UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE INFORMÁTICA

SANTIAGO - CHILE

Automation of retrieval, transformation and uploading of genomic data and their metadata for their integration into a GDM repository

JORGE IGNACIO VERA PENA

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE

INGENIERO CIVIL INFORMÁTICO

PROFESOR GUÍA: ANDRÉS MOREIRA

PROFESOR CORREFERENTE: MARCO MASSEROLI

OCTUBRE 2018

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE INFORMÁTICA

SANTIAGO - CHILE

Automation of retrieval, transformation and uploading of genomic data and their metadata for their integration into a GDM repository

JORGE IGNACIO VERA PENA

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE

INGENIERO CIVIL INFORMÁTICO

PROFESOR GUÍA: ANDRÉS MOREIRA

PROFESOR CORREFERENTE: MARCO MASSEROLI

OCTUBRE 2018

Acknowledgements

Thanks to all the GeCo team, in particular to Professor Masseroli for welcoming me into this challenging project with great and constructive critical view along the development of GMQLImporter.

Special thanks to Arif for guiding and helping me with the development of this work, your unconditional support, in the personal and in the technical; you helped me a lot. Sincerely it was a pleasure working with you and I hope to do so again in the future.

Gracias a toda mi familia por todo el apoyo y la oportunidad de embarcarme en esta aventura en Italia, esta experiencia concluye con la entrega de mi tesis; pero no solamente es eso, casi 3 años en los que pude descubrir más del mundo y pude también convertirme en una persona mucho más completa, con un mejor punto de vista ante los problemas que se vienen en adelante. Gracias por estar conmigo en todas y a pesar de haberme ido solo, nunca me sentí así.

Thanks to everyone I met and lived with in Italy and have become good Friends, living abroad has been an incredible experience and it is because of you.

Muchas gracias a todos!

Nacho 2017

Table of Contents

Abstract **1**

Abstract in Spanish **2**

Chapter 1. Introduction **3**

Chapter 2. Background Information **5**

2.1 Next generation sequencing (NGS) 5

2.2 Genomic Data Model (GDM) 6

2.3 Data sources 7

2.3.1 TCGA 7

2.3.2 TCGA2BED 8

2.3.3 ENCODE project 8

Chapter 3. State of the art **9**

3.1 GenoMetric Query Language (GMQL) 9

3.2 Genome Query Language (GQL) 10

3.3 Other solutions 10

3.4 Summary 10

Chapter 4. Thesis Goal **11**

Chapter 5. Software design **12**

5.1 Architecture design 12

5.1.1 Requirement analysis 12

5.1.2 Source and dataset analysis 14

5.1.3 General description for the 3 steps design 14

5.2 XML design 16

5.3 Database design 21

5.3.1 Sources, datasets and files 22

5.3.2 Statistics 22

5.3.3 Configuration 22

5.3.4 Database model 23

Chapter 6. Implementation 26

6.1 Downloader 26

6.2 Transformation 27

6.3 Loader 27

6.4 Fault tolerance 28

6.5 Logging 28

6.6 Technologies used 29

6.6.1 Scala 29

6.6.2 SLF4J + log4j 30

6.6.3 Maven 30

6.6.4 H2Database 31

Chapter 7. Evaluation 32

7.1 Configuration XML creation 32

7.2 Changes of implementation and adaptation 33

7.3 Parallel run 33

7.4 Results 33

Chapter 8. Conclusion and future work 37

References 38

Appendix A. Requirements Analysis for GMQLImporter 40

Appendix B. **Specifications for GMQLImporter** **49**
Appendix C. **ENCODE Metadata Explanation for GDM** **59**
Appendix D. **XSD schema for GMQLImporter configuration file** **63**
Appendix E. **Database design for GMQLImporter** **69**
Appendix F. **GMQLImporter console & file logging** **74**
Appendix G. **Creation of XML configuration file for GMQLImporter** **80**
Appendix H. **TCGA2BED datasets organization** **106**
Appendix I. **ENCODE metadata generation for experiment JSON** **109**
Appendix J. **Console manual for GMQLImporter** **135**
Appendix K. **Metadata replacement for ENCODE in GMQLImporter** **136**

List of figures

Figure 2:1 Cost per genome over the years by NIH research institute 5
Figure 5:1 initial domain model for GMQLImporter 13
Figure 5:2 root node of the GMQLImporter configuration XSD schema. 17
Figure 5:3 source node of the GMQLImporter configuration XSD schema. 18
Figure 5:4 dataset node of the GMQLImporter configuration XSD schema. 20
Figure 5:5 dataset node of the GMQLImporter configuration XSD schema. 21
Figure 5:6 database model for GMQLImporter. 23
Figure 5:7 file status usage diagram for GMQLImporter. 25

List of tables

Table 5:1 proposed initial entities for GMQLImporter development 13
Table 7:1 datasets from ENCODE and TCGA2BED to be imported. 33
Table 7:2 execution times in with and without parallelization for dataset. 35

Abstract

Due to NGS techniques, whole genome sequences are produced much cheaper and faster every year, thus genomic data is being gathered at a pace never seen before. By processing NGS data new sense making relationships between genomic regions are being found and fundamental biological questions are answered; therefore managing NGS data now seems to be the most important big data problem of humankind.

As the new NGS data generated are mostly heterogeneous, they are not easily interoperable. The Genomic Data Model (GDM) allows describing NGS data in a homogeneous way for their interoperation. GMQL is a next-generation query language that by means of using GDM data, gives genomics specific domain operations to biologists to process large volumes of data for discovering biological knowledge.

This thesis studies the improvement of NGS data analysis by automating and standardizing the genomic data and their experimental metadata integration into a GDM repository. The software developed is GMQLImporter; it extracts NGS data from multiple data providers, transforms the data according to GDM specifications and loads standardized GDM datasets into GMQL for further querying.

GMQLImporter was tested to download, transform and load into a GDM repository133,648 samples gathered from 2 different data providers and organized into 16 datasets. This work provides the capabilities to be easily extended to include samples from new data sources and in this way provide more NGS data to be queried and making new discoveries in bioinformatics.

Abstract in Spanish

A causa de las tecnologías NGS, las secuencias completas del genoma se producen cada vez más rápido y barato cada año, esto implica que la obtención de datos genómicos tiene un ritmo nunca antes visto. Procesando estos datos NGS se están descubriendo nuevas relaciones entre distintas regiones genómicas y se están encontrando respuestas a preguntas biológicas fundamentales. Por lo tanto parece que manejar los datos NGS ahora es el problema de big data más importante de la humanidad.

Dado que los nuevos datos NGS son mayormente heterogéneos, no son fácilmente interoperables. El Genomic Data Model (GDM) permite describir datos NGS y sus metadatos de manera homogénea para su interoperabilidad. GMQL es un next-generation query language que usando el modelo GDM, entrega a biólogos herramientas específicas del dominio genómico para procesar gran volumen de datos para así poder generar nuevo conocimiento biológico.

Este trabajo estudia la mejora del análisis de datos NGS mediante la estandarización y automatización de la integración de los datos experimentales y sus metadatos en un repositorio GDM. El software desarrollado es GMQLImporter, que extrae datos NGS de múltiples proveedores, los transforma de acuerdo a las especificaciones del GDM y carga los conjuntos de datos estandarizados en el sistema GMQL para posterior consulta.

GMQLImporter fue testeado para descargar, transformar y cargar en el repositorio de GMQL 133.648 muestras obtenidas de 2 proveedores de datos distintos y distribuidos en 16 conjuntos de datos. GMQLImporter permite ser extendido fácilmente para así incorporar muestras de nuevas fuentes de datos y de este modo disponer mayor cantidad de datos NGS para interrogar y generar nuevos conocimientos en la bioinformática

Chapter 1. Introduction

The rise of NGS technologies, thanks to their high throughput, is producing more genomic data than ever before, entire genomes can be sequenced within one day and several consortia are publishing this experimental data to be publicly available. By querying NGS data, fundamental biological questions are now being found; therefore, managing NGS data now seems to be the most important big data problem for humankind.

The bioinformatics have been mostly challenged by the *primary analysis*, the production of sequences from the DNA segments or *reads* and the *secondary analysis*, alignment of reads to a reference genome and search for specific features, such as variants/mutations and peaks of expression. The most important problem that has been occurring is called *tertiary analysis*, which is used to find sense making relationships to explain how different regions in the genome interact with each other. This newly generated NGS data is mostly heterogeneous and data collected in different sources cannot be easily interoperated. Thanks to Genomic Data Model (GDM), the data can be described in a homogeneous way so the NGS data can be interoperated with different formats. GDM defines the notion of datasets and samples, where in each dataset, the genomic data files have to follow a defined schema and their clinical information have to be in specified metadata files where the general non-genomic features of the sample are explained.

GMQL is a next-generation query language made for querying NGS data that is improving the interaction between biologists and genomic data by offering genomics domain specific operations capable of discovering new biological knowledge by processing huge volumes of NGS data. GMQL requires minimum informatics knowledge and by using a high level language that mixes MapReduce with SQL like characteristics can query NGS data homogeneously by using the GDM model.

The need of standardization of NGS data to GDM format is important to be able to easily integrate the different types of these data and process them through GMQL.In this thesis

GMQLImporter, a general Extraction, Transform and Load (ETL) tool, is presented to answer this need; it allows obtaining NGS data from different genomic databanks, and delivering them into standardized GDM files for further GMQL querying. It is sufficiently generic and allows further extensions as easy addition of new manageable types of data formats or databanks. The sources and datasets nowadays present in the GDM repository of the GMQL system are used for testing the designed and developed tool, which allows running either sequential or parallel executions with multithreading.

The whole GMQL project and therefore GMQLImporter module is developed under Apache License 2.0, which allow the freedom to use the software for any purpose, distribute it, modify it and to distribute modified versions of the software, without any royalties. All the projects developed by the GeCo team at Politecnico di Milano can be found on GitHub https://github.com/DEIB-GECO.

Chapter 2. Background Information

2.1 Next generation sequencing (NGS)

Next generation sequencing (NGS) is a revolutionary research technology for DNA sequencing. With NGS complete human genome sequencing can take less than one day, in comparison with the previous sequencing technology Sanger, which needed almost 10 years to sequence a final version for a human genome. Also as shown in the Figure 2:1 the cost per genome has substantially decreased over time, and the improvement in NGS technologies allows sequencing a whole genome for less than $1,000 U.S.

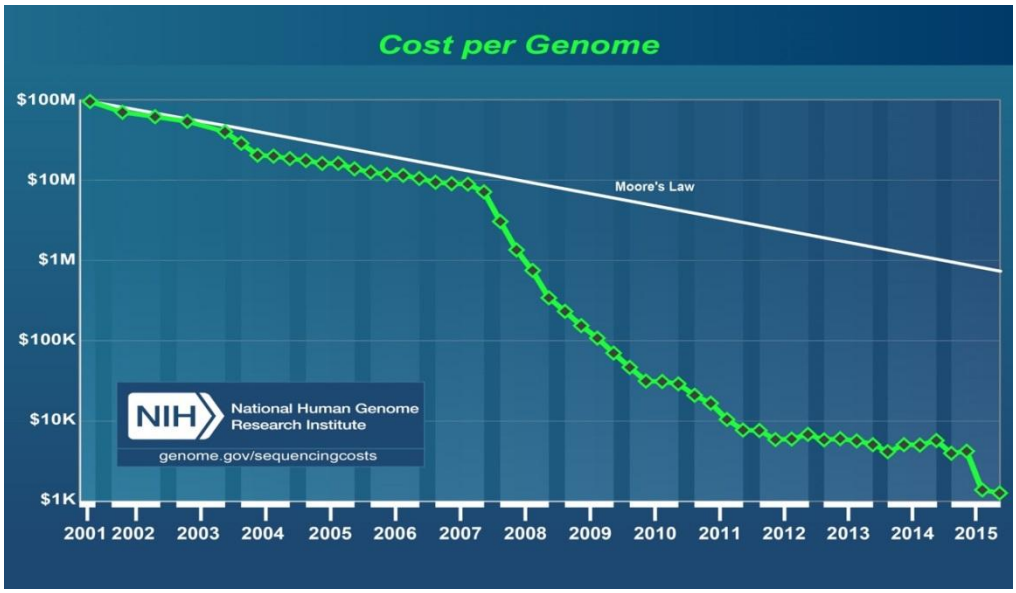


Figure 2:1 Cost per genome over the years by NIH research institute

NGS technologies are increasing the rate of finding answers for fundamental biological and clinical research questions, giving the possibility for comprehensive characterization of genomic and epigenomic landscapes. Questions as how DNA-protein interactions and chromatin structure affect gene activity, how does cancer develop, how genomic or epigenomic traits affect complex diseases such as cancer or diabetes. NGS catalyzes the arrival of personalized and precision medicine by finding sense-making relationships between different

biological samples on the same regions based on their reference genome. NGS are used to collect genomic and epigenomic features such as DNA mutations or variations (DNA-seq), transcriptome profiles (RNA-seq), DNA methylations (BS-seq), DNA-protein interactions and chromatin characterizations (ChIP-seq and DNase-seq). Research centers collect the very numerous genomic features produced by NGS raw data processing done by large-scale sequencing projects. Some research centers belong to world-wide consortia such as ENCODE, TCGA, 1000 Genomes Project, Roadmap Epigenomics, and others [1].

