stored, as well as the date, time and the user who executed the operation.

During the change propagation, automatic conditions can be checked before each step since they do not require the intervention of the developer. However, when evaluating a manual condition, it is necessary to keep information about the decision the developer has made. For this purpose, we introduced a class *Manual Condition Fulfillment*.

Moreover, various additional data may be required to be provided by a developer to perform operations and to evaluate conditions. If any additional information is needed, it is reflected by a class *Additional Data*, which is associated with a particular *Change* and stores information on a data type or purpose for which the additional data is used, as well as the data itself.

3.5 Change Discovery in Data Sources

The main task of the adaptation component of our proposed architecture is to detect changes that have taken place and to propagate each discovered change. Most of the atomic changes supported by our approach may be identified automatically by the change discovery algorithm implemented as a part of the adaptation component and described in detail in the paper [22]. Manually introduced changes are processed by the metadata management tool, however, automatic change discovery is triggered when any new data are loaded from data sources into the data highway by wrappers or during ETL processes.

Initially, the change detection algorithm gathers schema metadata and properties of existing data sources and data highway levels in temporary metadata. For metadata collection, special procedures are used depending on the format of data sets. Structured, semi-structured and unstructured data sets are handled by different procedures. To identify changes in metadata, a change discovery algorithm first processes data sources and data highway levels, then data sets and data items, and finally mappings and relationships. For each processed element, the algorithm compares metadata that describe the current structure and features of data used in the system with the metadata available at the metastore and identifies differences that determine types of atomic changes occurred. The identified changes are then saved in the evolution metadata.

3.6 Change Propagation

After changes have been detected and recorded in the evolution metadata, the adaptation component must first generate potential adaptation scenarios for each change and then execute scenarios according to branching conditions. We predefined adaptation scenarios for each atomic change type and operations and conditions necessary for each scenario. The full list of scenarios along with their components can be found in the paper [20], but we will focus on the details of change handling mechanism in this subsection.

In order to successfully propagate any change to the system, the change handling mechanism analyzes schematic and mapping metadata as well as change adaptation scenarios, operations and conditions predefined for each atomic change type. Based on the analysis results, the mechanism creates metadata necessary for change handling and leads the change propagation process by evaluating automatic conditions and performing automatic operations. There are two stages of the change handling mechanism described in detail in the following subsections.

Creation of Change Adaptation Scenarios. The goal of the first stage of the change handling process is to determine potential change adaptation scenarios and create initial instances of the classes *Change Adaptation Process* and *Manual Condition Fulfillment*.

The high-level pseudocode of the initial change processing stage implemented as a procedure *CreateChangeAdaptationProcess* is presented as Algorithm 1. First, changes with the status *New* are selected. These are changes that have not been processed yet. Then for each new change, the atomic change type is determined by the function *GetChangeType*. After that, all scenario steps predefined for the determined change type are selected and the corresponding instances of the class *Change Adaptation Processes* are created. Then, manual conditions for the current scenario step are selected, linked with the currently processed change and saved as an instance of the class *Manual Condition Fulfillment* with the status *Not fulfilled*. If, according to scenario definition, the same manual condition must be evaluated for multiple scenario steps, only one instance of the class *Manual Condition Fulfillment* is created. Finally, the status of the change is updated to *In progress* so that handling of this change can be considered as initiated.

```
Procedure CreateChangeAdaptationProcess()
    while exists Change C where C.status = 'New' do
        vProcessCreated ← false;
        vChangeType ← GetChangeType(C);
        if vChangeType is not null then
            foreach Scenario step S that exists for vChangeType do
                P ← InsertChangeAdaptationProcess(S,C);
                foreach Condition M that exists for scenario step S where M.type =
                  'Manual' and not exists Manual condition fulfillment for Change C
                  and Condition M do
                    InsertManualConditionFulfillment(C, M);
                end
                vProcessCreated ← true;
            end
            if vProcessCreated then
                UpdateChangeInProgress(C);
            end
        end
    end
end
```

Algorithm 1. Creation of change adaptation scenarios.

Execution of Change Adaptation Scenarios. When all initial metadata are created, the second stage of the change handling mechanism - execution of change adaptation scenarios is run. Scenario execution is based on condition checks and execution of operations. Manual operations and conditions require the intervention of a developer. In such a case, the algorithm is stopped and resumed only when the developer has made his or her decision regarding manual conditions or performed the specified operation.

The Algorithm 2 demonstrates the pseudocode of the procedure *RunChangeAdaptationScenario* that executes an adaptation scenario for a specific change. First, the function *GetChangeAdaptationScenarioSteps* retrieves adaptation process steps as instances of the class *Change Adaptation Process* created during the previous stage of the mechanism. Then for each step that has not been previously executed and has a status *Not executed*, the change propagation is continued only if it is necessary to perform an automatic operation, as well as the corresponding conditions are met. Manual conditions are checked using instances of the class *Manual Condition Fulfillment*. For automatically evaluable conditions, a function name is obtained from the attribute *Condition* of the class *Change Adaptation Condition*. By running the corresponding function it is possible to evaluate the condition fulfillment. Following the same principle, the procedures for performing operations of the change adaptation scenario are also executed. Their names are stored in the column *Operation* of the class *Change Adaptation Operation*. After execution of each process step, that the status of the process step is set as executed.

```
Procedure RunChangeAdaptationScenario(C: Change)
    Steps ← GetChangeAdaptationScenarioSteps(C);
    foreach process step S in Steps do
        if S.Type = Not executed then
            O ← GetProcessStepOperation(S);
            if O.OperationType = Automatic and ConditionsFulfilled(C, S) then
                ExecuteAdaptationProcessStep(C,O);
                S.Type ← Executed;
            end
            exit;
        end
    end
end
```

Algorithm 2. Execution of change adaptation scenarios.

