ASSIGNMENT OF BACHELOR’S THESIS

Title: Design and implementation of data flow analysis in the StreamSets platform for Manta project

Student: Marek Burdeň
Supervisor: Ing. Lucie Svitáková
Study Programme: Informatics
Study Branch: Web and Software Engineering
Department: Department of Software Engineering
Validity: Until the end of summer semester 2020/21

Instructions

The aim of this work is to design and implement a functional module prototype that performs syntactic and semantic analysis of topologies and pipelines in the StreamSets platform and then uses its result for data flow analysis. The module will be included in the Manta project.

1. Learn about the Manta project and the StreamSets platform.
2. Design a module in the Manta project for StreamSets topologies and pipelines. Use the existing project infrastructure.
3. Implement the prototype, document it properly and test it.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.  doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Head of Department  Dean

Prague November 15, 2019
Bachelor’s thesis

Design and Implementation of Data Flow Analysis in the StreamSets Platform for Manta Project

*Marek Burdel*’

Department of Software Engineering
Supervisor: Ing. Lucie Svitáková

June 4, 2020
Acknowledgements

First and foremost, I have to thank my supervisor Ing. Lucie Švitáková for her advices and pragmatism. I would also like to show gratitude to RNDr. Lukáš Hermann, Ing. Petr Košvanec and Manta colleagues for guidance during practical part of this work.
I hereby declare that I have authored this thesis independently, and that all sources used are declared in accordance with the “Metodický pokyn o etické přípravě vysokoškolských závěrečných prací”.

I acknowledge that my thesis (work) is subject to the rights and obligations arising from Act No. 121/2000 Coll., on Copyright and Rights Related to Copyright and on Amendments to Certain Laws (the Copyright Act), as amended, (hereinafter as the “Copyright Act”), in particular § 35, and § 60 of the Copyright Act governing the school work.

With respect to the computer programs that are part of my thesis (work) and with respect to all documentation related to the computer programs (“software”), in accordance with Article 2373 of the Act No. 89/2012 Coll., the Civil Code, I hereby grant a nonexclusive and irrevocable authorisation (license) to use this software, to any and all persons that wish to use the software. Such persons are entitled to use the software in any way without any limitations (including use for-profit purposes). This license is not limited in terms of time, location and quantity, is granted free of charge, and also covers the right to alter or modify the software, combine it with another work, and/or include the software in a collective work.

In Prague on June 4, 2020
Czech Technical University in Prague
Faculty of Information Technology
© 2020 Marek Burdeľ. All rights reserved.
This thesis is school work as defined by Copyright Act of the Czech Republic.
It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author’s permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis
Abstrakt

Cieľom tejto práce je preskúmať pohyby dát v prostredí StreamSets, navrhnuť a implementovať funkčný prototyp modulu, ktorý bude vykonávať dátovú analýzu nad StreamSets objektmi. Modul je spustitelný z projektu Manta, ktorý sa zameriava na poskytovanie analýzy dátových líníí pre široké spektrum technológií.

Výsledok dátovej analýzy je graf, ktorého uzly a hrany reprezentujú dátovú línii v prostredí StreamSets. V rámci práce je navrhnutý algoritmus na rozpoznávanie a rozširovanie elementárnych objektov z prostredia StreamSets pre dosiahnutie výsledného grafu. Graf je možné zobrazit pomocou Manta projektu.

Klúčové slová dáťová línia, správa údajov, StreamSets, Data Collector, Manta
Abstract

The purpose of this thesis is to research data movements in StreamSets platform, design, and implement a functional prototype module that will perform a data flow analysis for StreamSets objects. The module is executable from the Manta project, which focuses on providing data lineage analysis for a wide range of technologies.

The result of the data analysis is a graph whose nodes and edges represent the data lineage in the StreamSets environment. The work proposes an algorithm for recognizing and extending elementary objects from the StreamSets environment to achieve the resulting graph. The graph can be displayed using the Manta project.

**Keywords**  data lineage, data governance, StreamSets, Data Collector, Manta
# Contents

## Introduction

| Aim of the thesis | 2 |

## 1 Definition and Classification

| 1.1 Data Flow | 3 |
| 1.2 ETL | 4 |
| 1.3 StreamSets | 4 |
  | 1.3.1 Data Collector | 4 |
  | 1.3.2 Control Hub | 6 |
| 1.4 Manta | 8 |
  | 1.4.1 Graph | 8 |
  | 1.4.2 Manta Data Flow | 8 |

## 2 Analysis

| 2.1 StreamSets | 11 |
  | 2.1.1 Extraction | 11 |
  | 2.1.2 Pipeline | 12 |
  | 2.1.3 Stage | 12 |
  | 2.1.4 Field | 13 |
    | 2.1.4.1 JavaServer Pages 2.0 Expression Language | 14 |
  | 2.1.5 Summary | 14 |
| 2.2 Requirement Analysis | 15 |
  | 2.2.1 Functional Requirements | 15 |
  | 2.2.2 Non-functional Requirements | 16 |
| 2.3 Used Technologies | 16 |

## 3 Design

| 3.1 Connector Module | 20 |
  | 3.1.1 Extract Objects from Server | 20 |
List of Figures

1.1 Example of the Pipeline in the SDC. The figure was created in SDC [9] .................. 5
1.2 Example of topology in SCH [13]. The figure was created in SCH. .......................... 6
1.3 Example of the topology with pipeline view in SCH [13]. The figure was created in SCH. .......................... 7
1.4 Example of the data flow visualization with Manta Data Flow application [2] .................. 9
2.1 Example of a simple StreamSets data record in the stage from the pipeline’s preview. Figure was created in SDC [9] .................. 13
3.1 High level of the modules. Diagram was created in draw.io webpage [25] ............. 19
3.2 Diagram of the Extraction. Diagram was created with UMLet tool [26] ............. 20
3.3 Sequence Diagram of the Processing Pipeline – JSON File. Diagram was created with SequenceDiagram plugin for IntelliJ IDEA [27] ............. 21
3.4 Designed Model for the Topology and the Pipeline. Diagram was created with UMLet tool [26] ............. 22
3.5 Class diagram of general analyzers. Diagram was created in draw.io webpage [25] ............. 23
3.6 Class diagram of pipeline context. Diagram was created in draw.io webpage [25] ............. 24
3.7 Class Diagram of the Field. Diagram was created in draw.io webpage [25] ............. 26
3.8 Sequence Diagram of the Field Analysis. Diagram was created with SequenceDiagram plugin for IntelliJ IDEA [27] ............. 27
3.9 Sequence Diagram of the Field Expansion. Diagram was created with SequenceDiagram plugin for IntelliJ IDEA [27] ............. 28
3.10 Sequence Diagram of the Nodes and Flows Creation. Diagram was created with SequenceDiagram plugin for IntelliJ IDEA [27].

3.11 Example of the data flow visualization with Manta Data Flow application [2].
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>List of the supported stages – sources</td>
<td>39</td>
</tr>
<tr>
<td>4.2</td>
<td>List of the supported stages – destinations</td>
<td>39</td>
</tr>
<tr>
<td>4.3</td>
<td>List of the supported stages – processors</td>
<td>40</td>
</tr>
<tr>
<td>4.4</td>
<td>List of the supported stages – executors</td>
<td>40</td>
</tr>
</tbody>
</table>
Introduction

Organization’s data is a strategic asset. Just like finances and customer relationships, it needs proper management. When critical data is disorganized, organizations can face penalties for not complying with regulations, rising costs for storing and managing duplicate data, and other expenses. Moreover, they cannot be sure their business decisions are based on correct information. To minimize those risks, proper data governance is needed.