Due to the many different laboratories and techniques used in NGS, several formats for the storage of the results exist and cause the data to be highly heterogeneous. Multiple data formats are available for describing NGS experiments such as BED, NARROWPEAK, VCF, SAM, etc [2].

2.2 Genomic Data Model (GDM)

NGS technologies are creating huge amounts of epigenomic and genomic data of different types, and these data are used in tertiary analysis; tertiary analysis focuses in finding meaningful relations on how different regions interact with each other. A new paradigm is proposed for tertiary analysis, and is based in the Genomic Data Model (GDM) [2]. GDM allows describing in a homogeneous way, data that is semantically heterogeneous, in addition enables the interoperability for the data by setting a schema to describe heterogeneous genomic regions data and the incorporation of clinical and experimental metadata.

GDM defines the notion of dataset and samples, and for each sample, the abstraction of genomic region data and metadata. The region data provides representation of DNA and its genomic features, while the metadata describes general properties of the sample.

In GDM a sample is modeled by 3 attributes (id, R and M). Where id is an integer used as identifier of the sample, R is a set of genomic regions for the sample and M is a set of metadata for the sample. A genomic region (r) is a portion of the genome well defined and identified by 4 values called region coordinates (chr, left, right and strand). chr represents the

chromosome where the region belongs, left and right are the starting and ending position inside the chromosome, finally strand indicates the direction of reading for the region encoded as '+' (positive), '-' (negative) or '*' (missing: not assigned to a specific strand). The set R is built by pairs (c, f) where c represents the 4 coordinates of the region and f represents the region features as typed attributes, typed attributes can be any of boolean, char, string, integer, long or double and is assumed they have arbitrary names. The schema for the sample region contains a list of attribute names and types for the identifier, coordinates and features. M represents the metadata of the sample region, composed by attribute-value pairs (a, v) assuming values to be strings, attribute names are not necessarily unique as they can be multi-valued by appearing multiple times.

A GDM dataset, in short, is a group of samples sharing a common region schema, including features of the same types; every sample has its unique id inside a dataset and a corresponding metadata with experimental conditions.

2.3 Data sources

Multiple NGS data providers exist, and the main ones used in GMQL are TCGA2BED and ENCODE. Both sources have their data in GDM format and each one have multiple datasets with different schemas between them, in this section TCGA, TCGA2BED and ENCODE project are explained:

2.3.1 TCGA

The Cancer Genome Atlas (TCGA) [3] is a collaboration project between National Cancer Institute and National Human Genome Research Institute; they produce NGS data and have made available publically genomic changes in 33 types of cancer. Data provided by TCGA have contributed for many cancer studies. By creating a genomic data analysis pipeline, TCGA analyzes human tissues on a very large scale to collect and select genomic alterations.

2.3.2 TCGA2BED

TCGA to BED format (TCGA2BED) [4] is a software tool for extraction, extension and integration of TCGA genomic data and its clinical metadata. The process of enrichment of the TCGA data includes the addition of annotations from important genomic repositories such as Entrez Gene, HGNC, UCSC and miRBase. The output files are transformed into BED files for the region data and tab-delimited for its metadata. BED format is column-based and contains one line per gene and is GDM friendly also. An open access FTP repository is kept updated with the TCGA2BED generated files.

2.3.3 ENCODE project

Encyclopedia of DNA Elements (ENCODE) [5] is a consortium founded by National Human Genome Research Institute (NHGRI) and receives collaboration of multiple international research groups. Its goal is to generate a comprehensive list of functional human genome elements. A functional element is a segment of the genome that encodes a defined product. ENCODE has performed a large number of NGS studies to map functional elements across the human genome and they provide all their data and protocol descriptions publicly available via their website.

Chapter 3. State of the art

3.1 GenoMetric Query Language (GMQL)

As NGS data availability is increasing, the ability to process and gather information from these data is necessary: GenoMetric Query Language (GMQL) [6], a next-generation query language, is created to improve the interaction between biologists and NGS data. By offering bioinformatics domain-specific operations, it allows scientists to discover knowledge processing great volume of samples together, including the analysis of their region data and their experimental metadata.

GMQL deals with distances inside the genome: every sample is aligned to a reference genome, and samples with the same reference genome can be processed using these distances in different arithmetic operations between them. GMQL provides a simple yet powerful high level language that combines procedural MapReduce with SQL inspired capabilities with low requirements in informatics knowledge. GMQL changes the paradigm of how NGS data is being managed [7] by providing standard unary operations of SELECT, ORDER, AGGREGATE, PROJECT and MERGE; binary operations of UNION and DIFFERENCE. An on the other hand it provides domain specific operations as: COVER, JOIN and MAP; these operations have a biological interpretation. This paradigm abstraction is proven useful using several biological query examples.

GMQL enables biologists to easily query NGS genomic region data and their metadata, being a pioneer in including metadata in the computation process, and moreover, supports metadata management. Thus metadata is involved in the selection and matching and also is carried along the process, making it available to access it after the execution of the query. Actually GMQL includes available experimental data from ENCODE and TCGA2BED provided in GDM friendly format and therefore ready to be used for querying.

Being a state of the art tool, GMQL is highly scalable and portable due to the dominant cloud computing paradigms in which is inspired and therefore is well supported in cloud

execution. This allows millions of genomic samples to be queried at once with enough abstraction of the language to perform high level processing in a compact way.

3.2 Genome Query Language (GQL)

Genome Query Language (GQL) [7] analyzes NGS data using an SQL extension, they implement their tool by directly connecting it to the sequence reads from a NGS machine, therefore highly specific development is done to operate this kind of data, the processing of raw data is powerful in the sense that the data is non-generalized thus it can be queried with high precision at the cost of identifying the genomic regions of interest. Nowadays the most common way to solve this identification problem is done by developing ad hoc specific tools to fit every type of analysis separately.

3.3 Other solutions

Other tools as The Genome Analysis Toolkit [8], provides a programming framework for analyzing NGS data in SAM format. BEDOPS [9] provides high-performance genomic feature operations focused in BED format, they improve the performance of the tool by file compression. SAMtools [10] provide universal utilities for processing read alignments in Sequence Alignment/Map (SAM) format. Another genomic toolkit is BEDTools [11] allows comparing large datasets in multiple formats, and provides also tools for genomic features manipulation operations.

3.4 Summary

Among the available tools for genomic research, GMQL allows easier usage by the users, it encapsulates multiple processes commonly done in the field and in comparison with other tools where data selection, annotation retrieval and data interoperation have to be done manually; GMQL is robust enough to be simple to use, allows also the dataset selection over their experimental metadata. GMQL is also independent tool, does not need external scripting and thanks to these features and allows the integration for biologists with no advanced computer programming skills to research in the field of genomics.

Chapter 4. Thesis Goal

This thesis focuses on the improvement of NGS data analysis by standardizing the acquisition and integration of genomic region data and their clinical metadata into a GDM repository. Specifically in the GMQL project, the need of standardization and integration of NGS data to GDM friendly format is very important for further development and testing. The solution proposed is GMQLImporter, a software that allows acquiring NGS data including their metadata from different genomic databanks and deliver them into standardized GDM datasets ready to use for GMQL querying; it is sufficiently generic to import the nowadays available information in the GDM repository of the GMQL system and allows further extensions as easy additions of managed new data formats or databanks. The designed and implemented software allows incrementally gathering new datasets and iteratively integrating more metadata for better GMQL metadata based querying. For testing the implementation of GMQLImporter, the system will update the datasets already manually loaded in GMQL for the sources of ENCODE and TCGA2BED, where multiple dataset types are found with different downloading procedures and data formats.

Chapter 5. Software design

In this chapter the general software architecture and workflow are described, database design for storage of the historical progress of the importing process and the design for the configuration of the GMQLImporter tool for importing new genomic data.

5.1 Architecture design

This thesis requires developing an automatic system to build and keep updated a big integrative repository of genomic data and their metadata publicly available from several specific sources, transforming it into GDM format to be seamlessly used for genomic data processing through GMQL. The main sources of datasets to be imported are the Encyclopedia of DNA Elements (ENCODE) and The Cancer Genome Data into bed format (TCGA2BED). The module has to allow easy addition of future data sources to be imported. This section describes the business rules for the project, also description for key concepts used in GMQLImporter.

5.1.1 Requirement analysis

As pointed in appendix A "Requirements Analysis for GMQLImporter" the aim is to make easy integrating heterogeneous repository from publicly available genomic data and performing transformation to GDM if needed and standardization for the experimental metadata. As extension for GMQL, GMQLImporter is also developed in Scala programming language. NGS data and their metadata are provided by *sources* and kept organized in *datasets* with a variety of formats, to load the datasets inside GMQL, they have to be in a compatible GDM format, have a valid schema for the region data files it contains and one file of metadata for every file of region data. GMQLImporter has to receive the list of sources and their datasets to be imported into GMQLRepository. During the requirement analysis, the initial domain model is collected:

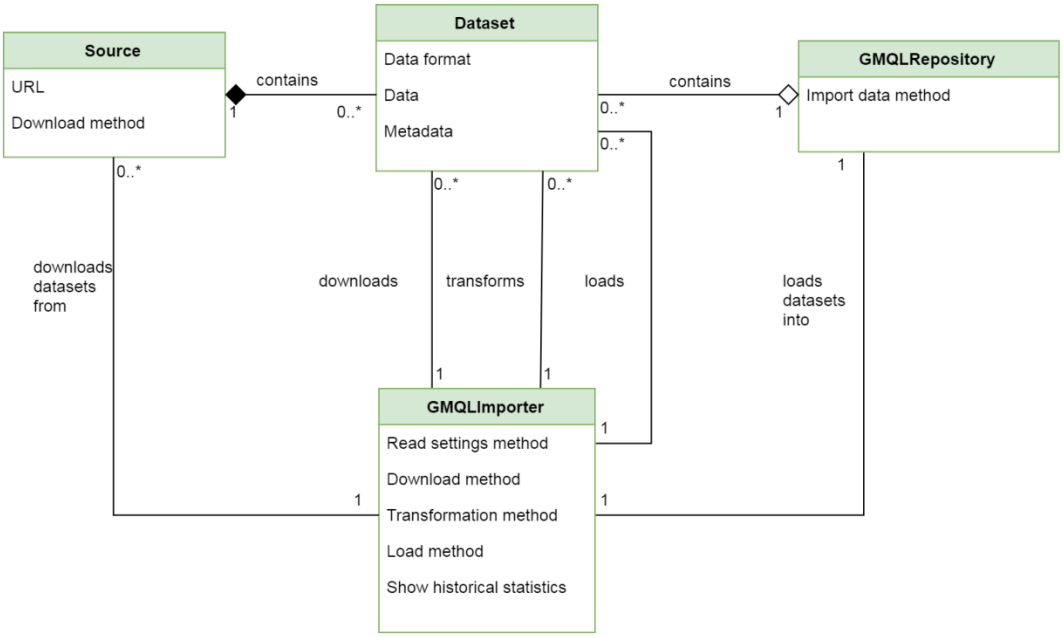


Figure 5:1initial domain model for GMQLImporter

Entity	Description
Source	NGS region data and metadata provider.
Dataset	Collection of region data and metadata, where all samples respect a given region data schema and file format.
GMQLRepository	Interface to connect and import datasets inside GMQL.
GMQLImporter	Downloads multiple datasets from different sources. Transforms downloaded Datasets into GMQL compatible format (GDM) with standardized metadata. Loads GDM compatible format datasets into the GMQL repository. Tracks historically the status of downloaded and transformed files.