3.7 Change Adaptation Scenarios

After any changes have been detected, the adaptation component of our proposed architecture must first generate potential adaptation scenarios for each change and then execute scenarios according to branching conditions. We have predefined adaptation scenarios for each change type and operations and conditions necessary for each scenario.

Several change adaptation scenarios for real-world changes that occurred in the case study system are described in Subsect. 4.4, however, in our paper [20] the detailed explanation of each scenario is given.

In general, for changes that involve addition of new data items, mappings, relationships, data sets, data sources or data highway levels, the adaptation component requires human participation to handle the change. The scenarios for these changes are mainly semi-automatic and include operations for additional metadata definition, provision of example data sets and/or discovery of structure of newly created elements.

However, when existing data items, mappings, data sets, data sources or data highway levels are deleted, usually multiple scenarios exist for each change. The goal of such scenarios is to try to replace the missing element with others or remove it and other dependent elements from ETL processes (so that this element is no longer updated) if replacement is not possible. If a relationship is deleted, the scenario is selected based on the relationship type. For foreign key, mappings that involve that foreign key are determined automatically and a developer must modify them. If a composition relationship is deleted from an XML document, the structure of that document is updated in the metadata.

Changes that involve renaming any element are usually processed automatically and only one scenario is available for each of such change type. The idea of the scenario is to update the metadata with a new name of an element.

Change of a data set format is processed semi-automatically. The idea of the available scenarios is to determine mappings that define transformations for the changed data set so that a developer can manually redefine them. If a structure of a data set changed along with its format, metadata describing a new structure are automatically gathered before other operations.

If data type of a data item is changed it must be determined whether a new type can still be used in mappings. Otherwise, the developer must manually modify ETL processes and mapping metadata.

Finally, any changes to metadata properties require human participation. In many cases changes in metadata properties impact the definition of ETL processes and mappings, however, it is rarely possible to determine dependent mappings automatically. If it is not possible, the developer is just informed about the change and must make a decision about the change propagation.

3.8 Implementation

We implemented a software prototype that includes the functionality of the metadata management tool, adaptation component and metadata repository (metastore) of the data warehouse architecture. Other components of the architecture depend on the system that our approach is applied to. Depending on the technologies used for the implementation of the data highway of the architecture, the prototype must be supplemented with procedures for automatic propagation of operations made by the tool in the metadata to the corresponding data sets. Wrappers that extract data from sources and collect metadata about source structure must be implemented and injected into the tool. Currently, there are wrappers for gathering metadata from relational, XML and unstructured text and PDF data sources.

For each of the automatically executable operation, we implemented a procedure that modifies the metadata and if necessary makes changes to the structure of data sets. We also implemented special functions that check each of the automatically evaluable conditions.

The following technologies were used for the prototype implementation:

– The metastore is implemented as a relational database Oracle in accordance with the designed metadata models.
– Two packages that perform metadata creation, change discovery and change treatment are implemented at the database using PL/SQL.
– The application contains a web interface with back-end implemented using PHP Laravel framework.
– For the operation of the application, Oracle database, web server and PHP engine are necessary.

4 Proof of Concept - Publication Data Warehouse

To perform an experimental approbation of the software prototype in order to validate the proposed approach to evolution handling, the developed solution was applied to the data warehouse that integrates data on research publications authored by the faculty and students of the University of Latvia. The system was appropriate for the approbation since it integrated data from multiple heterogeneous data sources and several changes occurred in its data sources and business requirements. The goal of the case study system is to integrate data about publications from multiple heterogeneous data sources and to provide these data for analysis in a data warehouse. The architecture of the developed system that includes data sources and data highway levels is shown in Fig. 4.

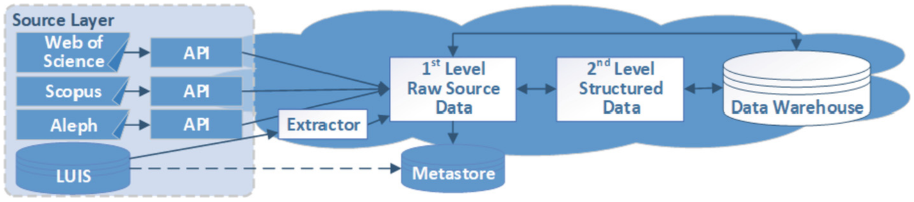


Fig. 4. Architecture of the publication data warehouse [18].

4.1 Data Sources

Data for the data warehouse are collected from one structured and three semi-structured data sources. None of the data sources contain information about absolutely all publications, so data at the sources are complementary, hence, they must be merged to obtain the most complete view on each publication.