An ability to track and analyze all usage of the data helps to understand how the data is passing through various systems. The task is to handle data usage, understand from where and to where data transfer, capture where and how data change. A result of this task is provided by software that is gathering data information in a researched technology and provides a report with analyzed facts.

One of that software is developed by Manta. Manta is a company, which is specializing in providing a wide spectrum of the data analyses and connect them together for more than 30 technologies, in which an amount is still increasing. Bigger or smaller companies around the world use Manta data flow analysis. Manta supports data flow analyses for technologies that deal with modeling, data integration, programming languages, databases, reporting & analysis. Manta extends support for other technologies. A technology that is designated for data integration is a StreamSets Data Collector. The integration of the data flow analysis for that technology to the Manta project is a task of this work. It includes finding and providing the best possible solution for data analysis on the StreamSets platform.

This work is about creating a map that will help to find and track requested data and offer the data flow analysis from the beginning to the end on the StreamSets platform.

The work is divided into the following chapters:

1. **Definitions and Classifications** – In the first chapter is explained the most important terms to understand the whole concept for analytical
and practical parts of the work.

2. **Analysis** – In the second chapter is analyzed StreamSets objects at a detailed level. The chapter also contains requirements for a prototype with chosen technologies for implementation parts.

3. **Design** – In the third chapter is designed pre-analysis operations, and data flow analysis on focused objects.

4. **Prototype Implementation** – In the fourth chapter is described implementation of data flow analysis.

5. **Testing** – In the last chapter are tested expected functionalities, which are divided into individual parts of the prototype.

**Aim of the thesis**

The work aims to create a visualization of a data flow in the StreamSets platform for data analysts. The created visualization will be united with the Manta project.

The aim can be divided into the following objectives:

- Analyze the StreamSets platform – it means to understand how the data are manipulated in topologies and pipelines.
- Extract, resolve, and save objects into a model.
- Generate data flow for several stages in the pipeline as a graph for the visualization.
Chapter 1

Definition and Classification

This chapter explains the essential terms to understand the whole concept for analytical and practical parts of the work. It includes principles of program analysis, classification of StreamSets platform, and a concept of analysis’s conversion into a readable form. Program analysis is about data flow and programs that are built upon ETL (extract, transform, load) idea. Classification of StreamSets platform is about identification what kind of products StreamSets is offering and how data is passing there. At the end of this chapter is described the concept of data flow visualization in the Manta project.

1.1 Data Flow

A data-flow is a path for data to move from one part of the information system to another. A data-flow may represent a single data element such as the Customer ID, or it can represent a set of data elements (or a data structure).

In the data flow analysis, it is necessary to think of a program as a graph. The graph in this context is a set of nodes or points connected with edges or lanes. A program or an application may contain some elementary blocks (elements) or structures of these blocks (elements group). These elements have defined data lineage. Elements and elements groups represent the nodes in the graph, and the data lineage is defined with the edges.

Data flow analysis does not imply execution of the program being analyzed. Instead, the program is scanned in a systematic way and information about the use of variables is collected so that certain inferences can be made about the effect of these uses at other points of the program. Manta Data Flow solves a visualization of the collected information as a graph.

Understanding of the data flow in used applications may have a critical impact on accomplishing regulations, where data flow analysis provides an overview of total used data.
1. Definition and Classification

1.2 ETL

ETL is a type of data integration that refers to the three steps (extract, transform, load) used to blend data from multiple sources. It is often used to build a data warehouse. During this process, data is taken (extracted) from a source system, converted (transformed) into a format that can be analyzed, and stored (loaded) into a data warehouse or other system.

The data warehouse may be a database, file system, or apache server. ETL tools provide operations where the data is processed. The change of the data can be recognized and noticed for each operation. Based on that fact, a data flow-oriented analysis can determine the data flow for the researched ETL tool. Examples of ETL tools:

- Informatica PowerCenter,
- Talend Open Studio,
- IBM Information Server (Datastage),
- StreamSets Data Collector (SDC).

1.3 StreamSets

StreamSets is a company that specializes in automatic data processing and the data government. StreamSets’ graphical easily adjustable environment offers options to build data management application without understanding programming languages. StreamSets products are a StreamSets Data Collector (SDC), that expands the use of ETL, and a StreamSets Control Hub, that allows design, deploy and monitor objects from the SDC.

1.3.1 Data Collector

StreamSets Data Collector (SDC) is a StreamSets product. SDC is an open-source software where offers to pick any of the data sources to transfer data to another destination, where data will be stored. Meanwhile, there is an option to change data in many ways to fulfill the user’s expectations.

SDC approaches ETL differently, providing ETL upon ingesting services. Extracted data is at the beginning of the pipeline parsed into a StreamSets Data Collector Record. StreamSets use the SDC data format to transforming data in a pipeline. A transformation of the data provides the loading part of ETL into configured destination’s data format.

Data is passing through a set of stages that are extracting, transforming or loading data. This set of stages is called a pipeline. The stage is a part of the pipeline that performs a data operation.
A pipeline defines the flow of data from the source system to destination systems and defines how to transform the data along the way. User can use a single origin stage to represent the origin system, multiple processor stages to transform data, and multiple destination stages to represent destination systems.

Each stage has a specific job to do. There are 4 types of stages:

- An **origin stage** represents the source for the pipeline.
- A **processor stage** represents a type of data processing that you want to perform.
- A **destination stage** represents the target for a pipeline.
- An **executor stage** triggers a task when it receives an event.

Currently, SDC offers more than 250 stages of these four types. In this work are implemented only several stages. For the prototype are selected interesting stages to analyze, design and implement support of the data flow analysis. Supported stages are described in the **Data Flow Generator Module**.
1. Definition and Classification

1.3.2 Control Hub

Control Hub provides a cloud-native environment for disciplined life-cycle management of multi-pipeline data movement architectures. Service for better monitoring pipelines (designed in Data Collector) with Topologies, Jobs and Fragments.

StreamSets Control Hub (SCH) is a paid service. The service is an extension of the SDC with extended monitoring tools for pipelines.

Topology is an interactive end-to-end view of data as it traverses multiple pipelines that work together. You can map all data flow activities that serve the needs of one business function in a single topology.

In a figure is topology sample.

![Figure 1.2: Example of topology in SCH](image1.png)

Figure 1.2: Example of topology in SCH. The figure was created in SCH.

Job is wrapped pipeline with more information for monitor and control purposes in Control Hub. A fragment is a set of connected stages usually smaller than the pipeline. The fragment can be reused in different pipelines.

Because the topology is composed of pipelines, it is justified to pay the most attention to the pipeline for the data flow analysis in this work. In a figure is topology with pipeline overview.
Figure 1.3: Example of the topology with pipeline view in SCH [13]. The figure was created in SCH.
1. Definition and Classification

1.4 Manta

In this section is described what Manta generally is and Manta approaches to data flow analysis. Manta is specializing in the data flow analyses for the most used technologies in the world. Manta exploits supposition that various technologies are using databases or other systems that store data. These data repositories can be imagined as points that stand between different technologies. This information is used by Manta to create an overview of data flow analysis. Supported technologies are for example:

- Data Modeling tools as erwin Data Modeler, SAP PowerDesigner and ER/Studio.
- Various Databases as Teradata, Oracle, and PostgreSQL.
- Data Integration tools as IBM DataStage, Informatica, and Talend.
- Analysis & Reports tools as Power BI, Cognos, and Excel.