Table 5:1proposed initial entities for GMQLImporter development

5.1.2 Source and dataset analysis

For the correct abstraction of the data importing process is needed to specify what is a source and a dataset in the GMQLImporter context, the full explanation is in the appendix B"Specifications for GMQLImporter" and its main specifications are now explained.

A source is considered to be a provider of NGS data, it could be potentially anyone, from ENCODE, TCGA, TCGA2BED, etc. The sources publishes genomic data with experimental metadata, they usually have their data accessible through a unique platform, as ENCODE provides in their website or TCGA2BED in their FTP server. The region data with its metadata is organized into datasets regardless their original format; the same dataset must be always together, from origin to delivery at GMQLRepository. This datasets have to be treated equally along the processing cycle. The general attributes for sources are their name, location and the way data is organized inside their side. For datasets, the important is to know their name, region and metadata files format, and the schema of their region data files.

TCGA2BED and ENCODE provide data in GDM friendly formats, and some of those datasets are used in the development of this thesis. They differ in the downloading process and the metadata gathering. The complete generalization of the sources and dataset performed in this thesis is later explained in the XML design section.

5.1.3 General description for the 3 steps design

For the correct and easy to understand the process of importing the heterogeneous data, a 3 step general procedure is defined. Inspired in Extraction, Transform and Load (ETL) paradigm the first step is named download and manages the connection and retrieval of datasets from a source, the second step is called transformation and handles, if needed, the conversion into GDM format and then, the modification of metadata to allow its interoperability with datasets from other sources. The final and third step is called loading where the transformed datasets in GDM format are loaded into the destination GDM repository. A full step by step detailed procedure is as follows:

5.1.3.1 Download

In this step the connection to the source's server is carried out, allowing the download of the files that compose the desired dataset. The download has to ensure only needed files are downloaded, in short, not to download the same file 2 times when updating the local copy of the data and to know which files are outdated or the ones that are no longer used in the source's datasets. For this management, the source should provide ideally with file size, file last modification date and the file's hash but even with one of those attributes the updating process could be done.

The hash is the most accurate attribute to know if the local copy is a reflect image of the source's, then it's the first priority to check, then the last modification date allow to know if the server side file has been updated but does not allow to certainly know if the downloading of the file is correct. The final attribute to check is the file size, by assuming the source only modifies the files to make them better or more updated, the file size can tell if the file has changed as to have 2 files with the exact size after modification is hard but not impossible, therefore this attribute constitutes the 3rd priority of checking as neither provides sufficient information to check if the file is correctly downloaded or if the version in the server is newer or not. There are multiple protocols that allow file transferring and many ways to distribute data on the server side so non-universal approach for downloading all sources is possible, worth notice that download procedure depends more in the source than the dataset itself, therefore different download methods can be visualized for different sources.

5.1.3.2 Transform

Transformation in this context means to modify the data to be queried seamlessly not caring about the source's differences, an important step for the correct transformation is done at metadata level, where the metadata names have to be standardized by modifying them if needed. Following this logic, the metadata from different sources could be used equally when the real attribute represented by it is the same or can be related. GMQLImporter gives the general tools for the correct transformation of the region data and their metadata. First ensures the genomic data and their metadata are in a GDM compatible format therefore

performs if needed modification of the original sources' files into GDM format, this operations could include full translation from raw data to GDM or extracting compressed containers to name some, but could potentially be any data modification or translation. Once the NGS data and metadata are GDM friendly the transformation process checks for the region data to fulfill the dataset schema and the metadata to be revised and modified if needed for importing it. In the specific case of TCGA2BED they provide the region data and metadata in perfectly compatible GDM format because their project is based on that. For the ENCODE side, they provide region data in GDM friendly format compressed, therefore extraction is needed. The metadata from ENCODE project is provided in several ways, as seen in appendix C "ENCODE Metadata Explanation for GDM" and an special operation has to be done to transform it into GDM. The transformation process as the files provided by the source could be potentially in any format, cannot be universal therefore a generalization is needed, notice that same source could provide different file formats as ENCODE, causing that process to potentially be specific for every different kind of region data and metadata obtained. This generic approach gives GMQLImporter to be potentially capable of managing any kind of genomic data.

5.1.3.3 Load

As the thesis goal is to import all the processed data into a GDM repository, the final step of this process is to load the data into it. Without performing this loading step, GMQLImporter could be an independent tool for gathering GDM data, but it's particular use as part of the GMQL project needs to connect with the GMQLRepository, the interface provided by GMQL for dataset importing, to load and make available for GMQL querying the integrated datasets. Potentially could connect to any GDM repository for loading the datasets, therefore the loading step could be specialized to load into another repository.

5.2 XML design

GMQLImporter is designed to receive a configuration XML file with the needed parameters to perform the downloading, transforming and loading steps of the datasets, to provide a

general approach for the formal creation of this file; an XSD schema file is designed to validate any configuration XML file given as input for the tool. The schema comprehends a root node where general settings and a source list are stored. Sources as seen before, represent NGS data providers which provide those genomic data and experimental metadata divided in datasets, each source contains a list of datasets, each dataset after processing, represents a GDM dataset where every sample has a region data file and a metadata and every sample share the same region data schema. The configuration XSD shown in appendix D "XSD schema for GMQLImporter configuration file", the file is organized in a tree structure starting from the root node, passing through the sources and ending in the datasets as shown below:

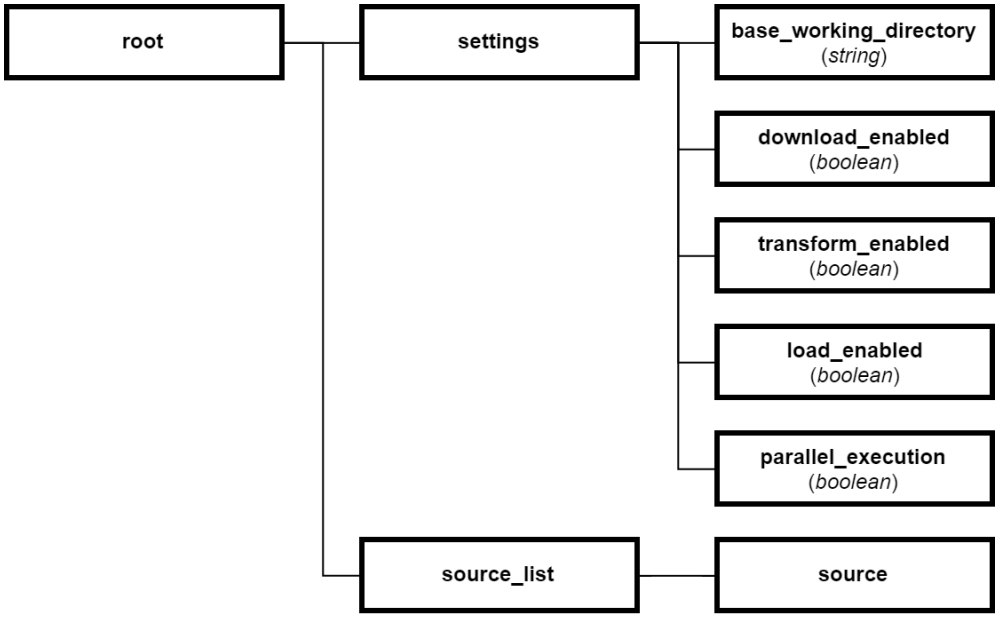


Figure 5:2root node of the GMQLImporter configuration XSD schema.

- **root**: contains general settings and a list for sources to import.
    - **settings**: general settings for the program execution.
        - **base_working_directory**: folder where the importer will use during execution.

- **download_enabled**: indicates if download process will be executed.
- **transform_enabled**: indicates if transformation process will be executed.
- **load_enabled**: indicates if loading process will be executed.
- **parallel_execution**: indicates if the whole execution is run in single thread processing or multi-thread processing.

- **source_list**: collection of sources to be imported.

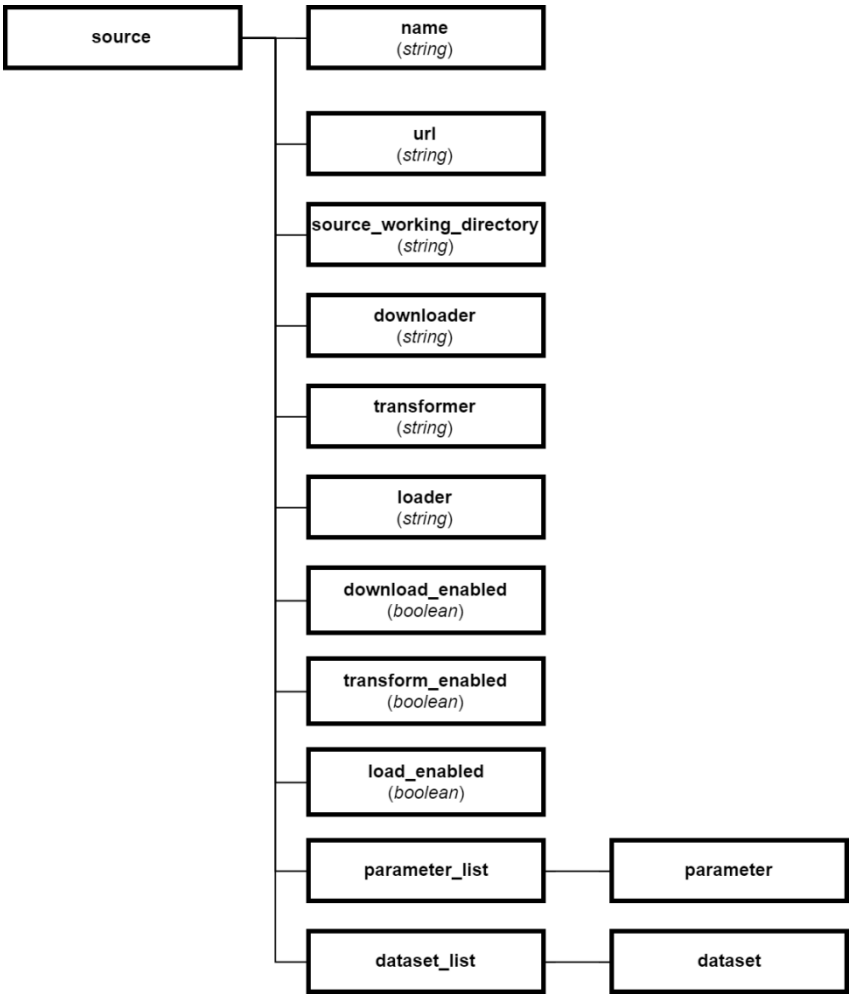


Figure 5:3 source node of the GMQLImporter configuration XSD schema.

- **source**: represents an NGS databank, contains basic information for downloading, transforming and loading process.
    - **name**: identification for the source.
    - **url**: address of the source.
    - **source_working_directory**: sub directory where the source's files will be processed.
    - **downloader**: indicates the downloading process to be performed to download the samples from this source.
    - **transformer**: indicates the transformation process to be performed to change the source samples into GDM compatible files for interoperability.
    - **loader**: indicates the responsible for loading the processed data into a GDM repository.
    - **download_enabled**: indicates if this source is going to be downloaded from the source.
    - **transform_enabled**: indicates if transformation process is executed for this source.
    - **load_enabled**: indicates if loading into GDM repository is executed for this source.
    - **parameter_list**: collection of parameters for downloading or loading the source.
- **dataset_list**: collection of datasets to import from the source.