LUIS is a university management system implemented in Oracle database. Along with information necessary for the provision of various university processes, LUIS stores data about publications entered by publication authors or administrative staff. We gather these publication data as well as data about authors and their affiliations from LUIS.

Aleph is an external library management system that stores data about books and other resources available at various libraries of educational institutions in Latvia. In addition to that, Aleph contains information on papers affiliated with the University of Latvia. Data in Aleph are entered by librarians. For the data warehouse, we gather bibliographic data about publications in XML format using a special API.

Scopus and Web of Science (WOS) are external indexation systems. Data from these systems are collected in XML format using API. We use four data set types from Scopus: publication bibliographic data, author data, affiliation data, and data about publication citation metrics. Only one data set type is available to the University of Latvia from WOS. It contains citation information, limited bibliographic information and author data (names, surnames and ResearcherID field).

4.2 Data Highway

The data highway of the publication system consists of three levels. First, data from data sources are ingested and loaded into the raw data level. We loaded data from the relational database LUIS into Hive tables using Scoop. For other sources, we first collected data files using API and saved them in Linux file system, then transferred these data to HDFS using a custom script.

In level 2 of the data highway, we transformed XML files into Hive tables. Data obtained from LUIS were not included in the 2nd level since they did not require additional transformation.

Finally, we implemented a data warehouse in Hive. Partially transformed data from external sources were integrated with LUIS data and loaded into the data warehouse. We also solved several data quality issues and performed elimination of duplicate publications in this process.

4.3 Metadata

Since we had access directly to LUIS data source, we embedded a procedure directly into the data source system that collects metadata about the structure and other metadata properties of tables used to populate the publications system. After source data in XML format are loaded from other data sources, we run another procedure that collects metadata about the structure of XML documents. This procedure was already available in LUIS, so we reused it.

Using the metadata management tool, we initially defined mappings between data items of the three data highway levels and metadata properties that were not discovered automatically.

4.4 Evolution

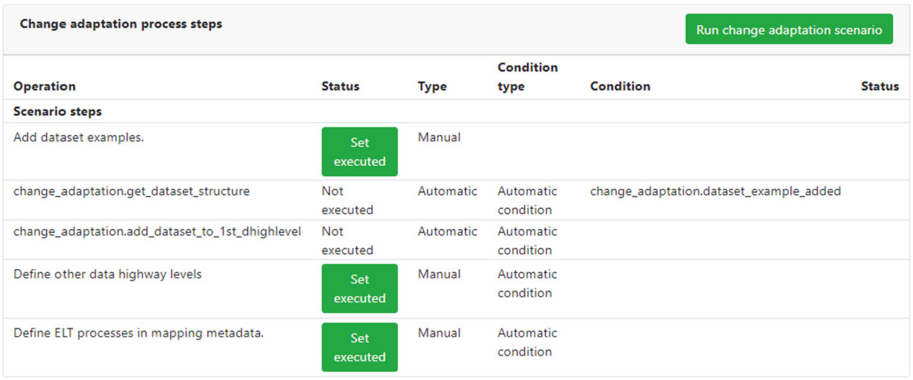
During operation of the publication system, several changes occurred in data sources, as well as in business requirements. Changes in data sources were discovered during the comparison of the metadata present in the metastore and structure and properties of the data incoming from the data sources. Changes in business requirements resulted in a manual modification of the system and metadata via the metadata management tool. Such real-world changes include an addition of new data items and removal of existing data items in data sources, addition of a new data source and change in a value of a data set property. In order to practically verify our solution, we also emulated test cases for other types of changes. Due to space limitations, in the following sub-sections we give only examples of several real-world changes to demonstrate how they were handled according to our approach.

Addition of a Data Source. In line with new requirements, the publication system was supplemented by a new data source DSpace that contained full text files of papers and metadata associated with them as tags. This change was implemented semi-automatically using the metadata management tool.

To create a new data source, the developer first manually entered data source name and description in the form via the metadata management tool. Then, a new record corresponding to the new source was created in the metadata table *Data Source* that implements the class *Data Source* from the schematic metadata model. Automatically, a linked record in the table *Change* that implements the class *Change* with the type *Addition* was created too.

There is only one adaptation scenario defined for the addition of a new data source. According to scenario definition, if a new data source is required for decision making, examples of data sets from the new source must be added manually so that the metadata collection procedures can generate the necessary metadata for the new source structure. The adaptation component must then create the data structures according to the data sets of the new source at the first level of the data highway. The developer must then define schemas of other new data highway levels, ETL processes, and create the corresponding metadata.

To handle the change, the developer initiated the change handling process in the metadata management tool. The change adaptation process steps given in Fig. 5 were created with the status *Not executed*.