1.4.1 Graph

Manta approaches the visualization through an oriented graph. The graph serves as a diagram with relations between analyzed objects.

A single data element, a set of a data element, or a data structure are represented as nodes. The node may represent an operation that processes data and data elements as the set is assigned to this operation.

Manta graph format is adapted to display the data flow. So the data operations are provided by the node’s structure that encapsulates the data elements, and flow is presented as an edge. The edge is a connection (relation) between two nodes. A direction of the data flow is solved with associated directly to the edge. With this graph’s predispositions can be created detailed visualization of the data flow for a broad spectrum of technologies.

1.4.2 Manta Data Flow

The result of the analyzed technologies, where data is passing through various objects, is in the Manta graph format, and with the utilization of a Manta Data Flow is displayed. The visualization may be provided in detail through the oriented structured graph or at a high level as a simple list of used technologies with their objects. Analyzed technologies are connected through databases, file systems, or other services.

The visualization can be filtered by the user’s demands as hide unimportant objects, minimize displayed objects, etc. Users can also browse the created graph by a Find element operation in the Manta Data Flow application. In figure 1.4 is the visualization provided by the Manta Data flow application.
Figure 1.4: Example of the data flow visualization with Manta Data Flow application [2].
Chapter 2

Analysis

This chapter is about the understanding in detail how the data is passing in the pipeline, what are functional requirements (FRs) and non-functional requirements (NFRs), and what kind of technologies can be used to implement a scanner – a software, that performs the data flow analysis for the pipeline.

2.1 StreamSets

In this work is analyzed the StreamSets Data Collector (SDC) Version: 3.11.0, other versions that are greater than 3.0.0 should be compatible. The StreamSets analysis is about pipelines, stages and fields that are described below. Definitions of these objects are described in §1.3. This section also includes analyzed extraction options for these objects from SDC.

2.1.1 Extraction

Data flow analysis starts chronologically with the extraction, that guarantees to get information about the pipelines. The SDC offers two ways how to download created pipelines – manually or via a REST (Representational state transfer) API (Application programming interface). The SDC REST API offers all needed methods to extract the pipelines. The extracted pipeline is stored in a JSON file format. The JSON file contains all configured information about the pipeline.

List of GET methods that are needed for extraction:

- $\langle$scheme$\rangle:\langle$address$\rangle:$$\langle$port$\rangle$/rest/v1/pipelines
  (e.g., http://192.168.0.24:18630/rest/v1/pipelines)
  returns list of all pipelines.

- $\langle$scheme$\rangle:\langle$address$\rangle:$$\langle$port$\rangle$/rest/v1/pipeline/$\langle$pipelineId$\rangle$/export
  (e.g., http://192.168.0.24:18630/rest/v1/pipeline/TestId/export)
  returns pipeline’s configuration.
2. Analysis

2.1.2 Pipeline

*Pipeline is a representation of a stream of data that is processed by the Data Collector.*

For the data flow analysis, it is important to understand how the data are manipulated in the pipeline and the stages. The pipeline analysis is about the detailed examination of the elements (fields) and structures (stage) in the SDC. The following list consists of the relevant information about the pipeline from the JSON file:

- title – a representative name for the pipeline defined by the user,
- pipelineId – identifier,
- configuration – contains pipeline’s parameters defined by the user,
- list of the stages – a list of all data operations.

The JSON file includes more information about used libraries in the pipeline to process requested data, configuration about starting and stopping the pipeline, how an error should be handled when the pipeline is running and processed data is invalid, etc. The information about the pipeline that is unimportant for the data flow analysis is ignored in this work.

2.1.3 Stage

The stage represents a data operation. The following list consists of the relevant information about the stage as a JSON object from the pipeline’s list of the stages:

1. instanceName – unique name in pipeline.
2. stageName – specific name for stage.
3. configuration – a content depends on the stageName. Configuration contains list of JSON objects that each of them contains a "name" (string) and a "value" (content depends on "name"). For example a *Field Renamer Stage* configuration looks like:

   "stageName" : "com_streamsets_pipeline_stage_origin _salesforce_ForceDSource"

   "configuration" :

   - "name" : "forceConfig.soqlQuery", "value" : "soql_query"
   - "name" : "forceConfig.authEndpoint", "value" : "auth_endpoint"
   - "name" : "forceConfig.username", "value" : "username"
4. lanes – a lane is defined as a stream that is going from the source (stage) into multiple targets (stages).

- inputLanes – lanes that connects mentioned stage with previous stages. Data are transferred to mentioned stage.
- outputLanes – lanes that connects mentioned stage with subsequent stages. Data are transferred from mentioned stage.
- eventLanes – lanes that connects mentioned stages with subsequent stages. Only information about event (e.g., no more data event) is transferred from mentioned stage.

5. uiInfo – UI (User Interface) information about stage for visualization in SDC.

- label – a stage’s name defined by the user (custom name).
- stage type – origin/processor/destination/executor.

2.1.4 Field

A field is an elementary object in the pipeline. In the SDC extracted data from the origin stage is transformed into a StreamSets data record. In figure 2.1 is example of record. The record contains a list of the used element in a field path structure. Field’s name without single or double quotes can contains: 'a'..'z' | 'A'..'Z' | '0'..'9' | '_' | '*' | '[' | ']' | '(' | ')'. With the utilization of single or double quotes can be used unlisted chars to define the field. The field path structure defines the used field in the stage.

```
Record1: {MAP}
  name: {STRING} "126aa255-1665-4021-8545-t2fb086262d3"
  surname: {STRING} "c7386956-55bc-4afde63-3d5cc029c091"
  age: {INTEGER} -241487556
  manta: {STRING} "22b245f2-eddd-4b30-9167-cb971899aa6c"
```

Figure 2.1: Example of a simple StreamSets data record in the stage from the pipeline’s preview. Figure was created in SDC [9].

Examples of the field path structure:

```
/name
/address/city
/"first name"
```
2. Analysis

2.1.4.1 JavaServer Pages 2.0 Expression Language

JavaServer Pages 2.0 Expression Language is used in the SDC to create an expression that manipulates the data.

JavaServer Pages Expression Language (JSP EL) makes it possible to access application data stored in JavaBeans components easily. JSP EL allows the user to create expressions both (a) arithmetic and (b) logical. Within a JSP EL expression, the user can also use integers, floating-point numbers, strings, the built-in constants true and false for boolean values, and null.

The expression can contain constants, functions, fields’ paths, operators, runtime values, etc. The specific configuration may be expressed by global variables that are defined as the Runtime Values in the pipeline or in the SDC. The Runtime Values are called with the EL and divided into the following objects:

- Runtime parameters – defined in pipeline’s parameters. Example: `${JDBC_URL}`.
- Runtime properties – defined in sdc.properties or a separate properties file. Example: `${runtime:conf('OracleJDBCConnectionStr')}`.
- Runtime resources – defined file(s) in the SDC directory. A file must contain one piece of information to be used when the resource is called. Example: `${runtime:loadResource("JDBC.txt", true)}`.

Example of EL with the field path:

```java
${record:value('/age') +1}
${str:indexOf(record:value('/revenue'),'$')}
```

2.1.5 Summary

This part explains if the StreamSets Data Collector has a predisposition for the data flow analysis.