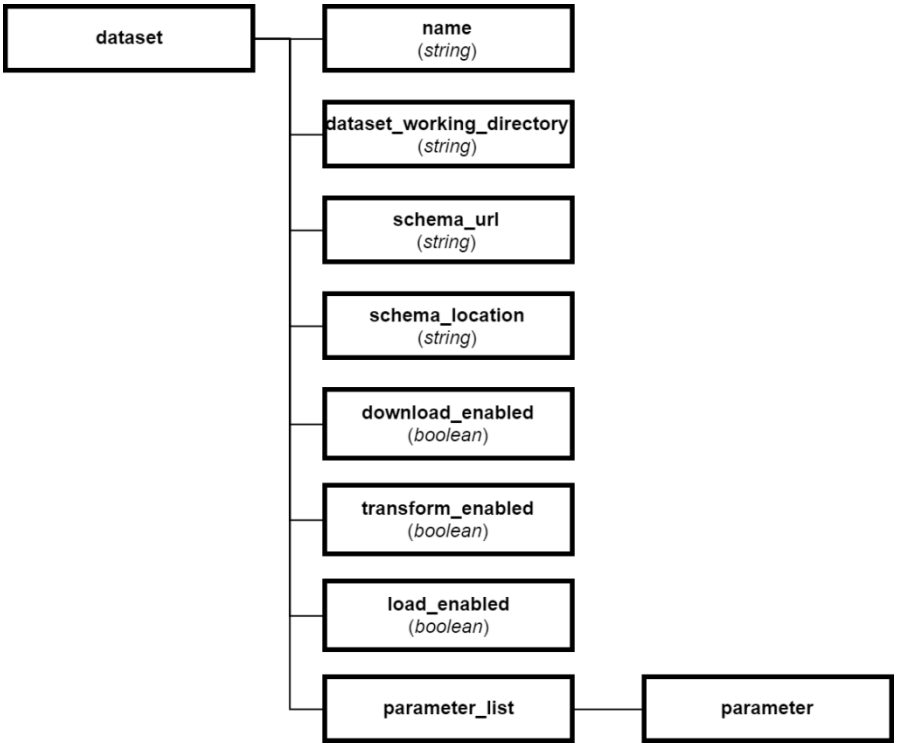


Figure 5:4 dataset node of the GMQLImporter configuration XSD schema.

- **dataset**: represents a set of samples that share the same region data schema and the same types of experimental or clinical metadata.
    - **name**: identifier for the dataset.
    - **dataset_working_directory**: subfolder where the download and transformation of this dataset is performed.
    - **schema_url**: address where the schema file can be found.
    - **schema_location**: indicates whether the schema is located in FTP, HTTP or LOCAL destination.
    - **download_enabled**: indicates if the download process will be performed for this dataset.

- **transform_enabled**: indicates if the transformation process will be performed for this dataset.
- **load_enabled**: indicates if the loading process will be executed for this dataset.
- **parameter_list**: list of dataset specific parameters for downloading, transforming or loading this dataset.

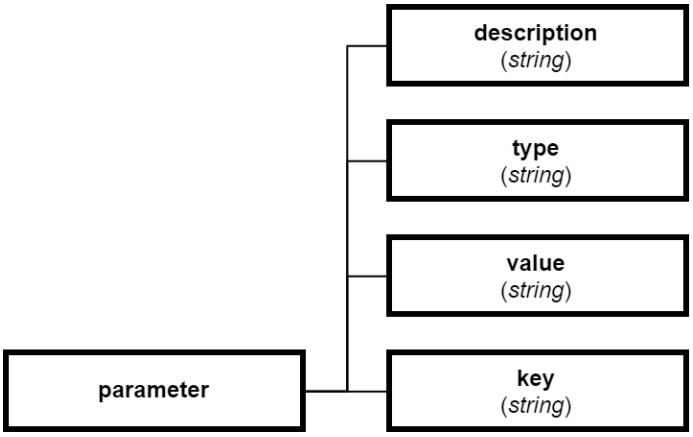


Figure 5:5 dataset node of the GMQLImporter configuration XSD schema.

- **parameter**: defines specific information for a source or a dataset, this information is useful for downloading, transforming or loading procedures.
  - **key**: is the name for the parameter, its identifier.
  - **value**: parameter information.
  - **description**: explains what the parameter is used for.
  - **type**: optional tag for the parameter.

5.3 Database design

A really important requirement for the solution is to maintain the genomic repository updated and provide historical status for every file and to know when there are new files or

ones that are obsolete. The database for GMQLImporter allows knowing the status for every file, on every dataset, on every source ever imported through GMQLImporter. It is possible to know whether a source was downloaded or transformed, also it's belonging datasets and all the files inside the dataset. The full database design description is in the appendix E "Database design for GMQLImporter" and the main components are:

5.3.1 Sources, datasets and files

To know exactly the status and which files were provided by the source in a given execution, a general structure for sources, datasets and files is contained inside the GMQLImporter's database, the possibilities to know how many files are provided on the source, how many were downloaded, transformed or loaded already. The history for every file, to know when they were updated is also contained. With this implementation, it is possible to know the status of the files inside the repository at any given point in time, with the correspondent details of the files at that time. GMQLImporter can give important feedback for every execution so the user can know easily the status of the repository.

5.3.2 Statistics

During the execution of GMQLImporter some important information is collected, such as the number of available files for download from the source, the amount of failed or replaced files, the correctly transformed ones and the also the wrong ones. In every run the total number of files available to download or transformed is stored, together with the number of correct and failed ones, after the number of files that do not fit the dataset's schema. GMQLImporter can provide this statistical historical data as it keeps track of it in the database.

5.3.3 Configuration

On every run, the configuration XML file could differ and give a different output or substantial change to the status of the repository, therefore the information provided in the configuration file is stored for every run to know what the desired actions to perform in

GMQLImporter were. As the configuration XML schema, is needed to store the general settings information, the specifications for every source involved and their datasets.

5.3.4 Database model

Here the main features of the database model are explained, most important indices and methods:

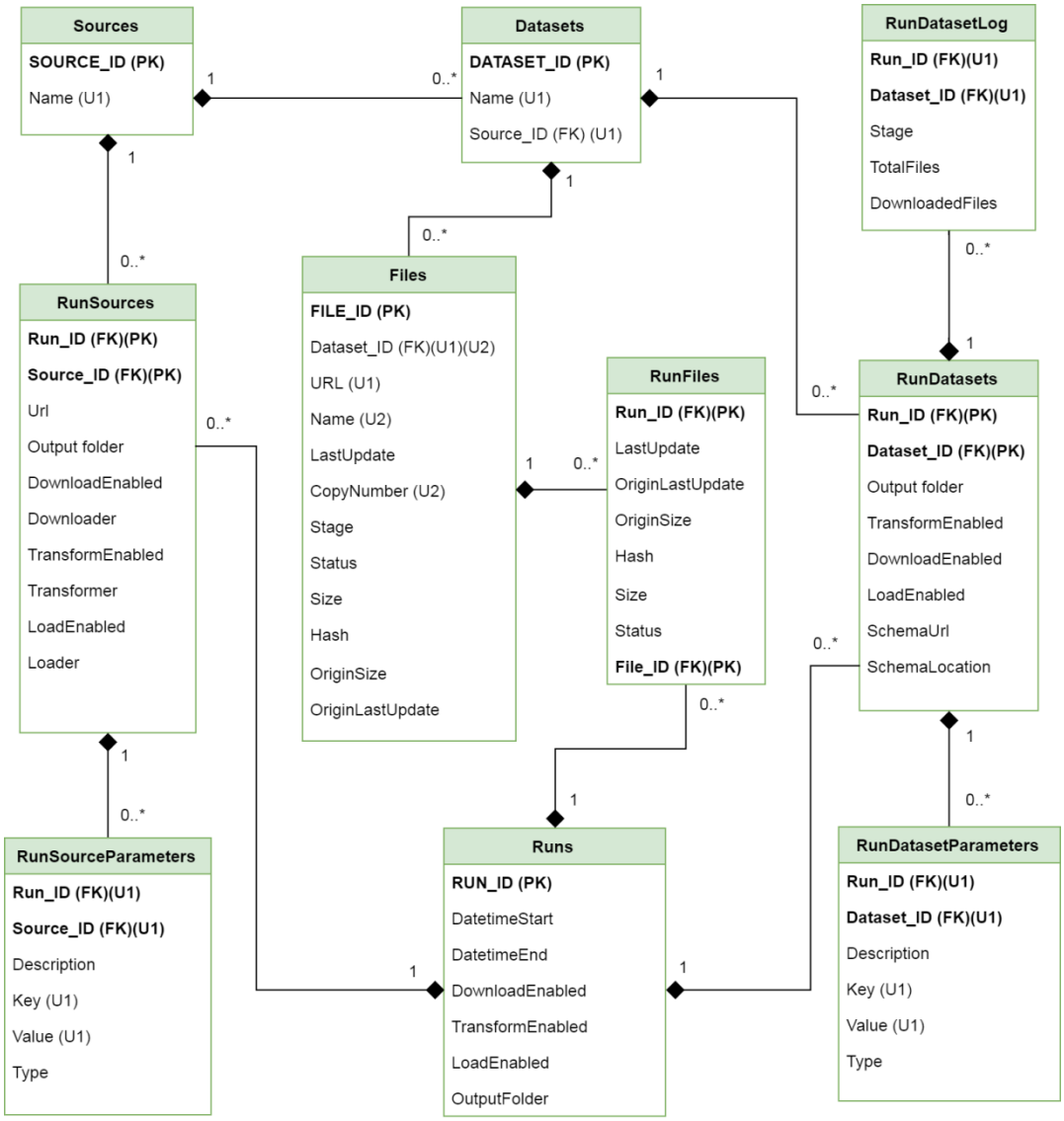


Figure 5:6 database model for GMQLImporter.

In this model, the sources with their datasets and files are represented by the tables Sources, Datasets and Files; the configurations for the execution are saved in tables Runs, RunSourceParameters and RunDatasetParameters where the configuration XML file parameters are saved distributed in the same way the XSD schema defines. Statistics are carried out using tables of RunFiles and RunDatasetLog where information is stored during runtime to let the user know the total available files to download and also failed, outdated and updated files during the execution.

To know the status of any file, 4 statuses are defined for a file:

- Updated: The file download/transformation is correctly performed in the local repository as it was in the server the last time the dataset where the file belongs was downloaded.
- Failed: The file download/transformation failed and the file may not be valid, in this case the file is marked as failed.
- Outdated: The file was removed from the server; this causes the file to exist locally but not remotely.
- Compare: Auxiliary status used to know which files do not exist anymore on the server, this status is meant to be used while the program is executing.

The following diagram shows the transitions of different file statuses and after every function inside is explained.

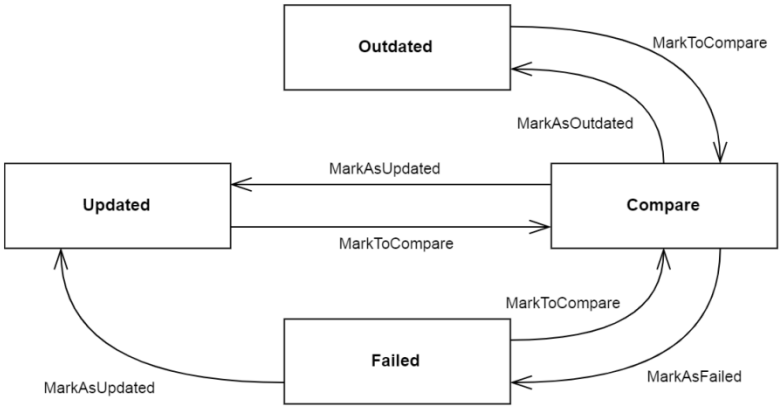


Figure 5:7 file status usage diagram for GMQLImporter.

The main database methods include:

- MarkToCompare: by receiving a dataset, the database changes the status of every file in the dataset as to compare, this method is used to notice which files are no longer in the server side and have to be marked after as outdated in the local copy.
- MarkAsUpdated: indicates the file was correctly downloaded or transformed and it is ready for the next step that could be transformation or loading.
- MarkAsFailed: when trying to download or transform a file, if the procedure fails, the file has to be excluded from further processing and thus it is marked as failed.
- MarkAsOutdated: once the whole dataset is downloaded or transformed this procedure allows finding the files that no longer exist in the remote server, and marks those local files as outdated. These files are no longer used in transformation or loading procedures.

Chapter 6. Implementation

The GMQLImporter module uses several technologies and frameworks. The project is mainly developed in Scala programming language. It is used to import GDM processed data into GMQL through GMQLRepository, the interface used by GMQL to load the datasets. Before the implementation of GMQLImporter, the importing processes for GMQL were done by running specifically developed scripts for TCGA2BED and ENCODE sources.

As defined in the design, the overall importing process is decomposed into 3 sequential steps. GMQLImporter has to tolerate all the possible faults from multiple technologies involved in the process such as internet loss or missing files in the source and also needs to provide enough feedback for the user to know the actual status of the repository. This chapter explains these processes in detail as implemented in GMQLImporter.

6.1 Downloader

The first step is downloading the files from the source's server, and as discussed in the design, multiple download methods can exist depending on different servers. The implementation for downloading is done by using a generalization of the process of download for any source. The generalization includes the ability to download by source, where every source has its datasets, only datasets that are marked to download will be downloaded. During the download, the downloader has to feed the database with the files it is checking to download and downloading, indicating if the file is correctly downloaded, outdated, updated or failed. Also the generalization was implemented with the possibility of downloading the failed files, in this way not the whole downloading process has to be checked again and can download just the missing files.