Change adaptation process steps

Run change adaptation scenario

Operation	Status	Type	Condition type	Condition	Status
Scenario steps					
Add dataset examples.	Set executed	Manual			
change_adaptation.get_dataset_structure	Not executed	Automatic	Automatic condition	change_adaptation.dataset_example_added	
change_adaptation.add_dataset_to_1st_dhighlevel	Not executed	Automatic	Automatic condition		
Define other data highway levels	Set executed	Manual	Automatic condition		
Define ELT processes in mapping metadata.	Set executed	Manual	Automatic condition		

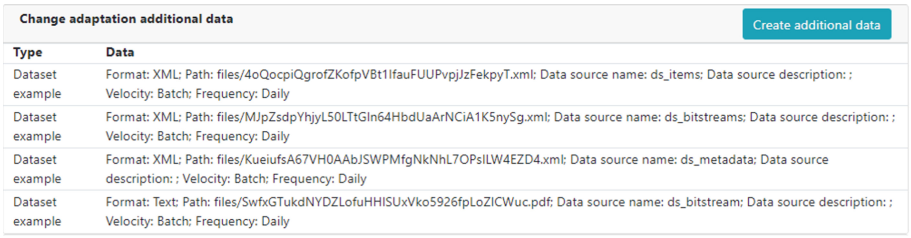
Fig. 5. Change adaptation scenario for the addition of a data source.

No manual conditions were created, but according to the definition of the scenario, the automatic condition *change_adaptation.dataset_example_added* must be checked after the execution of the operation *Add dataset examples*. This condition verifies that a developer added examples of data sets in a new data source. The status of the change was updated to *In progress*.

Since the first step of the scenario must have been performed manually, the developer added 4 data set examples used in the new data source. Three of the new data sets were XML files describing properties of papers and one data set was a PDF of a paper full text. The files were uploaded into the folder at the server, but the data about data set examples were saved in the metadata. The metadata stored for the data set examples are shown in Fig. 6. Then, the developer set the status of the first process step *Add dataset examples* as *Executed*.

Since there were no manual conditions that must have been checked or verified and the next 2 steps of the process are automatically executable, the developer launched change adaptation scenario via the metadata management tool. The tool executed 2 procedures corresponding to the automatic process steps.

The procedure *change_adaptation.get_dataset_structure* analyzed the structure of the example files and created metadata describing the structure and properties of 4 new data sets that correspond to the data set examples. These data sets were added to the new



Change adaptation additional data

Type	Data
Dataset example	Format: XML; Path: files/4oQocpiQgrofZKofpVBt1IfauFUUPvpjJzFekpyT.xml; Data source name: ds_items; Data source description: ; Velocity: Batch; Frequency: Daily
Dataset example	Format: XML; Path: files/MJpZsdpYhjyL50LTtGIn64HbdUaArNCiA1K5nySg.xml; Data source name: ds_bitstreams; Data source description: ; Velocity: Batch; Frequency: Daily
Dataset example	Format: XML; Path: files/KueiufsA67VH0AAbJSWPMfgNkNhL7OPsILW4EZD4.xml; Data source name: ds_metadata; Data source description: ; Velocity: Batch; Frequency: Daily
Dataset example	Format: Text; Path: files/SwfxGTukdNYDZLofuHHISUxVko5926fpLoZICWuc.pdf; Data source name: ds_bitstream; Data source description: ; Velocity: Batch; Frequency: Daily

Fig. 6. Data set examples for the addition of a data source.

data source. For PDF file, the procedure also created metadata properties: file extension (PDF) and file path at the server.

The procedure *change_adaptation.add_dataset_to_1st_dhighlevel* copied the metadata describing the structure and properties of 4 new data sets created in the previous step to the data highway level *Raw data level* and created mappings from data items in the data source to data items in the first data highway level.

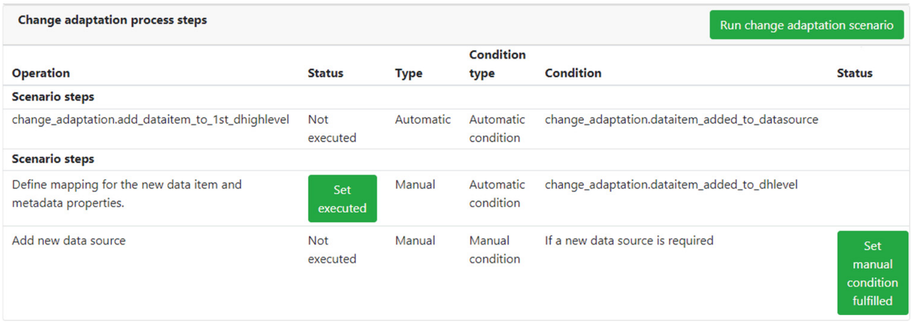
The last two steps of the change adaptation process (*Define other data highway levels* and *Define ETL processes in mapping metadata*) were executed manually by the developer using the metadata management tool. After the developer performed all necessary manual actions and set the aforementioned two steps as executed, he must have launched the change adaptation process again. Since there were no more adaptation process steps with the status *Not executed*, the change status was modified to *Processed*.

Addition of a Data Item. There were several changes of the type: data item addition in the case study system. Let us discuss one of them. A new XML element *citeScoreYearInfo* was added to the data set *SCOPUS_RA* that represents citation metrics obtained from the data source *SCOPUS*. It was composed of several sub-elements that were also absent in the previously gathered data sets.

The change was detected automatically by the change discovery algorithm that analyzed metadata describing the existing data set and compared it with the actual structure of the data set. The data set *SCOPUS_RA* of the data source *SCOPUS* was automatically supplemented with a new data item *citeScoreYearInfo* and its sub-elements in the schematic metadata. Since *SCOPUS_RA* is an XML data set, relationships of the type: composition were created for all parent and child data items that are descendants of the data item *citeScoreYearInfo*. Automatically, the record in the table *Change* with the type *Addition* was created too for the parent element citeScoreYearInfo, the change records were not created for sub-elements.