If it is possible to provide the best visualization of the data flow, all fields should be known. Exported pipelines are not containing a list of all fields that are used in each stage. The result may miss some fields, which were not specified in the whole pipeline, and there is no way how to detect them.

Options that can provide a list of all fields in each stage are:

- pipeline’s preview – unwilling changes or a data loss can occur, when preview uses the data from a configured external sources. For example, the preview runs a pipeline for ten records. When these records are processed with stages, the preview is displayed to the user. In this case, the data for these records are gained from some system, so the user can unwillingly start processing the data when he wants to make data flow analysis.
2.2. Requirement Analysis

- pipeline’s snapshot of the records – reachable only when the pipeline is running and the data is passing through the pipeline.

These options are extractable via StreamSets’ REST API, but both of them are not acceptable for the data flow analysis.

A process that comfortably provides a list of used fields is described in an algorithm below. The user in the pipeline defines field changes that are configured in the stages. These configured changes can be used to identify the list of the used fields in the pipeline. This approach requires to understand how data is changing in stages individually, especially what fields are passing through input and output lanes there. Each stage notifies the connected stages by the lanes about defined fields. The stages keep information about input and output collections of the fields. The data transferring must be restricted according to the data flow in the pipeline.

The pipeline JSON file may contain sufficient information about fields (elements) and stages (sets of elements) to create the data flow visualization. Otherwise, the analyzed pipeline is displayed without a piece of information about the used fields.

Option with field analysis and field expansion is chosen for the best possible result. Fields that are determined in the fields analysis phase are with deterministic rules used in other stages. Each stage does his directly assigned job in the SDC. So, in general, the job is to take data from input’s lane, process record of data (fields), and then put them into output lanes. So fields are only pushed through lanes and processed by stages. With this information, it is possible to assemble the data flow analysis.

2.2 Requirement Analysis

Software requirement is a functional or non-functional need to be implemented in the system. Functional means providing particular service to the user. Non-functional Requirements (NFRs) define system attributes such as security, reliability, performance, maintainability, scalability, and usability. They serve as constraints or restrictions on the design of the system across the different backlogs.

2.2.1 Functional Requirements

- F1: Module provides extraction of the pipeline from the SDC.

- F2: Module provides a result of the data flow analysis of the extracted pipelines in the graph.

- F3: Module provides validation of the customer’s license.
2. Analysis

2.2.2 Non-functional Requirements

- N1: Compatibility. The software should be executable from the Manta project.

- N2: Performance. Extraction, resolving and data flow generating should returns results in a reasonable time. It means that response time should not be longer than 5 seconds for one pipeline with minimum system requirements that are mentioned below.

  Specified System Requirements for Manta software [18]:
  - CPU: minimum 4 cores at 2,5 GHz, optimal 8 cores at 3 GHz
  - RAM: minimum 8 GB, optimal 16 GB
  - 5 GB free disk space recommended
  - Windows Vista/Server 2008 or newer, Linux or Solaris, Mac
  - Java: 1.7, 1.8 or 1.9

- N3: Documentation. The modules should be documented and readable by other co-workers.

- N4: Testability. Modules should be tested appropriately to reduce possible future errors.

- N5: Extensibility. Module’s architecture, design and implementation should be designed for future needs as the addition of new stages.

2.3 Used Technologies

In this section is a list of used technologies in the scanner for the data flow analysis for the StreamSets Data Collector. The selection of the used technologies is limited because in a Manta code structure is used technologies on previous decisions. To accomplish the integration with the Manta project is used the same technologies.

List of used technologies:

- Java – a powerful general-purpose programming language. It is used to develop desktop and mobile applications, big data processing, embedded systems, and so on. According to Oracle, the company that owns Java, Java runs on 3 billion devices worldwide, which makes Java one of the most popular programming languages [19].

- SVN – Subversion is an open-source centralized version control system. Subversion is also referred to as a software version and a revisioning control system. Developers use it to maintain current and historical versions of projects. [20]
2.3. Used Technologies

- **Maven** – **Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.** [21]

- **IDE** – Integrated Development Environment, enables programmers to consolidate the different aspects of writing a computer program. IDEs enhance programmer productivity by combining everyday activities of writing software into a single application: editing source code, building executables, and debugging. [22]

  This work is implemented with the utilization of IntelliJ IDEA – IDE by JetBrains. The environment was chosen based on previous experience.

- **JUnit version 4.13** – JUnit is an open-source Unit Testing Framework for JAVA. It is useful for Java Developers to write and run repeatable tests. [23]

- **Spring** – The Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications - on any kind of deployment platform. A key element of Spring is infrastructural support at the application level: Spring focuses on the "plumbing" of enterprise applications so that teams can focus on application-level business logic, without unnecessary ties to specific deployment environments. [24]

- **ANTLR 3** – ANother Tool for Language Recognition. This technology is used to perform syntactic and semantic analysis of the field path structure.

- **json-simple** - is a library from Google to process JSON files efficiently in Java. This library is used because of the simplicity of the API.

- **commons-el** – JSP 2.0 Expression Language library parses and evaluates the expression. This library performs syntactic and semantic analysis on the expression and returns created an abstract syntax tree (AST). The library belongs to Apache Commons project.
Chapter 3

Design

In the previous chapter are analyzed the extraction from the SDC, the pipeline, the stage, and the field. Based on this analysis is in this chapter designed processes to create the data flow analysis. A figure 3.1 lays out a composition of designed modules to fulfill the Aim of the thesis. All these modules are integrated with the Manta project.

Figure 3.1: High level of the modules. Diagram was created in draw.io webpage [25].
3. Design

3.1 Connector Module

The connection module provides pre-analysis’s operations. This module supports the extraction of the required pipelines and the pipelines import into a model.

3.1.1 Extract Objects from Server

The extraction of the pipelines solves StreamSets Connector Extractor Module. The module uses the REST (Representational state transfer) API (Application programming interface) accessible from the Data Collector.

Connection parameters to access the SDC REST API are:

- scheme,
- address,
- port,
- username,
- password.

To specify what objects should be extracted, the user has the option to select pipelines. The module loads selected pipelines and the connection parameters with Spring from Extensible Markup Language (XML) file(s). The user can define pipelines with a pipeline’s identification (ID) or with a label. The label may be considered as a pipeline group. The user can also exclude the pipelines by ID or label. Desired objects are extracted from the SDC and saved in the specified folder for the later data flow analysis. Figure 3.2 proposes how the pipeline(s) or the topology can be loaded from the SDC or the SCH.

![Figure 3.2: Diagram of the Extraction. Diagram was created with UMLet tool.](image)

If the user cannot connect to the SDC with Connector Extractor Module, StreamSets Data Collector offers to download objects manually. The manually downloaded pipeline should be stored in the defined file for the extraction.
3.1.2 Import Objects to Model

This part deals with the design of a model. The model stores all the needed information about the pipeline. The extracted pipeline is stored in a JSON file format. All usable JSON objects for visualization are processed by ParserService in a figure 3.3. The processed objects are saved in the Model (StreamSetsServer, Pipeline, Topology, Job, Fragment, Stage and Lane) in figure 3.4, which was designed on the basis of the previous analysis 1.3 and 2.1.