The implementation done in this thesis comprehends the implementations of FTPDownloader and ENCODEDownloader, the first one goes over folders provided by a FTP server, finds folders to download by using regular expressions and inside those folders uses regular expressions to know which files to download. The FTPDownloader is used to download

TCGA2BED datasets and could be used in many other FTP servers. On the other side, ENCODEDownloader is used specifically for ENCODE data, it uses the provided HTTP batch download by ENCODE discussed in appendix C“ENCODE Metadata Explanation for GDM” and by specifying a set of rules to generate an specific URL, allows to get an index file with the references to download every region data file with its metadata from ENCODE. These 2 different implementations follow the general procedures of downloading files and downloading failed ones.

6.2 Transformation

The transformation step consists in turn NGS data into GDM friendly format and to modify if needed the metadata provided. In this process, if the data is not provided as needed for the GDM repository, a transformation of the files is needed. The transformation process is done different for different types of files, therefore for every type of files have to exist a transformation definition to turn it into GDM. The data transformers have to provide the ability to transform any specific file and to know how many files are derived from it, to allow the insertion in the database to know the origin of every file that is going to be in the repository. After getting all region data and metadata in GDM friendly formats, it checks the region data files to know if they fit the dataset schema, then to make metadata easily interoperable changes metadata names as indicated in the configuration. Before finishing the transformation step, the metadata names contained for the experiment have to be transformed if needed to be valid as Java identifier for its correct function in GMQL. Any transformer that follows the general procedure can be added later in the GMQLImporter module.

6.3 Loader

GMQL provides an interface for loading data into it, it is GMQLRepository and GMQLImporter has to communicate with it. By giving the GMQL user name to load a source’s datasets it checks the consistency of the dataset by checking it has a schema defined and that every region data file has its respective metadata. As by business rules of GMQL, a dataset cannot have added files once is already created, if this happens, GMQLImporter allows the user to

confirm the choice to delete the previous one to be replaced, or to change the name of the updated dataset using the configuration XML file. This process is done by the functions provided by GMQLRepository as check if the dataset already exists, delete a dataset and to load dataset into GMQL.

6.4 Fault tolerance

Due to the mix of technologies, there are many faults that can affect the GMQLImporter functioning. Internet connection is one of the possible problems, it may cause some file downloads to fail and this problem is solved by retrying on every failed file and giving proper feedback to the user. Some files can be provided by non-functioning links and GMQLImporter has to deal with it, if missing region data or metadata, the whole sample must be discarded from the importing process and give feedback to the user. Files also can be filtered locally to ensure correctness and completeness if the source provides the data with anomalies, this includes as example, metadata filtering for ENCODE files since the batch download in some attributes does not filter correctly. Also if files on a dataset do not fit to the schema, this is notified to the user.

6.5 Logging

GMQLImporter provides constant feedback to the user both by console and by file logging. These logs allow the user to know the updated status while GMQLImporter is running and also to search over the operations done in the run. Also at the end of each run, a general statistics report for the run is given for the different sources and datasets.

Every GMQLImporter module defines its log, as the downloaders, importer, transformers and loaders. Full description for the implemented logging is on appendix F “GMQLImporter console & file logging”.

The logging structure for the messages has 4 levels: DEBUG, INFO, WARN and ERROR. The messages are given in hierarchical order depending in the process being run, this means first messages correspond to general process then download messages for each source and da-

taset, then transformation and finally loading. DEBUG messages are used for checking the correct working and fine grained status in the run, example of this are checking if the internet connection is active, to know also the changes done in metadata files, notify if a folder is created, among others. INFO messages indicates which action is being done and when it finishes, as example notifies when a file is being downloaded or transformed and when the process is done, also gives information about the configuration for the user to check whether is correct. WARN messages tell the user if something is not correct but the program still can handle the processing; it can be a file that does not match the schema or if the local files referenced do not exist. ERROR is used to notify a file download failed after retrying, if a link is not working or if the given parameters are causing problems and the system cannot work properly.

Different information is shown on console log and on file log throughout the execution. The console log is designed to let the user know the overall status, including messages as ERROR, WARN and INFO. The file logging is designed for debugging of the GMQLImporter functioning, it gives sufficient information to know where the program failed if so and to generate a fast identification of the problem.

6.6 Technologies used

Many technologies are used in the development of GMQLImporter, the main ones are explained below:

6.6.1 Scala

Scala is a highly scalable programming language that combines both functional and object oriented languages. It runs on the Java Virtual Machine (JVM) by compiling Scala to Java bytecode, so Scala and Java can be freely mixed. Scala allows a much faster development than plain Java with the same performance. Conceptually Scala is a pure object oriented

language, all variables and values are objects and any operation is a method. Allows easy transition to a functional style programming and is backed up by a huge range of libraries, frameworks and IDEs. Scala is maintained by the Scala community using GitHub for code management.

6.6.2 SLF4J + log4j

Simple Logging Facade for Java (SLF4J) is a Java logging API that serves as abstraction for multiple logging frameworks, such as the one used in GMQLImporter, log4j. This logger is set up at runtime and allows the communication in the console, and also straight to a file. The logging framework provides a level oriented messaging with the following levels (in decreasing priority order) ERROR, WARN, INFO, DEBUG or TRACE; ERROR is meant to communicate wrong functioning of the process, WARN is used to notice the user that something is not completely correct but the program can still work, INFO is to communicate general message feedback to the user in the application, DEBUG is used to describe the internal steps taken in the program for developers to have extra information during the program run, and TRACE is to fine grain the debugging by communicating more details to developers. The logger has the option to set up a threshold level, this allows to limit information shown to the user and to easily create multiple types of log, such as in GMQLImporter where log4j is used to show just up to INFO to the user, and DEBUG for developers.

6.6.3 Maven

Maven is a tool used mainly in Java projects for build automation; it describes how the software has to be built, including dependencies used inside the project and how those components are built together. It uses an architecture based in plugins, and provides a full plugin catalog for easily using them inside a project. The full configuration relies in a Project Object Model (POM) XML file. The full development of GMQLImporter is done using Maven, causing the project to be easy to expand, as the programming environment is easily portable.

6.6.4 H2Database

H2Database is a lightweight database management system that runs embedded in the program code, this means it needs no installation and as it runs in Java over the JVM, is easily portable between many platforms. In GMQLImporter the database configured as text based database, the database allows a subset of SQL queries. Is a complete transactional database and provide SQL injection protection, encryption password by using SHA-256 and communication security as SSL or TLS connections.

Chapter 7. Evaluation

For the evaluation phase, the datasets that are at present already loaded into GMQL-Repository are chosen for testing the GMQLImporter module. In this chapter, the essential features on the creation of the configuration XML using the previously defined schema are explained. After download, transform and load processes are discussed and finally a summary of the overall results in the execution of the module is presented. The execution of GMQLImporter was done using 32 cores with 2 threads each Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz with 378 GB System memory with an internet connection tested between 605.39 and 749.43 Mbit/s during the download process, the test was done in parallel mode and also sequentially.

7.1 Configuration XML creation

The whole creation process is detailed in appendix G "Creation of XML configuration file for GMQLImporter" and here, the main settings used for evaluation are explained. The included sources to be imported are 2 sources already included in GMQL and the new imported datasets will be the update of them, the sources are TCGA2BED and ENCODE. Following the schema for the configuration file, the datasets desired to load into GMQL are:

TCGA2BED	ENCODE
cnv	HG19_broadPeak
dnametylation	HG19_narrowPeak
dnaseq	GRCh38_broadPeak
mirnaseq_isoform	GRCh38_narrowPeak
mirnaseq_mirna	
rnaseq_exon	
rnaseq_gene	
rnaseq_spljxn	
rnaseqv2_exon	

rnaseqv2_gene	
rnaseqv2_isoform	
rnaseqv2_spljxn	

Table 7:1 datasets from ENCODE and TCGA2BED to be imported.

7.2 Changes of implementation and adaptation

The versatility of the GMQLImporter was also proven during the development, as the requirements changed constantly, the XML configuration file provided enough freedom to give multiple parameters for the correct download and transformation of the datasets. The presented module is robust with fault tolerance. Also the 3 steps of download, transform and load gave a correct abstraction for the problem proposed and allowed the generalization of the Importing process.

7.3 Parallel run

To evaluate GMQLImporter it is tested using both parallel and non-parallel executions, as the process includes multiple downloads and file processing in a very tidy way, the overall process is highly parallelizable. The download or transformation of multiple datasets at the same time is enabled for testing the parallel execution of GMQLImporter and it is activated or deactivated in the XML configuration file. The main variable to compare is the time used for the complete process of downloading and transforming the data, both runs are executed in different times and using different folders to isolate the processes.

7.4 Results

The imported data results are given in the following table specifying comparable times (expressed as HH:MM:SS) for download and transform, using parallel download and using sequential download:

Dataset	Nº of samples	Size	Download Parallel	Download sequentially	Transform parallel	Transform sequentially
cnv	22632	869 MB	19:53:17	75:57:58	02:59:47	01:02:27
dnamethylation	12860	235 GB	19:55:28	76:05:16	04:44:48	03:19:32
dnaseq	6914	305 MB	19:52:54	75:57:06	00:58:09	00:18:27
mirnaseq_isoform	9909	4.1 GB	19:52:55	76:19:14	01:55:57	00:31:48
miranseq_mirna	9909	775 MB	19:52:56	76:18:21	01:28:03	00:26:51
rnaseq_exon	3675	46 GB	18:01:29	72:35:19	01:38:35	00:40:39
rnaseq_gene	3675	5 GB	18:59:35	72:37:32	00:57:03	00:14:56
rnaseq_spljxn	3675	43 GB	19:05:05	72:58:50	01:37:14	00:40:57
rnaseqv2_exon	9825	119 GB	19:54:09	76:01:29	03:19:26	01:46:39
rnaseqv2_gene	9825	21 GB	19:53:06	75:58:17	02:24:43	00:43:14
rnaseqv2_isoform	9825	50 GB	19:53:22	75:59:30	02:32:07	01:02:20
rnaseqv2_spljxn	9825	110 GB	19:54:08	76:02:03	03:16:03	00:41:23

HG19_broadPeak	1534	7.7 GB	02:47:45	02:19:53	01:19:24	00:27:11
HG19_narrowPeak	9783	54 GB	19:17:04	20:03:15	03:41:39	02:07:12
GRCh38_broadPeak	367	3.4 GB	00:36:44	00:37:52	00:11:29	00:04:02
GRCh38_narrowPeak	9415	57 GB	19:12:24	18:16:22	03:48:32	02:12:57

Table 7:2 execution times in with and without parallelization for each dataset.

It is worth to mention the times on TCGA2BED datasets for download always have similar times for download as the download is done by traversing the folder structure in the FTP repository; this causes all the datasets to start the downloading process on the first folder visited and to end in the last folder to visit.

For calculating the total download time sequentially, the maximum time spent for a dataset download in TCGA2BED (76:19:14) is considered as the total time for downloading that source while in ENCODE, the sum of all download times for each dataset is the total time for downloading the source (41:54:22); giving a total time for download sequentially of 118:13:36. Downloading in parallel for TCGA2BED speeds up every dataset as 1 process takes care of traversing the repository and multiple sub processes download the files; for ENCODE in parallel download the speed up is also substantial as all datasets are downloaded at the same time instead of waiting sequentially. For the total time of download in parallel, the maximum time for a dataset download is considered as the total time for downloading all the sources, giving a total time for parallel download of 19:55:28.

As for the transformation time when done sequentially, the total transformation time of all sources is equal to the sum of every transformation time for each dataset thus the total transformation time sequentially is 16:20:35. When transformation is done in parallel, multiple dataset are processed at the same time, therefore sub processes share the local resources causingthe time for individual datasets to increase; but as for parallel execution, the

total transformation time is equal to the maximum transformation time among the datasets which is 04:44:48 and therefore the parallel execution speeds up the overall transformation process.

All these datasets were loaded into GMQL in a total time of 11:43:44. The loading process is always done sequentially.

Chapter 8. Conclusion and future work

GMQLImporter by using correspondent generalization for the process of retrieving genomic data and metadata from multiple sources, standardize them towards their integration, and importing them into a GDM repository for their comprehensive processing through GMQL, effectively accomplishes these tasks and allows easy further development for the inclusion of new data sources that may be added later.