In general, if a data item has been added to an existing data set which is a part of a data highway level, mapping metadata and properties of the new data item are created. If the developer has made this change, he or she must specify a name, type, and (if applicable) data warehouse role of the new data item, the data set that contains the new data item, mapping metadata, and metadata properties. If an additional data source which was not previously used in the system is required for data loading, it must be added by implementing the change *Addition of a data source*. If a new data item has been added to a source data set, such change is processed automatically. Metadata describing the new data item is collected by the adaptation component, and the new data item is added to the data set at the first level of the data highway that corresponds to the source data set.

So, when the developer initiated the change handling process of the addition of the data item *citeScoreYearInfo*, the change adaptation process steps shown in Fig. 7 were created. As it is seen in the figure, there are 2 different scenarios for this change type. The choice of scenario in this case depends on two automatic conditions that check to which element a new data item was added (to data source or data highway level). One manual condition *If a new data source is required* was created and associated with the last scenario step *Add new data source*. The status of the change was updated to *In progress*.



Change adaptation process steps

Operation	Status	Type	Condition type	Condition	Status
Scenario steps					
change_adaptation.add_dataitem_to_1st_dhighlevel	Not executed	Automatic	Automatic condition	change_adaptation.dataitem_added_to_datasource	
Scenario steps					
Define mapping for the new data item and metadata properties.	Set executed	Manual	Automatic condition	change_adaptation.dataitem_added_to_dhlevel	
Add new data source	Not executed	Manual	Manual condition	If a new data source is required	Set manual condition fulfilled

Fig. 7. Change adaptation scenarios for the addition of a data item.

Since in this case, the new data item was added to a source dataset, the first scenario which was fully automatic was selected and the developer just launched the change adaptation process. The change handling mechanism executed the single step of the scenario *change_adaptation.add_dataitem_to_1st_dhighlevel* which added the new data item along with all its sub-elements to the data set at the first level of the data highway that corresponded to the source data set. This was done by

– copying the schematic and mapping metadata describing the new data item and its sub-elements to the data highway level *Raw Source Data*;
– creating a new relationship with the type *Composition* which specified that the new data item *citeScoreYearInfo* is a child of a data item *citeScoreYearInfoList* that was already present at the first data highway level;
– creating mappings from new data items in the data source to new data items in the first data highway level.

Since there were no more adaptation process steps with the status *Not executed*, the status of the change was set to *Processed*.

Deletion of a Data Item. An XML element *IPPList* was removed from the *Scopus metrics* obtained from *SCOPUS* data source. It was composed of several subelements that were also absent in the previously gathered data sets.

This change was detected automatically by the change discovery algorithm that analyzed metadata at the metastore describing the data set and compared it with the actual structure of the same data set. The missing element and its sub-elements were discovered in the data set *SCOPUS_RA*. In the metadata, the corresponding *Data Set* was marked as deleted and date and time of the change discovery was recorded. Automatically, the record in the table *Change* with the type *Deletion* was created for the parent element *IPPList*, but change records were not created for sub-elements.

In general, there are 3 possible scenarios for the propagation of the addition of a data item, two of them involve human participation and one is fully automatic. If the deleted data item belonged to a source data set, two adaptation scenarios can be applied:

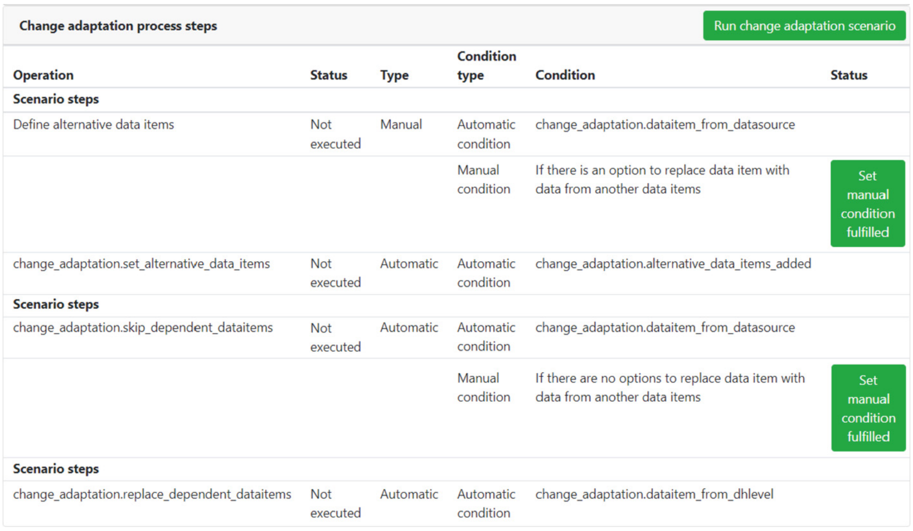
– *Replacement of a Deleted Data Item with Data from Other Sources or Data Sets*. In order to implement this adaptation scenario, the developer must provide additional information on an alternative data item or a formula that calculates the deleted data item from other data items. Since the alternative data item or other data items used in the formula might not be present in the system, it may be necessary to add metadata about the structure and properties of the new data items.
– *Data item skipping*. If the deleted data item can not be replaced by others or calculated by any formula, the adaptation component automatically determines data items of the data highway affected by the change and modifies ETL processes along with the mapping metadata to skip these affected data item.