Figure 3.3: Sequence Diagram of the Processing Pipeline – JSON File. Diagram was created with SequenceDiagram plugin for IntelliJ IDEA 27.
3. Design

Figure 3.4: Designed Model for the Topology and the Pipeline. Diagram was created with UMLet tool [26].

3.2 Data Flow Generator Module

StreamSets Dataflow Generator Module analyzes imported objects that are acquired from StreamSets Connector Module and creates the Manta graph (see 1.4.1). The scope of the graph creation is focused on each stage individually. Dataflow Generator algorithm is designed to generate the data flow analysis. Dataflow Generator algorithm’s phases:

- Fields’ Analysis – Analysis of known field in each stage individually.
- Fields’ Expansion – Expansion of these known fields from each stage to others according to certain rules.
- Creation of nodes and flows – Transforming previous analysis into the graph.

Each object from figure 3.4 is analyzed by a single analyzer to separate logical steps during the data flow analysis. E.g., for the pipeline exists a PipelineAnalyzer that calls a StageAnalyzer for all stages. Because of the various stages’ configuration, each stage has its own analyzer that extends the StageAnalyzer. A figure 3.5 lays out general analyzers.

22
3.2. Data Flow Generator Module

Figure 3.5: Class diagram of general analyzers. Diagram was created in draw.io webpage [25].
3. Design

During the algorithm is stored analyzed information in a pipeline’s context – a figure 3.6. The context contains information that is reusable during the algorithm. Graph helper serves to perform graph’s operations.

The Field is the elementary object in the pipeline. The field path structure (field path) is stored for each stage in the pipeline context. Processing of the fields needs to allow easy manipulation with known values as add child to field, merge fields according to specific rules, create nodes, etc. For this purpose, the field is handled by own class Field to group these methods and store the field path.

The field path is parsed with an ANTLR 3 FieldParser to ensure reliable fields’ storage. The FieldParser is generated from a FieldLexer.g rules and a FieldParser.g rules by the ANTLR 3. Example of the ANLTER 3 grammars are simplified in the code below.
3.2. Data Flow Generator Module

```plaintext
WHITESPACE
   : (NEWLINE | ' ' | '\f' | '\t' | '\u00a0')+ { $channel = HIDDEN; }

SLASH : '/';

SINGLE_QUOTE : '"';

DOUBLE_QUOTES : '"';

IDENTIFIER : SINGLE_QUOTE (~ SINGLE_QUOTE | ESC_SINGLE_QUOTE)+ SINGLE_QUOTE
            | DOUBLE_QUOTES (~ DOUBLE_QUOTES | ESC_DOUBLE_QUOTES)+ DOUBLE_QUOTES
            | (CHARACTER | REGEX_CHARACTER | BACKSLASH)+;

fragment ESC_SINGLE_QUOTE : BACKSLASH SINGLE_QUOTE;
fragment ESC_DOUBLE_QUOTES : BACKSLASH DOUBLE_QUOTES;

fragment BACKSLASH : '\\';

fragment NEWLINE : CR? LF;
fragment CR : '\r';
fragment LF : '\n';

fragment UNDERSCORE : '_';

fragment CHARACTER : 'a'..'z' | 'A'..'Z' | '0'..'9' | UNDERSCORE;
fragment REGEX_CHARACTER : CHARACTER | '*' | '[ | ]' | '(' | ')';

tokens {
   AST_FIELD_PATH;
   AST_FIELD_NAME;
}

start_rule :
   field_path EOF       -> ^(AST_FIELD_PATH field_path)
   |
   field_path:
   (SLASH IDENTIFIER)+  -> ^(AST_FIELD_NAME IDENTIFIER+)
   |
   SLASH
```

25
3. Design

In a figure 3.7 is a class diagram for the field and field’s service, that uses the generated FieldParser by the ANTLR 3 to process field and helps to resolve all needed information from the Expression Language.

![Class Diagram of the Field](image)

Figure 3.7: Class Diagram of the Field. Diagram was created in draw.io webpage [25].

3.2.1 Field Analysis

During the field analysis phase, found fields are saved in the pipeline context for every stage separately. Figure 3.8 is a design for the field analysis. The PipelineAnalyzer launches the field analysis for each stage in the pipeline.

In this phase, each stage is individually analyzed. Analyzed fields that are set in stage’s configuration or gained with the utilization of the Manta’s libraries are stored for every stage separately in the pipeline context. Integra-
3.2. Data Flow Generator Module

tion with the Manta libraries helps to analyze other used technologies in the pipeline as SQL Statements for Oracle database, PostgreSQL, or Hive.

Figure 3.8: Sequence Diagram of the Field Analysis. Diagram was created with SequenceDiagram plugin for IntelliJ IDEA [27].

Configured fields can be expressed with JavaServer Page 2.0 Expression Language (EL) in the stages.

Common-el library (mentioned in Used Technologies) helps to analyze dependencies or evaluate/replace the Runtime Values that are set in external files.

3.2.2 Field Expansion

Each stage contains input (and output) field’s structure that is called a root field. So the stage has its own input (and output) root field that is accessible to other stages. The lanes, connections between the stages, are transferred into bindings for faster access during the Field Expansion. These bindings are stored for each stage individually in the pipeline context. If performed operation by stage is simple, input and output root fields are the same. Otherwise, stage is composited from a chain of root fields.

Event lanes are special lanes that don’t transfer data to the next stages. These lanes are considered to be not part of the graph. Connected stages only with event lane are omitted from the analysis.

In a figure 3.9 PipelineAnalyzer launches the field expansion.
3. Design

Figure 3.9: Sequence Diagram of the Field Expansion. Diagram was created with SequenceDiagram plugin for IntelliJ IDEA [27].

The field expansion’s algorithm for the stages consists of these steps:

1. Initiation of stages’ expansion information to new Map<Stage, Boolean> with false values. This expansion map stores information about expansion for each stage. In the map, a false value means that the associated stage needs to share familiar fields to the surroundings stages. A true value means that the associated stage has already informed all linked stages about used fields.

2. Start of the expansion from the random stage. The expansion for each stage means to:
   • Expand known fields to input stages. Input stages are defined in input bindings. The input stage provides its own output root field.
   • Expand known fields inside of the stage.
   • Expand known fields to output stages. Output stages are defined in output bindings. The output stage provides its own input root field.

The stage is expanded when all its known fields are shared to input, inner and output root fields according to expansion rules. Then the algorithm assigns a true value for the expanded stage in the expansion map. If one of the root field in the stage is extended by other stage’s expansion, for the changed stage is set the false value in the expansion map.

3. While all stages aren’t expanded – expansion map contains at least one false value – continue with the expansion (step 2) of the stages with false values in the expansion map.
3.2. Data Flow Generator Module

In addition, finding and merging fields is used case-insensitive comparison because of the best possible connection between databases’ columns (sometimes upper-case) and the configured fields (case-sensitive).

Field expansion rules may restrict the expansion between the stages. Field expansion rules can be affected by the field’s type and binding mode. The field’s type indicates in which direction the field should be expanded. Types of the field are:

- **I/O** – input/output. The field is passing through the input and output lanes. So all neighboring stages should know about this field.

- **I** – input (end). Input type indicates that the field data flow finished in the stage, so only the input lanes contain this field.

- **O** – output. The output type indicates that the field is known from the input lanes.

- **BEGIN** – opposite to input. Begin type indicates that the field data flow started in the stage, so only the output lanes contain this field.

The binding mode determines how the fields’ types is handled during the expansion (merging). The binding between stages contains binding mode and two connected fields.