GMQLImporter solves the problem of supporting the automated gathering of genomic data with their respective metadata into a GDM repository, providing the correspondent feedback to the user. Thus, the process of importing distributed heterogeneous data for their integrated GMQL processing is highly automated and no complex supervision is needed.

As the presented results show, the use of parallel execution speeds up substantially the downloading process by taking less than a fifth of the time for the sequential download; and the transformation process done in parallel takes less than a third of the time for sequential transformation. These times can be improved even more by optimizing the parallelization processes as doing transformation in parallel to handle multiple files for each dataset at the same time.

As part of future work, new downloaders and transformers will be developed; also further analysis of clinical metadata to allow a complete interoperation between data from different sources will be done.

References

[1] V. Jalili, M. Matteucci, M. Masseroli and S. Ceri, "Indexing Next-Generation Sequencing data," *Information Sciences,* vol. 384, no. 1, pp. 90-109, 2017.

[2] M. Masseroli, A. Kaitoua, P. Pinoli and S. Ceri, "Modeling and interoperability of heterogeneous genomic big data for integrative processing and querying," *Methods,* vol. 111, no. 1, pp. 3-11, 2016.

[3] The Cancer Genome Atlas, "THE CANCER GENOME ATLAS," [Online]. Available: https://cancergenome.nih.gov/. [Accessed 2 6 2017].

[4] F. Cumbo, G. Fiscon, S. Ceri, M. Masseroli and E. Weitschek, "TCGA2BED: extracting, extending,integrating, and querying The Cancer GenomeAtlas," *BMC Bioinformatics,* vol. 18, no. 1, p. 1, 2017.

[5] The ENCODE Project Consortium, "An integrated encyclopedia of DNA elements in the human genome," *nature,* vol. 489, pp. 57-74, 2012.

[6] M. Masseroli, P. Pinoli, F. Venco, A. Kaitoua, V. Jalili, F. Palluzzi, H. Muller and S. Ceri, "GenoMetric Query Language: a novel approach to large-scale genomic data management," *Bioinformatics,* vol. 31, no. 12, pp. 1881-1888, 2015.

[7] S. Ceri, A. Kaitoua, M. Masseroli, P. Pinoli y F. Venco, «Data Management for Heterogeneous Genomic Datasets,» *IEE/ACM Transactions on Computational Biology and Bioinformatics,* vol. PP, nº 99, pp. 1-1, 2016.

[8] C. Kozanitis, A. Heiberg, G. Varghese and V. Bafna, "Using Genome Query Language to uncover genetic variation," *Bioinformatics,* vol. 30, no. 1, p. 1, 2014.

[9] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly and M. DePristo, "The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data," *Genome Research,* vol. 20, no. 9, pp. 1297-1303, 2010.

[10] S. Neph, S. Kuehn, A. Reynolds, E. Haugen, R. Thurman, A. Johnson, E. Rynes, M. Maurano, J. Vierstra, S. Thomas, R. Sandstrom, R. Humbert and J. Stamatoyannopoulos, "BEDOPS: high-performance genomic feature operations," *Bioinformatics,* vol. 28, no. 14, pp. 1919-1920, 2012.

[11] H. Li, B. Handsaker, A. Wysoker, T. Fennel, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin and 1. G. P. D. P. G. , "The Sequence Alignment/Map format and SAMtools," *Bioinformatics,* vol. 25, no. 16, pp. 2078-2079, 2009.

[12] A. Quinlan y I. Hall, «BEDTools: A flexible suite of utilities for comparing genomic features,» *Bioinformatics,* vol. 26, nº 6, pp. 841-842, 2010.

[13] A. Goguel d'Allondans, S. Boutillier, D. Uzunidis y N. Labère, Méthodologie de la thèse et du mémoire, Studyrama - Vocatis , 2012.

Appendix A. Requirements Analysis for GMQLImporter

A.1. Project objective

This project requires developing an automatic system to build and keep updated a big integrative repository of genomic data and their metadata publicly available from several specific sources, transform them into GDM format to be seamlessly used for genomic data processing through GMQL.

A.2. Project summary

The aim is to make easy integrating heterogeneous genomic data and performing complex processing on them. GMQL is currently being developed in Scala programming language and for this module to be an extension; Scala is also used in the development of GMQLImporter module. Data and their metadata are provided by Sources and kept organized in Datasets with a variety of formats, to load the data inside GMQL, these have to be in a compatible format, have a valid schema file and one file of metadata for every file of region data; the GDM is being used in the GMQL project for region data and metadata representation. The main sources of datasets to be imported into GMQL are the Encyclopedia of DNA Elements (ENCODE) and The Cancer Genome Data into bed format (TCGA2BED). The module has to allow easy addition of future data sources to be imported.

A.3. Requirements list

During the analysis, a starting point for the overall process was set by defining a list of requirements to be accomplished in the software implementation:

A.4. Heterogeneous genomic data and metadata download from multiple different sources

- The software must support different downloading protocols.
- Flexibility to allow downloading from different sources and therefore it has to adapt to remote source data organization.
- Specific parameters for download are defined by source and/or dataset.

A.5. Genomic data and metadata transformation towards data integration

- Organize region data and metadata according to the Genomic Data Model (GDM), this may include processes as:
    - Extract compressed files.
    - Obtain metadata from different files.
    - Define or change a region data schema.
- Modify/standardize data/metadata attribute names, contents or format.

A.6. Load downloaded and transformed data and metadata into an integrative GDM repository

- Connect to the repository by using provided API according to the rules defined in the repository.
- Create, delete and update datasets into the repository.

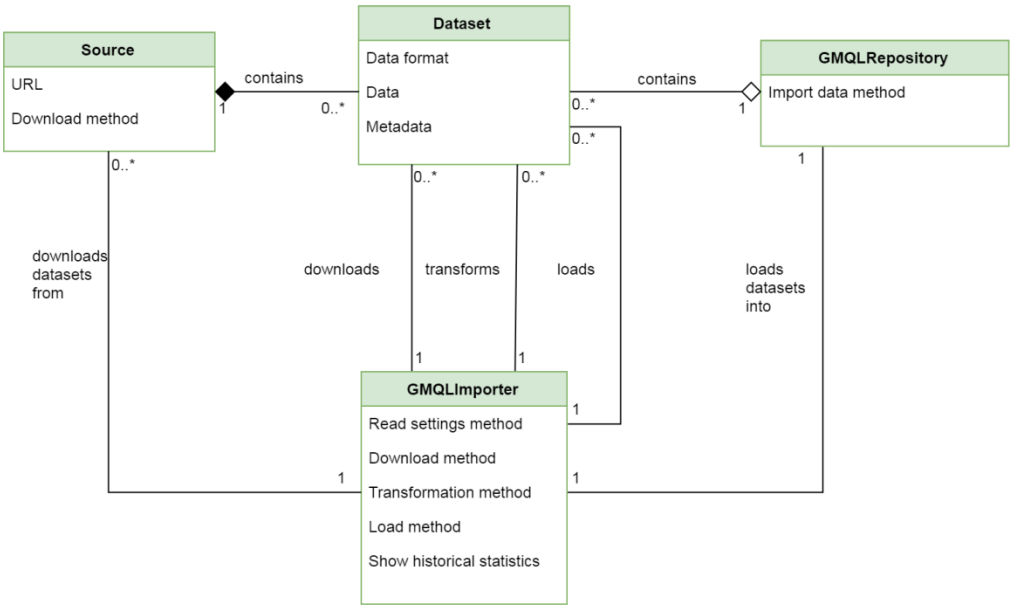
A.7. Automatic checking, monitoring and recording quantitatively

- Keep track of the processes done in region data and metadata from its download to its loading into the repository.
- Control automatically when possible the correctness and completeness of the steps performed to the data and metadata.
- Provide quantitative summary descriptions of the process steps.
- Support in efficient and feasible way the easy evaluation of correctness and completeness on each step.

A.8. Support data and metadata update and extension

- The software has to allow easy extension of the processes previously performed over the data and metadata for further processing or for aggregation of similar sources, datasets, region data, metadata or other files.
- Newer versions of the data with their respective metadata provided by the sources must be easily updated in future executions. As different sources release new information at different pace, the software must check for updates in the origin source and add the new data and metadata to the process.
- The main application of the development must be done for the sources of TCGA2BED and ENCODE and the loading process into GMQL repository.

A.9. Initial domain model



Entity	Description
Source	NGS region data and metadata provider.
Dataset	Collection of region data and metadata, where all samples respect a given region data schema.
GMQLRepository	Interface to connect and import datasets inside GMQL.
GMQLImporter	Downloads multiple datasets from different sources. Transforms downloaded datasets into GMQL compatible format (GDM) with standardized metadata. Loads GDM compatible format datasets into the GMQLRepository. Tracks historically the status of downloaded and transformed files.

A.10. Key actors and tasks

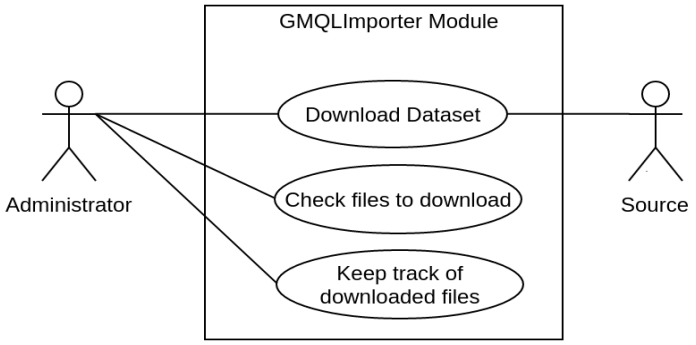
Actor	Description
Source	Provider of the datasets used in the GMQLRepository.
GMQLRepository	Module of GMQL project that has the datasets to be queried.
GMQLImporter	Module of GMQL project that obtains datasets from sources and transforms their region data and metadata into GDM to be loaded inside the GMQLRepository. Keeps track of downloaded, transformed and loaded files in order to keep the GMQLRepository updated and non-redundant.
Administrator	Configures the GMQLImporter based on his knowledge of the sources and their datasets

Task	Description
Download	Gets datasets from a source, has to keep the files updated.
Transform	Transforms and/or process data and metadata into GDM format to be imported in GMQLRepository, has to keep the files updated.
Load	Inserts into GMQLRepository GDM compatible datasets with their corresponding schema.

A.11.

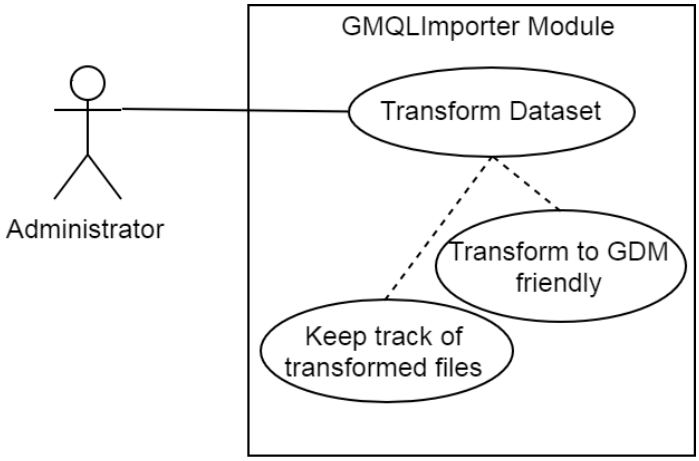
A.12. Use cases

A.13. Download



Name	Download datasets from source
Description	By giving description of a source and specifying types of datasets of interest, the application downloads those datasets and puts them ordered into a local folder.
Preconditions	The source and its datasets have to exist. The source downloading procedure has to be defined previously.
Post conditions	There is a local copy of the files in the source ordered in separate folders by dataset.
Nonfunctional Requirements	Is needed to track of downloaded files, and when download only download files to be updated if already have been downloaded. Keep the user notified about the loading process.