If the developer has deleted a data item from a data set that is a part of a data highway level, the change is propagated automatically. Any other data items that have been obtained from the deleted data item are identified by analysing the mapping metadata. If such data items exist, the deleted data item is replaced in the mapping metadata and ETL processes. Such replacement is performed automatically if a data source from which the deleted data item was extracted is still available. If the data source is not available any more, the change must be processed by one of the above described scenarios.

In case of the deletion of a data item *IPPList*, the change adaptation processes demonstrated in Fig. 8 were created when the developer initiated change propagation in the metadata management tool. As it is shown in the figure, there are 3 different scenarios for this change type.

According to the definition of the scenarios, two automatic conditions must be checked before the execution of the first step of each scenario. These conditions determine whether the deleted data item was removed from an existing data set which is a part of a data source or a data highway level. Two manual conditions were also created. If a data item gets deleted from a data set belonging to a data source, a developer must evaluate whether there is an option to replace a deleted data item with data obtained from other existing data items and set the corresponding manual condition as fulfilled. This allows the change handling process execute the scenario unambiguously.

Since in this case, the data item *IPPList* was deleted from a source data set, the condition *change_adaptation.dataitem_from_datasource* returned true and only the first two scenarios could be executed. As it was not possible to substitute the deleted data item by another data item present in any of the data sources, the developer selected the second scenario and set the condition *If there are no options to replace data item with data from another data items* as fulfilled. After that, the developer just launched the change adaptation process because the only step of it was executable automatically.



Change adaptation process steps

Run change adaptation scenario

Operation	Status	Type	Condition type	Condition	Status
Scenario steps					
Define alternative data items	Not executed	Manual	Automatic condition	change_adaptation.dataitem_from_datasource	
			Manual condition	If there is an option to replace data item with data from another data items	Set manual condition fulfilled
change_adaptation.set_alternative_data_items	Not executed	Automatic	Automatic condition	change_adaptation.alternative_data_items_added	
Scenario steps					
change_adaptation.skip_dependent_dataitems	Not executed	Automatic	Automatic condition	change_adaptation.dataitem_from_datasource	
			Manual condition	If there are no options to replace data item with data from another data items	Set manual condition fulfilled
Scenario steps					
change_adaptation.replace_dependent_dataitems	Not executed	Automatic	Automatic condition	change_adaptation.dataitem_from_dhlevel	

Fig. 8. Change adaptation scenarios for the deletion of a data item.

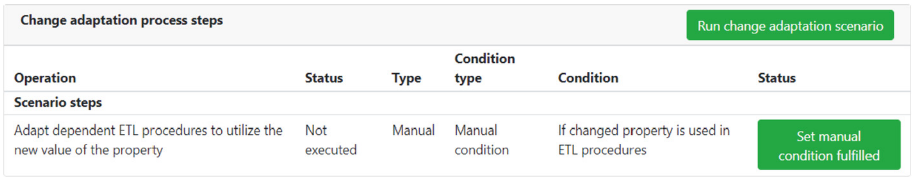
The change handling mechanism executed the step *change_adaptation. skip_dependent_dataitems*, which set all mappings that involve the deleted data item as well as depend on other data items that are obtained from the deleted data item as deleted. Then, the same actions were performed with mappings involving sub-elements of the deleted data item. As a result, the data items affected by the change would not be updated anymore.

Since the selected scenario did not include any other adaptation process steps, the change handling mechanism set change status to *Processed*.

Update of a Metadata Property Value. During the operation of the publication system, the API request used to obtain *Scopus metrics* was changed. The information about the API request was represented as a metadata property with the name *API request*. The change was discovered during the execution of the script that extracts data from the API since the script executed with errors. The change had to be processed manually since the new API request could not be discovered automatically. The developer updated the value of the metadata property using the metadata management tool. As a result, the record in the table *Change* with the type *Metadata value update* was created. The values of the property before and after modification as well as the name of the property were also added to the evolution metadata.

In general, the update of a metadata property value is an example of a change that is handled fully manually in the following way. If change in a value of a property has not been recorded as another change type (for example, as a change of data set format or data item type), it must be checked whether the changed property has been used in ETL procedures. In this case, all dependent ETL procedures must be adapted to utilize the new value of the property.

So, to process the change in the API request, the developer initiated the change propagation and one change adaptation process step shown in Fig. 9 was created. According to the definition of the scenario, one manual condition *If changed property is used in ETL procedures* was created and associated with the only scenario step. The status of the change was updated to *In progress*.


Change adaptation process steps
Run change adaptation scenario
Set manual condition fulfilled

Operation	Status	Type	Condition type	Condition	Status
Scenario steps					
Adapt dependent ETL procedures to utilize the new value of the property	Not executed	Manual	Manual condition	If changed property is used in ETL procedures	Set manual condition fulfilled

Fig. 9. Change adaptation scenario for the update of a metadata property.

Since the only adaptation scenario step must be executed manually and the condition that must be checked before the step execution is also manual, the scenario execution was performed by the developer. In this case, the scenario contained the instructions for the developer to be executed to process the change. When the developer updated the script used for data acquisition from the data source, set the condition and scenario step as executed, the change handling mechanism set change status to *Processed*.