The field that is found in the stage can be expanded to other stages (input/output) according to certain rules:

1. Created field in the stage cannot be expanded to the input stages nor to the input of the output’s stages – BEGIN field type.

2. Removed/Renamed (old version of field’s name) field in the stage cannot be expanded to output stages. The field flow ends in the stage - I field type.

3. The field that is found and wasn’t created nor removed/rename in the stage is expanded to all linked stages – I/O field type

Situations with fields’ types in the field expansion that can happen during default merging is written in the following list.

- **BEGIN → O.** The expansion of the BEGIN field’s type is only allowed to the output stages. The expanded field’s type is output. The expansion to the input stages is not allowed.

- **O → O.** The expansion of the O field’s type can only generate output field’s type. The expansion to the input stages is not allowed.

- **I/O ← I.** The expansion of the I field’s type is allowed to the input stages.
3. Design

- I/O ← I/O → I/O. The expansion of the I/O field’s type is allowed to the input and the output stages.

3.2.3 Creation of Nodes and Flows

After the field expansion phase, the data flow analysis is almost done. This phase guarantees a transformation of the previously analyzed pipeline and stages to required Manta’s graph’s format.

Before the creation of the nodes and flows is called Empty Stage Validation. The validation checks if any stage exists without at least one known field. When at least one stage without a known field is found, then an unknown field is created for each stage. In this case, used fields aren’t known. So the unknown field is created to inform about the impossibility to identify used fields in the pipeline. This information indicates to the user that he should not rely on the completeness of the generated data flow analysis.

List of the steps to complete the data flow algorithm:

1. Create nodes for the StreamSetsServer, the Topology, the Pipeline, the Fragment, and the Stage. For these objects, a hierarchical structure is created as a graph.

2. Create nodes for every known field in the stage.

3. Create flows between the fields in the stages.

In a figure, *PipelineAnalyzer* launches the Creation of Nodes and Flows phase.

![Sequence Diagram of the Nodes and Flows Creation](image)

Figure 3.10: Sequence Diagram of the Nodes and Flows Creation. Diagram was created with SequenceDiagram plugin for IntelliJ IDEA [27].

30
3.2.4 Summary

After the field analysis, various fields merging during the field expansion and creation of the nodes’ structure with clearly assigned relations between the fields, the result is integrated into the Manta Dataflow visualization. In figure 3.11 is part of the visualization of pipeline.

Figure 3.11: Example of the data flow visualization with Manta Data Flow application [2].
Prototype Implementation

Implementation of the required features can be divided into packages according to a MANTA code structure to accomplish compatibility and integrity. Two main packages are:

- **Connector Module** – provides pre-analysis operation.
  1. Connector Extractor Module – figure 3.2
  2. Connector Model Module – figure 3.4
  3. Connector Resolver Module – provides transformation from the pipeline JSON file into the Connector Model Module.

- **Dataflow Generator Module** – analyzes the data flow from the given objects and returns the Manta graph.

## 4.1 Connector Module

This module is designed to cover all pre-analysis operations as the extraction and the JSON file loading into the model.

### 4.1.1 Extract Objects from Server

The extraction of the pipelines is included in the Connector Extractor Module, which extracts required objects defined in the configuration via the SDC REST API (see figure 3.2).

The SDC REST API provides methods to get all the required information about the pipeline. Java HttpClient class allows users to request HTTP resources. Before the extraction part, the user configures the connection’s parameters (scheme, address, port, username, password) and set of the required pipelines for the data flow analysis. The pipeline can be defined with Id or a label. The label is a set of pipelines.
4. Prototype Implementation

If the extraction fails, the problem is logged. Extraction fails when:

- Username was not specified.
- Password was not specified.
- Failed to execute HTTP request to the StreamSets Data Collector.
- StreamSets Data Collector returned unexpected status.
- Response from the StreamSets Data Collector is not in the JSON file format.
- URL to the StreamSets Data Collector was not specified.
- URL to the StreamSets Data Collector was specified, but it is not a valid URL.

In the following code is a main extraction method of a class, that is integrated with Manta solution. Method runs the extraction queries and stores the result into specified directory.

```java
@Override
protected void extract() {
    prepareOutput();
    try {
        LOGGER.info("Extracting StreamSets objects to {}",
                   getOutputFile().getAbsolutePath());
        if (enabledControlHub) {
            LOGGER.error("Control Hub is not supported.");
        } else {
            open();
            checkCanExecute();
            Set<String> itemsToExtract = analyzeItemsToExtract();
            extractItems(itemsToExtract);
            close();
        }
    } catch (StreamSetsExtractorException e) {
        LOGGER.error("Problem occurred when extracting from StreamSets Server.", e);
    }
}
```
4.1. Import Objects to Model

After the extraction, the JSON files are stored and prepared to be processed. The processing requires resolving of the JSON file (see figure 3.3) and import of resolved objects into the model (see figure 3.4). This task is divided into two modules:

- **Connector Resolver Module** – module provides resolving of the JSON file and stores it into the model.
- **Connector Model Module** – module provides interfaces to all loaded objects from the extracted files.

The following method loads the StreamSets pipeline as a single JSON file or a directory. Pipelines in the directory are processed individually. The pipeline is transformed into JSON objects with the utilization of the simplejson library (see 2.3), and then the method imports it into the model.

```java
@Override
protected IStreamSetsServer readFile(File file) throws IOException {
    try {
        FileInputStream is = new FileInputStream(file.getAbsolutePath());
        InputStreamReader isr = new InputStreamReader(is, jsonEncoding));
        BufferedReader buffReader = new BufferedReader(isr);
        JSONObject jsonObject = (JSONObject) jsonParser.parse(buffReader);
        return parserService.processStreamSetsServer(jsonObject);
    } catch (IOException | ParseException e) {
        throw new IOException("Error encountered when parsing json export file: "+ file.getName(), e);
    }
}
```
4. Prototype Implementation

4.2 Data Flow Generator Module

Dataflow Generator module returns analyzed pipeline as the Manta graph. The Manta libraries provide execution of the data flow analysis. Class `StreamSetsDataflowTask` extends `AbstractGraphTask` and secures the integration with Manta via overridden method `doExecute`. This method is presented in the following code.

```java
@Override
protected void doExecute(IStreamSetsServer input, Graph outputGraph) {
    StreamSetsGraphHelper graphHelper = new StreamSetsGraphHelper(outputGraph, getScriptResource(), input);
    serverAnalyzer.analyze(input, graphHelper);
}
```

4.2.1 Data Flow Generator Algorithm

The algorithm analyzes the data flow of the given pipeline and creates the graph. `PipelineAnalyzer` (figure 3.5) includes the following method.

```java
@Override
public void analyze(IPipeline pipeline, StreamSetsGraphHelper gh) {
    PipelineContext ctx = new PipelineContext(pipeline, gh);
    init(pipeline, ctx);
    analyzeFields(pipeline, ctx);
    analyzeBindings(pipeline, ctx);
    expandFields(pipeline, ctx);
    createNodesAndFlows(pipeline, ctx);
}
```

**Initiation** of the pipeline’s context. The method initializes an expansion map that will be used in the field expansion and creates stages’ containers, where will be stored gained information during the data flow analysis for the intended stage.

**Field Analysis** – specified configuration parameters in the stages that can contain field are analyzed. During this process are sought fields’ dependencies too. The dependencies are saved in the pipeline context and will be processed during the flow creation. The prototype currently supports field analyses for 29 stages.