5 Statistical Evaluation of the Proposed Approach

For the 20 atomic change types listed in Subsect. 3.2, we defined 34 different change adaptation scenarios. These scenarios were constructed from a total of 36 different operations and 46 conditions.

The distribution of scenarios by types (see Table 1) shows that 20% of scenarios are fully automatic and only 15% of all scenarios are fully manual. Even though the majority of scenarios still require human participation, in 85% of cases the proposed approach to evolution management reduces the manual work.

Table 1. Distribution of scenarios by type.

Type	Number	Percentage
Automatic	7	15%
Semi-automatic	22	20%
Manual	5	65%
Total	34	100%

Figure 10 demonstrates the distribution of individual conditions and operations used in scenarios by type. It can be observed that half of all operations that reflect steps of change adaptation scenarios are automatically executable. The same indicator for conditions is also close to the half (46%).

Figure 11 shows the proportions of automatic and manual operations and conditions within each change adaptation scenario. The total ratio of automatic and manual parts of all scenarios is 54 to 61, i.e. almost half (47%) of all scenario parts are executable automatically.

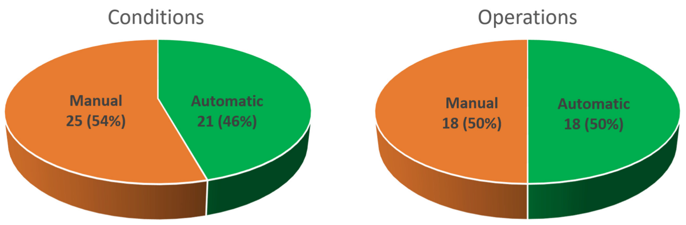


Fig. 10. Distribution of conditions and operations by type.

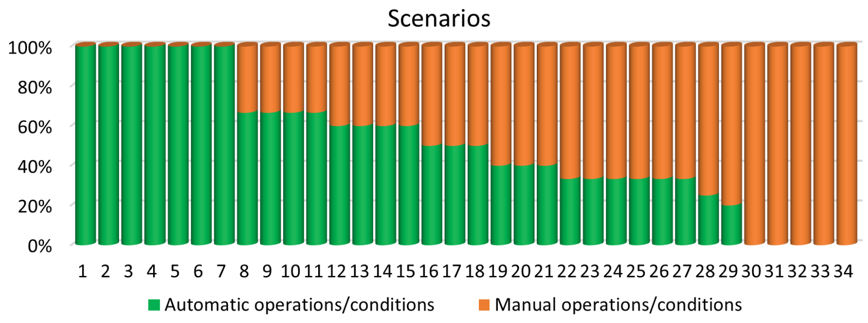


Fig. 11. Proportions of automatic operations/conditions.

In addition to real-world changes described in Sect. 4.4, we emulated test cases for every atomic change type based on the data sources and data highway of the case study system described in Sect. 4 to verify the completeness of our solution. For change types that can be propagated following multiple scenarios, we successfully tested all scenarios defined in our solution.

Since in our solution we mainly operate with metadata and examples of data sets that are not usually huge in terms of volume, we have not performed separate tests on the performance of the change propagation mechanism. The most data-intensive change adaptation operations in our solution are those that analyze the structure of example data sets. However, since example data sets are much smaller in volume comparing to data sets that are used for data loading, there are no performance issues even for the aforementioned operations.

6 Conclusions

In this paper, we presented a solution to problems caused by the evolution of heterogeneous data sources or information requirements of the data analysis system that utilizes data warehouse features for analysis support. The main results of our study include:

– A data warehouse architecture that allows to integrate data of various types and formats and analyze it using OLAP as well as other data analysis methods;

- A metadata repository that describes structure and other features of data sets involved in integration and analysis in a flexible way, as well as changes occurred in these data sets;
- A list of atomic changes that may occur in a data analysis system along with multiple automatic, semi-automatic and manual change adaptation scenarios for each change type;
- An adaptation metadata model for flexible storage of change adaptation scenarios that allows definition of new operations and conditions and construction of new scenarios out of them;
- An algorithm for automatic discovery of changes occurred in the system that saves information about detected evolution in the metadata repository;
- A mechanism for processing of discovered changes and changes performed manually that generates one or several change adaptation processes for each change according to change adaptation scenarios defined in the metadata and manages step-by-step execution of these processes;
- A prototype of a tool for the management of metadata and change handling that encompasses the implementation of the aforementioned algorithms.

There are several benefits of the proposed approach comparing to manual processing of changes in data sources and information requirements:

- Changes of certain types are discoverable automatically, which is faster than a human can detect them;
- Comprehensive information about changes occurred is available to the developer in one place;
- Management of evolution is ensured with less human participation;
- Change processing is transparent as all operations performed and conditions verified are available to the developer;
- The proposed approach is flexible and may be extended by defining additional operations and conditions in the corresponding metadata tables, then building new change adaptation scenarios from them and assigning these scenarios to change types.