The field unambiguously determined with the field path. The field path can be located as a single subject or as a part of the Expression Language. The open-source common-el library performs syntactic and semantic analysis of the EL. A result is returned as the Abstract syntax tree (AST). The field path can be found in the AST. The field path is parsed with the FieldParser.
that is created by the ANTLR 3 with defined lexer and parser. All found dependencies and fields are saved in the pipeline’s context.

The following method invokes methods, which gradually trigger the field analysis method for each stage. It includes to determine the stage and assign the correct analyzer to it. If the stage is not supported, a default analyzer is assigned to it. After the end of this method, analyzed fields are stored in the pipeline context.

```java
@Override
public void analyzeFields(IPipelineConfig pipelineConfig, PipelineContext ctx) {
    for (IStage stage : pipelineConfig.getStages()) {
        getStageAnalyzer(stage.getStageName()).analyzeFields(stage, ctx);
    }
}
```

**Binding Analysis** – for the faster access to other stages are analyzed bindings between stages with a merging method for the field expansion. During this process are omitted the event lanes.

**Field Expansion** – starts when all fields are found and set in the pipeline context. The method expands analyzed fields to the stages until all stages are not fully expanded. After this method, all fields that are connected, fully merged according to the expansion rules. This description of the field expansion is implemented in the code below.

```java
@Override
public void expandFields(IPipelineConfig pipelineConfig, PipelineContext ctx) {
    boolean isExpanded = false;
    while (!isExpanded) {
        isExpanded = true;
        for (Map.Entry<IStage, Boolean> expansion : ctx.getExpansionMap().entrySet()) {
            if (!expansion.getValue()) {
                isExpanded = false;
                IStage stage = expansion.getKey();
                getStageAnalyzer(stage.getStageName()).expandFields(stage, ctx);
            }
        }
    }
}
```

The method below expands analyzed fields to all neighboring stages in Stage Analyzer.

```java
public void expandFields(T stage, PipelineContext ctx) {
    StageAnalyzerSetting setting = ctx.getSetting(stage);
```
4. Prototype Implementation

```java
Map<IStage, Boolean> expansionMap = ctx.getExpansionMap();

inputExpansion(expansionMap, setting);
if (otherExpansion(setting) || outputExpansion(expansionMap, setting)) {
    // If new field is found, we need to expand it to all stages. Otherwise input stages won't know about its.
    outputExpansion(expansionMap, setting);
    otherExpansion(setting);
    inputExpansion(expansionMap, setting);
}
expansionMap.put(stage, true);
```
### 4.2. Data Flow Generator Module

**Sources:**

<table>
<thead>
<tr>
<th>Stage name</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directory</td>
<td>✔</td>
</tr>
<tr>
<td>Hadoop FS Standalone</td>
<td>✔</td>
</tr>
<tr>
<td>JDBC Query Consumer</td>
<td>✔</td>
</tr>
<tr>
<td>Kafka Multitopic Consumer</td>
<td>△</td>
</tr>
</tbody>
</table>

MANTA doesn’t extract metadata from Kafka, so the analysis might be incomplete.

<table>
<thead>
<tr>
<th>Stage name</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salesforce</td>
<td>△</td>
</tr>
</tbody>
</table>

MANTA doesn’t support Salesforce Object Query Language (SOQL). Default SQL analysis is provided.

<table>
<thead>
<tr>
<th>Stage name</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle CDC Client</td>
<td>✔</td>
</tr>
<tr>
<td>PostgreSQL CDC Client</td>
<td>✔</td>
</tr>
</tbody>
</table>

**Destinations:**

<table>
<thead>
<tr>
<th>Stage name</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop FS</td>
<td>✔</td>
</tr>
<tr>
<td>Hive Metastore</td>
<td>✔</td>
</tr>
<tr>
<td>HTTP Client</td>
<td>✔</td>
</tr>
<tr>
<td>Kafka Producer</td>
<td>△</td>
</tr>
</tbody>
</table>

MANTA doesn’t extract metadata from Kafka, so the analysis might be incomplete.

<table>
<thead>
<tr>
<th>Stage name</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local FS</td>
<td>✔</td>
</tr>
<tr>
<td>Trash</td>
<td>✔</td>
</tr>
<tr>
<td>JDBC Producer</td>
<td>✔</td>
</tr>
</tbody>
</table>

**Table 4.1:** List of the supported stages – sources.

**Table 4.2:** List of the supported stages – destinations.
4. Prototype Implementation

<table>
<thead>
<tr>
<th>Processors:</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage name</td>
<td></td>
</tr>
<tr>
<td>Expression Evaluator</td>
<td>✓</td>
</tr>
<tr>
<td>Field Hasher</td>
<td>✓</td>
</tr>
<tr>
<td>Field Masker</td>
<td>✓</td>
</tr>
<tr>
<td>Field Order</td>
<td>✓</td>
</tr>
<tr>
<td>Field Remover</td>
<td>✓</td>
</tr>
<tr>
<td>Field Renamer</td>
<td>✓</td>
</tr>
<tr>
<td>Field Replacer</td>
<td>✓</td>
</tr>
<tr>
<td>Field Splitter</td>
<td>✓</td>
</tr>
<tr>
<td>Field Type Converter</td>
<td>✓</td>
</tr>
<tr>
<td>Hive Metadata</td>
<td>✓</td>
</tr>
<tr>
<td>Schema Generator</td>
<td>✓</td>
</tr>
<tr>
<td>Stream Selector</td>
<td>✓</td>
</tr>
<tr>
<td>Field Pivoter</td>
<td>✓</td>
</tr>
<tr>
<td>Data Parser</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 4.3: List of the supported stages – processors.

<table>
<thead>
<tr>
<th>Executors:</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage name</td>
<td></td>
</tr>
<tr>
<td>Shell</td>
<td>△</td>
</tr>
</tbody>
</table>

MANTA doesn’t support script analysis.

Table 4.4: List of the supported stages – executors.

The field analysis is unavailable for unsupported stages and default behavior for the field expansion is used. The default behavior provides fields passing through unsupported stages without restrictions.
This chapter is about tests for significant parts of the modules. The framework JUnit is used to test modules. Tests try to cover common but also extreme cases of used features to ensure future changes will be made effectively.

5.1 Connector Module

In the Connector Module is tested the extraction of the JSON files. Tests cover a connection with valid and invalid parameters, downloading of the selected pipelines, various options with pipelines’ identifiers and labels. Tests for the Connector Resolver Module and Connector Model Module are covered during stages’ analyzers testing.

5.1.1 Extraction

Extraction test case with a one pipeline identifier.

```java
@Test
public void test_ExtractPipelineById() {
    restExtractor.setIncludePipelines("ExpressionEvaluatorEx...");
    restExtractor.execute(null, null);

    Assert.assertEquals(1, getListOfFilesLength());
}
```

Extraction test case with an invalid pipeline identifier. The extraction for the invalid pipeline fails.

```java
@Test
public void test_ExtractUnknownPipelineById() {
    restExtractor.setIncludePipelines("unknown_pipeline_id");
    restExtractor.execute(null, null);

    Assert.assertEquals(0, getListOfFilesLength());
}
```
5. Testing

Extraction test case with 2 included pipelines that are excluded by identifier and label.

```java
@Test
public void test_ExtractPipelines_withExclude_00() {
    restExtractor.setIncludePipelines("TestRuntimeValuese5a61971-b66f-4459-9a55-7a1429556c1b," +
    "IncorrectFieldPathTest8dcd97da-77ba-4dc5-8861-8a3a06be7eaf");
    restExtractor.setExcludeLabels("Test");
    restExtractor.setExcludePipelines("IncorrectFieldPathTest8dcd97da-77ba-4dc5-8861-8a3a06be7eaf");
    restExtractor.execute(null, null);
    Assert.assertEquals(0, getListOfFilesLength());
}
```

5.2 Data Flow Generator Module

Tests in the Data Flow Generator Module cover the use of the resolving of dependencies from the expressions, the runtime values replacement, parsing of the field path structure, field’s operations (add, find, merge) and stages’ analyzers for the supported stages. These cases are also tested with complex pipelines with more than ten stages.

5.2.1 Expression Language

Expression Language may contain the field path and the SDC runtime values. A list of the resolved dependencies consists of all determined field paths that are used in the expression. The field path can be nested in functions.

```java
@Test
public void test_DependenciesResolverCorrectInput_InAnotherFunction_00() {
    String expressionString = "${str:concat(record:value('/hello'), 'world')};"
    List<Dependence> dependencies = expressionLanguageHelper.getDependenciesFromEL(expressionString);
    Assert.assertEquals("/hello", dependencies.get(0).getFlowPath());
    Assert.assertEquals(Edge.Type.DIRECT, dependencies.get(0).getFlowType());
}
```

Test for the runtime values replacement, where value of a variable is stored in external properties file.
5.2. Data Flow Generator Module

```java
@Test public void test_runtimeProperties_CorrectInput_00() {
    String runtimePropertiesExpression = "${runtime:conf('HDFSDirTemplate')}";
    String replacement = expressionLanguageHelper.replaceRuntimeValues(runtimePropertiesExpression, pipelineParameters, false);
    Assert.assertEquals("/HDFS/DirectoryTemplate", replacement);
}
```

5.2.2 Field Parser

Tests for the field path covers parsing paths with allowed characters. This operation is delegated to Field Parser that is generated from the defined parser’s grammar by the ANTLR 3 tool.

```java
@Test public void test_ParseFieldPath_SingleQuotes() {
    String fieldPath = "'/test1/test2/test3'";
    List<String> parsedFieldPath = parserService.parseFieldPath(fieldPath);
    Assert.assertEquals(4, parsedFieldPath.size());
    Assert.assertEquals("ROOT", parsedFieldPath.get(0));
    Assert.assertEquals("test1", parsedFieldPath.get(1));
    Assert.assertEquals("test2", parsedFieldPath.get(2));
    Assert.assertEquals("test3", parsedFieldPath.get(3));
}
```

5.2.3 Field

Tests for the field cover add, find and merge operations.

```java
@Test public void test_AddChildFieldToField_SimpleName_00() {
    Field addedField = testField.add("/test/field/name01", EFieldType.IO);
    Assert.assertEquals("name01", addedField.getName());
    Assert.assertEquals("/test/field/name01", addedField.getFieldPath());
    Assert.assertEquals(EFieldType.IO, addedField.getFieldType());
}
```

```java
@Test public void test_FindAddedFieldByPath_00() {
    testField.add("/test/field/name01", EFieldType.IO);
    Assert.assertNull(testField.findFieldByPath("/test/field/name00"));
    Assert.assertNotNull(testField.findFieldByPath("/test/field/name01"));
}
```

43
5. Testing

} }

5.2.4 Stage Analyzer

Tests for each supported stage cover configurable functionalities. A list of various pipelines, that are tested contains pipelines with all supported stages. A graph of the data flow analysis for the pipeline is compared with the previous verified version of the graph for the same pipeline. The graph is stored in a text file for straightforward comparison.

```java
@Test public void test_JDBCQueryConsumer_00() throws IOException {
  provideTest("./src/.../TestJDBCQueryConsumer_00.json",
              "./src/.../TestJDBCQueryConsumer_00_output.txt",
              "./src/.../expected_TestJDBCQueryConsumer_00_output.txt",
              "./src/.../graph_TestJDBCQueryConsumer_00_output.png",
              TestMode.COMPARE_EXPECTED_AND_OUTPUT);
}
```
Conclusion

The main aim of this work was to create the functional prototype of the module that helps to generate data flow visualization of the StreamSets topologies and pipelines for the Manta project. This goal was split into three smaller parts. Firstly, the analysis of the StreamSets products and its objects was needed to understand and design possible data flow visualization. Secondly, the extraction of the specified objects from the SDC and resolving the pipeline into the Java model were implemented. Lastly, it was needed to generate data flow for several stages as the graph visualization of these extracted objects. The implemented module is documented, tested and it is ready to be integrated into the Manta project. In this case, the main aim was fulfilled.

Currently created prototype module offers data flow analysis support only for tens stages. For the future, there is a possibility to increase the amount of the supported stages or support of the evaluation for extended regular expressions. The prototype module is planned to be released as the component of the Manta project.
Bibliography


<table>
<thead>
<tr>
<th>Number</th>
<th>Reference</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.</td>
<td>Build, Operate, Monitor Data Pipelines from Control Hub [online] [visited on 2020-05-03].</td>
<td><a href="https://streamsets.com/products/dataops-platform/control-hub/">https://streamsets.com/products/dataops-platform/control-hub/</a></td>
</tr>
<tr>
<td>15.</td>
<td>JSP - Expression Language (EL) [online] [visited on 2020-05-03]. Available from:</td>
<td><a href="https://www.tutorialspoint.com/jsp/jsp_expression_language.htm">https://www.tutorialspoint.com/jsp/jsp_expression_language.htm</a></td>
</tr>
<tr>
<td>21.</td>
<td>PORTER, Brett; ZYL, Jason van; LAMY, Olivier. Welcome to Apache Maven [online] [visited on 2020-06-03]. Available from:</td>
<td><a href="https://maven.apache.org/">https://maven.apache.org/</a></td>
</tr>
</tbody>
</table>
25. *diagrams.net - free flowchart maker and diagrams online* [online] [visited on 2020-06-04]. Available from: https://app.diagrams.net/.


Appendix

Acronyms

ANTLR  ANother Tool for Language Recognition
API    Application Programming Interface
AST    Abstract Syntax Tree
CDC    Change Data Capture
EL     Expression Language
ETL    Extract, Transform, Load
FR     Functional Requirement
FS     File System
HDFS   Hadoop Distributed File System
HTTPS  HyperText Transfer Protocol Secure
HTTP   HyperText Transfer Protocol
IDE    Integrated Development Environment
ID     Identity
JDBC   Java Database Connectivity
JSON   JavaScript Object Notation
JSP    JavaServer Pages
NFR    Non-functional Requirement
REST   Representational State Transfer
A. Acronyms

**SCH** StreamSets Control Hub

**SDC** StreamSets Data Collector

**SOQL** SalesforceObject Query Language

**SQL** Structured Query Language

**SVN** Apache Subversion

**UI** User Interface

**URL** Uniform Resource Locator

**XML** Extensible Markup Language
Appendix B

Contents of enclosed CD

- readme.txt.....................the file with CD contents description
- src..................................the directory of source codes
- implementation......................implementation sources
- thesis.........................the directory of \LaTeX\ source codes of the thesis
- BP_Burdel_Marek_2020.pdf ..............the thesis text in PDF format