The possible directions of future work include development of additional automatic scenarios for changes that currently require human involvement and preferences for change adaptation scenarios that can be set by a developer to promote automatic change handling. To implement this additional functionality, we are working on the metadata that would allow to save user preferences if evolution of certain elements in the system is probable in the future and a mechanism that would automatically propagate changes in the data highway. Since our architecture is built based on the data lake paradigm and source data are initially loaded in their original format, we can safely perform change propagation. In case if any change was processed incorrectly, the system can be recovered using the history of performed change adaptation operations available in the metastore.

References

1. Bentayeb, F., Favre, C., Boussaid, O.: A user-driven data warehouse evolution approach for concurrent personalized analysis needs. Integr. Comput.-Aided Eng. **15**(1), 21–36 (2008)

2. Wojciechowski, A.: ETL workflow reparation by means of case-based reasoning. Inf. Syst. Front. **20**, 21–43 (2018)
3. Ahmed, W., Zimányi, E., Wrembel, R.: A logical model for multiversion data warehouses. In: Bellatreche, L., Mohania, M.K. (eds.) DaWaK 2014. LNCS, vol. 8646, pp. 23–34. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10160-6_3
4. Golfarelli, M., Lechtenbörger, J., Rizzi, S., Vossen, G.: Schema versioning in data warehouses: enabling cross-version querying via schema augmentation. Data Knowl. Eng. **59**(2), 435–459 (2006)
5. Malinowski, E., Zimányi, E.: A conceptual model of temporal data warehouses and its transformation to the ER and object-relational models. Data Knowl. Eng. **64**(1), 101–133 (2008)
6. Thenmozhi, M., Vivekanandan, K.: An ontological approach to handle multidimensional schema evolution for data warehouse. Int. J. Database Manag. Syst. **6**(3), 33–52 (2014)
7. Thakur, G., Gosain, A.: DWEVOLVE: a requirement based framework for data warehouse evolution. ACM SIGSOFT Softw. Eng. Notes **36**(6), 1–8 (2011)
8. Kaisler, S., Armour, F., Espinosa, J.A., Money, W: Big data: issues and challenges moving forward. In: Proceedings of the 2013 46th Hawaii International Conference on System Sciences, HICSS 2013, pp. 995–1004. IEEE Computer Society (2013). https://doi.org/10.1109/HICSS.2013.645
9. Cuzzocrea, A., Bellatreche, L., Song, I.-Y.: Data warehousing and OLAP over big data: current challenges and future research directions. In: Proceedings of the Sixteenth International Workshop on Data Warehousing and OLAP (DOLAP 2013), San Francisco, California, USA, pp. 67–70 (2013)
10. Holubová, I., Klettke, M., Störl, U.: Evolution management of multi-model data. In: Gadepally, V., et al. (eds.) DMAH/Poly -2019. LNCS, vol. 11721, pp. 139–153. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33752-0_10
11. Solodovnikova, D., Niedrite, L.: Handling evolution in big data architectures. Balt. J. Mod. Comput. **8**(1), 21–47 (2020)
12. Sumbaly, R., Kreps, J., Shah, S.: The big data ecosystem at linkedin. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, pp. 1125–1134. ACM, New York (2013). https://doi.org/10.1145/2463676.2463707
13. Chen, S.: Cheetah: a high performance, custom data warehouse on top of MapReduce. VLDB Endow. **3**(2), 1459–1468 (2010)
14. Kimball, R., Ross, M.: The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, 3rd edn. Wiley, Hoboken (2013)
15. Nadal, S., Romero, O., Abelló, A., Vassiliadis, P., Vansummeren, S.: An integration-oriented ontology to govern evolution in Big Data ecosystems. In: Workshops of the EDBT/ICDT 2017 Joint Conference (2017)
16. Wang, Z., Zhou, L., Das, A., Dave, V., Jin, Z., Zou, J.: Survive the schema changes: integration of unmanaged data using deep learning. arXiv preprint arXiv:2010.07586 (2020)
17. Holubová, I., Vavrek, M., Scherzinger, S.: Evolution management in multi-model databases. Data Knowl. Eng. **136** (2021)
18. Solodovnikova, D., Niedrite, L., Niedritis, A.: On metadata support for integrating evolving heterogeneous data sources. In: Welzer, T., et al. (eds.) ADBIS 2019. CCIS, vol. 1064, pp. 378–390. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30278-8_38
19. Quix, C., Hai, R., Vatov, I.: Metadata extraction and management in data lakes with GEMMS. Complex Syst. Inform. Model. Q. **9**, 67–83 (2016)
20. Solodovnikova, D., Niedrite, L., Svilpe, L.: Managing evolution of heterogeneous data sources of a data warehouse. In: Proceedings of the 23rd International Conference on Enterprise Information Systems, ICEIS 2021, vol. 1, pp. 1–2. Online Streaming (2021)

21. Solodovnikova, D., Niedrite, L.: Towards a data warehouse architecture for managing big data evolution. In: Proceedings of the 7th International Conference on Data Science, Technology and Applications (DATA 2018), Porto, Portugal, pp. 63–70 (2018)
22. Solodovnikova, D., Niedrite, L.: Change discovery in heterogeneous data sources of a data warehouse. In: Robal, T., Haav, H.-M., Penjam, J., Matulevičius, R. (eds.) DB&IS 2020. CCIS, vol. 1243, pp. 23–37. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-57672-1_3