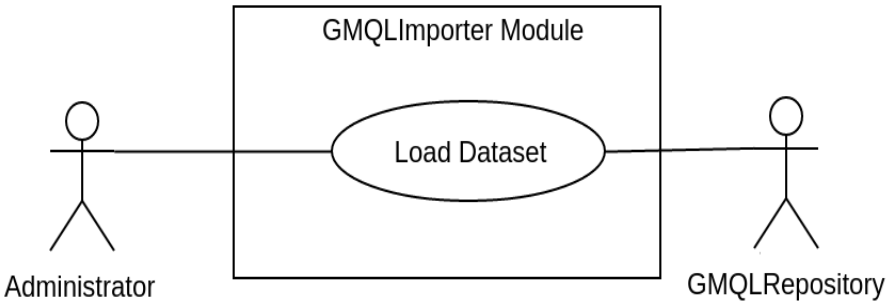
A.14. Transform



Name	Transform datasets from source
Description	By giving description of a source and specifying datasets of interest, the system if needed applies a specific transformation for the files to be in GDM format for further data or metadata standardization.
Preconditions	The source and its datasets have to be already downloaded and their transformation procedure has to be defined previously.
Post conditions	There is a transformed copy of the source's original datasets in the destination folder in GDM compatible format.
Nonfunctional Requirements	Is needed to track of the files to know their status. Keep the user notified about the loading process.

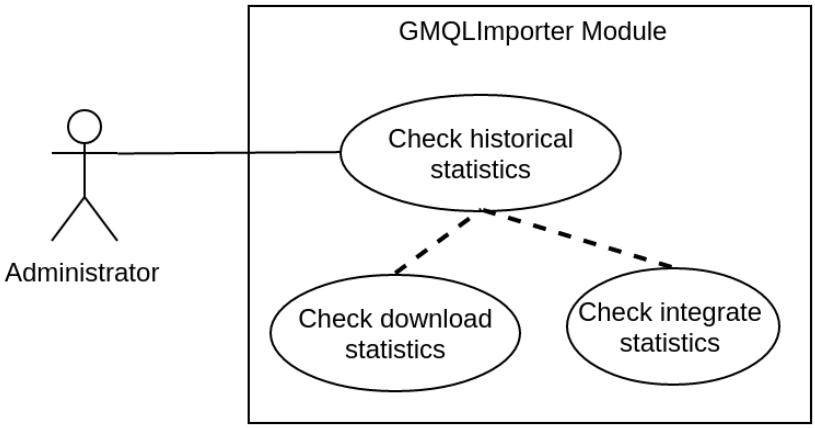
A.15.

A.16. Load



Name	Load datasets from source
Description	By giving description of a source and specifying datasets of interest, the application loads into GMQLRepository those datasets with their respective schema file.
Preconditions	The source and its datasets files have to be in a GDM compatible format with region data and metadata plus the schema file.
Post conditions	The datasets are available for querying inside GMQL.
Nonfunctional Requirements	Keep the user notified about the loading process.

A.17. Check historical statistics



Name	Check historical statistics
Description	Shows to the administrator the statistics of files tracked while downloading and transforming datasets.
Preconditions	Files are tracked already in download or transform steps, indicating how many files have been downloaded or transformed and information about the problems had in the process.
Post conditions	
Nonfunctional Requirements	Has to provide enough information to understand the status of the GMQLRepository, downloaded and transformed files.

Appendix B. Specifications for GMQLImporter

B.1. Specifications Summary

This appendix is a continuation of appendix A “Requirements Analysis for GMQLImporter”. Shows how data is downloaded from different Sources, how the data is transformed into a GDM compatible format and how it is loaded for each Source. The sources explained are TCGA2BED and ENCODE. After explaining the process, a generalization of the sources and their procedures is presented, followed by a new general domain model of the GMQLImporter module.

B.2. Requirements Analysis by Source

This section explains how the data is acquired;specifies the data location, retrieval methods, and how to import them into GMQLRepository. For inserting files into the repository, they have to be in GDM format, separated by experiment every data and metadata file. Also a schema file is required to define the file format.

B.3. TCGA2BED

Contains data from TCGA converted in BED format, for experiments of DNA-Seq, RNA-Seq (V1 and V2), DNA-Methylation, miRNA-Seq and CNV. Data is located inside a FTP repository where original (TCGA) and converted (TCGA2BED) files are stored. Inside the converted directory the files are grouped by tumor name into folders with their respective tumor tag, inside every tumor folder, the files are again grouped by experiment type into folders with the experiment name. Each of those folders contains: the dataset schema, file with total number of files in the folder, file with md5 checksum for every file in the folder, region files and metadata files. The region data files are presented in BED format with its associated metadata, each experiment has a .bed and a .bed.meta file. Inside GMQLRepository the

TCGA2BED files should be grouped by experiment type and not by tumor name. Files are downloaded from the source; grouped by experiment type and then the schema file is added, after this if needed, metadata can be modified before being imported into GMQLRepository.

B.4. ENCODE

Contains data for a great amount of experiments and data types, data is provided by HTTP access using a direct link for downloading every file, the file format depends on the type of experiment. ENCODE allows 3 types of metadata download as explained in appendix C "ENCODE metadata explanation for GDM". To load ENCODE data inside GMQLRepository is necessary to download every data file, initially the GMQLImporter project aims to integrate BED broadpeak and BED narrowpeak format types of ENCODE, so bed files have to be downloaded, they are provided in compressed containers (.gz) so extraction after downloading is needed. For the creation of metadata files is necessary to download the metadata and then create for every data file, a respective metadata file by merging the rows from both metadata files or extracting the metadata from JSON files downloaded. Data and metadata have to be put together with the dataset schema and then they are ready to be imported into the GMQLRepository.

B.5. Model Generalization

Thinking in a complete generic model to describe all the available sources and datasets is not possible, but the main operations needed to achieve correct integration are possible.

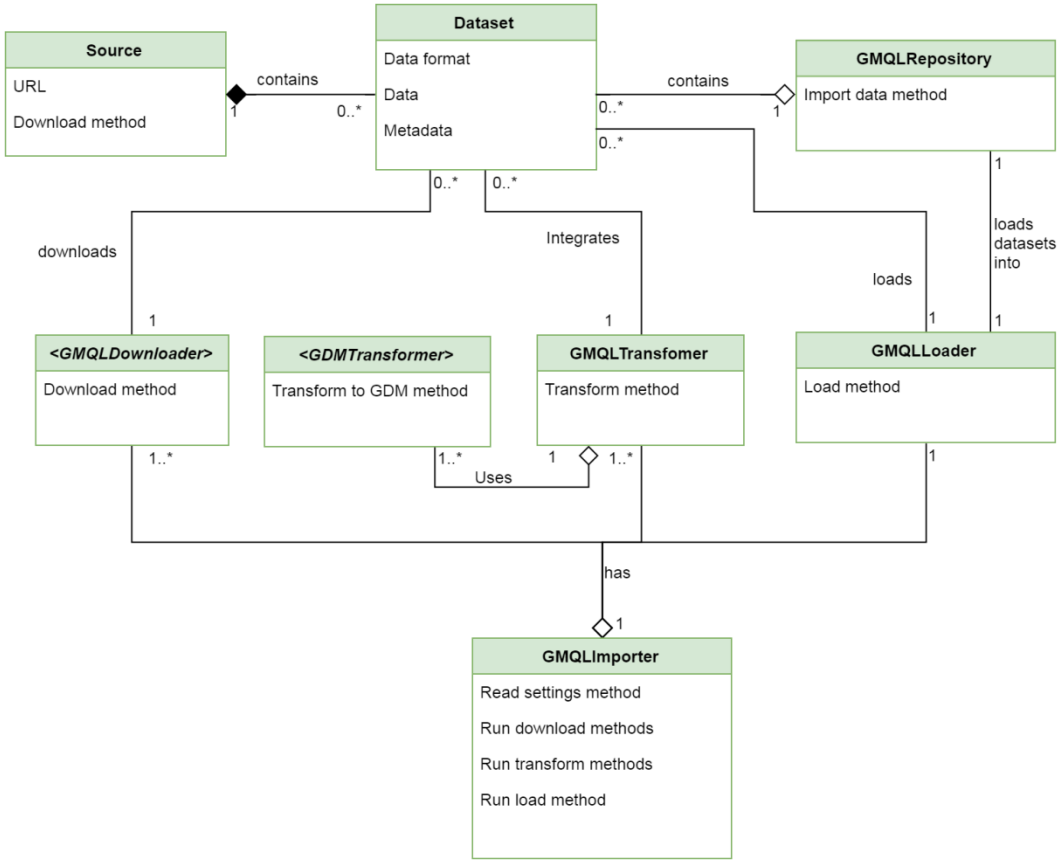
For transforming procedure, initially complex data file transformation is not needed, but in the future could be, and this problem has to be addressed. TCGA2BED has the files just as GMQLRepository needs them just not in the order it is desired. ENCODE data files come in .gz containers that have to be extracted, their metadata has to be separated into several files, and source metadata files may be in different formats. At the loading stage, GMQLRepository needs to have the following files for each dataset:

- Schema file describing region data format.
- Set of region data files in GDM compatible format.
- Set of metadata with 1 metadata file for each region data file.

Loading procedure in the development is by using the GMQLRepository interface provided by GMQL project, but any other repository could be used and that flexibility also has to be addressed.

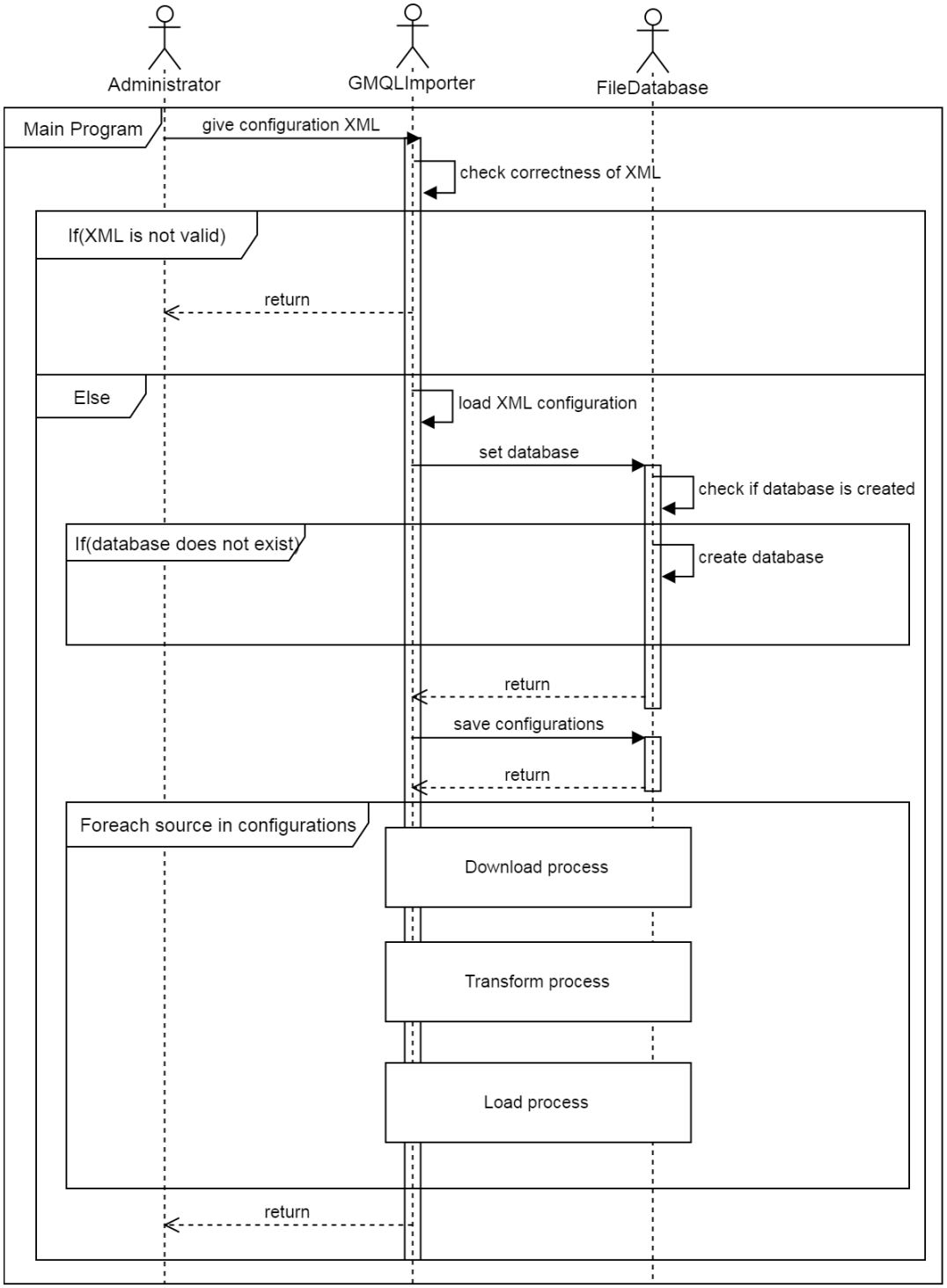
B.6. General Domain Model

The GMQLImporter module has been divided by the 3 basic operations to be performed in the import process; those ones are download, transform and load. For the transformation procedure, although similarities could be found, because of the variety of different NGS formats and the need of transform those to GDM format, GDMTransformer represents a type of transformation procedure, where a particular set of operations can be defined for each source to transform original files into GDM friendly ones. Same as for downloading procedure, no unique generic way to download datasets from sources is feasible so GMQLDownloader represents a type of downloading procedure, where any particular implementation for downloading different sources can be implemented. The new domain model is the following:

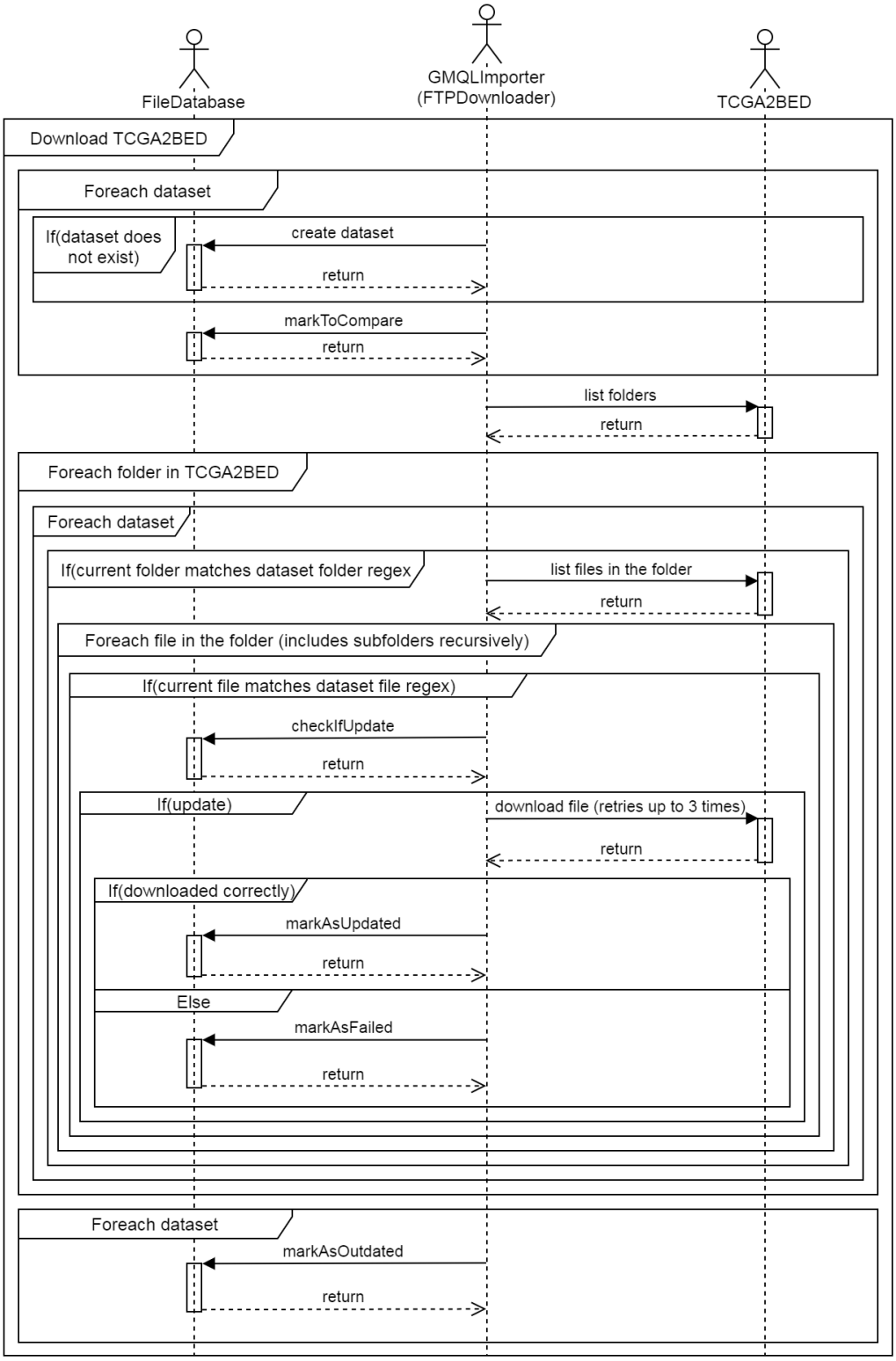


B.7. Sequence diagrams

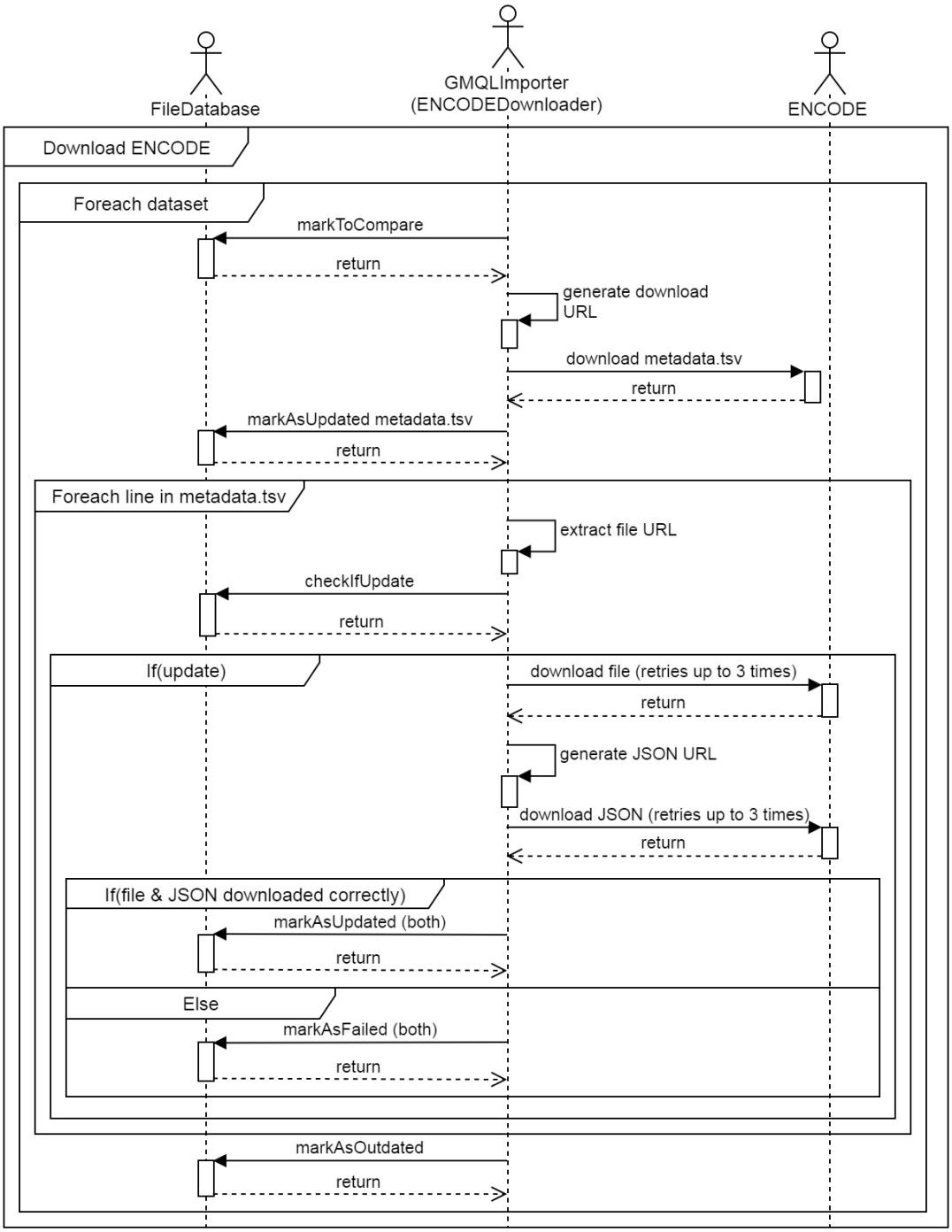
In this section the main processes for GMQLImporter are defined and also the details for the interaction sequences between the different modules involved in the software execution. The diagrams explain the main desired behavior:

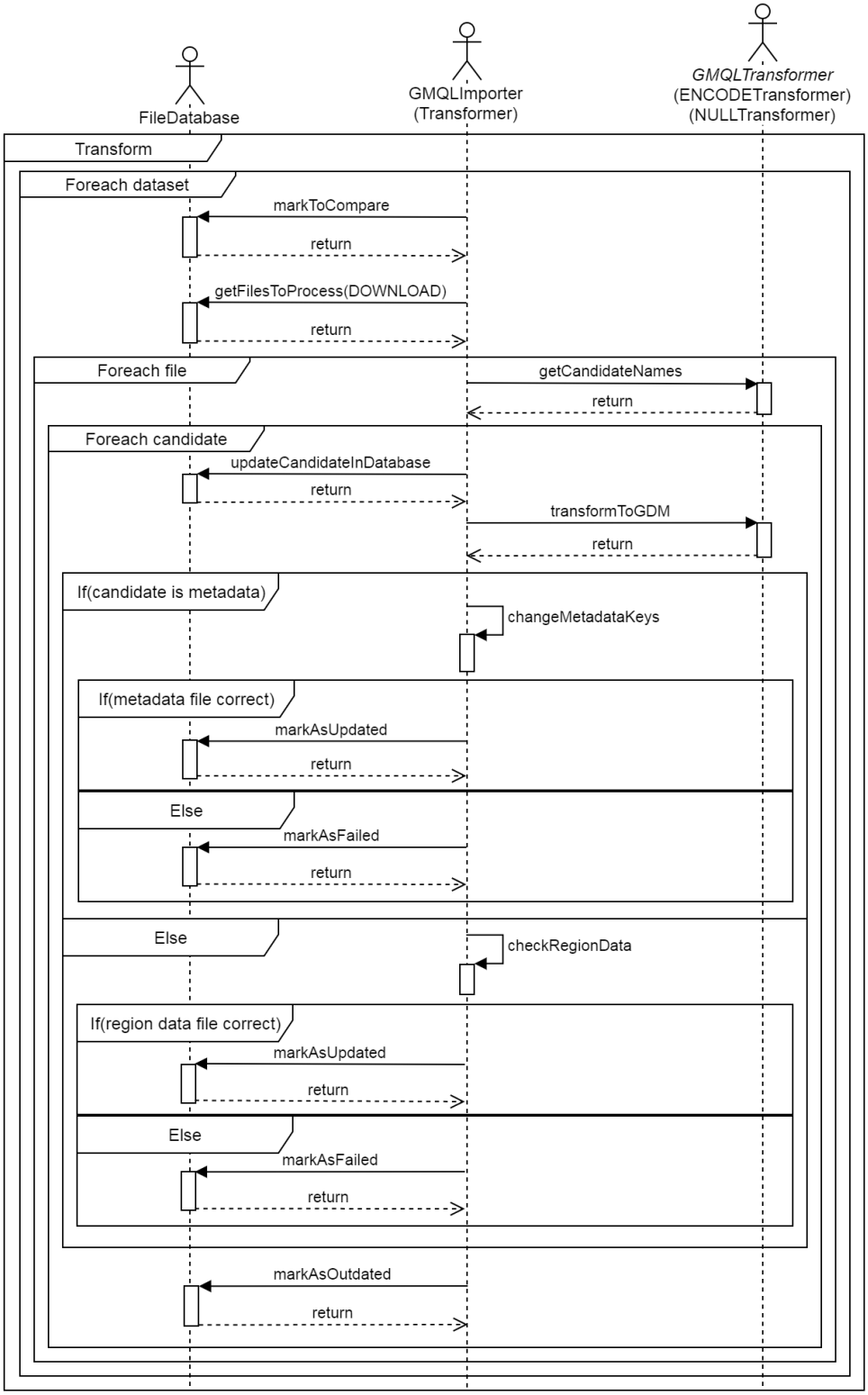

Administrator
GMQLImporter
FileDatabase
Main Program
give configuration XML
check correctness of XML
If(XML is not valid)
return
Else
load XML configuration
set database
check if database is created
If(database does not exist)
create database
return
save configurations
return
Foreach source in configurations
Download process
Transform process
Load process
return

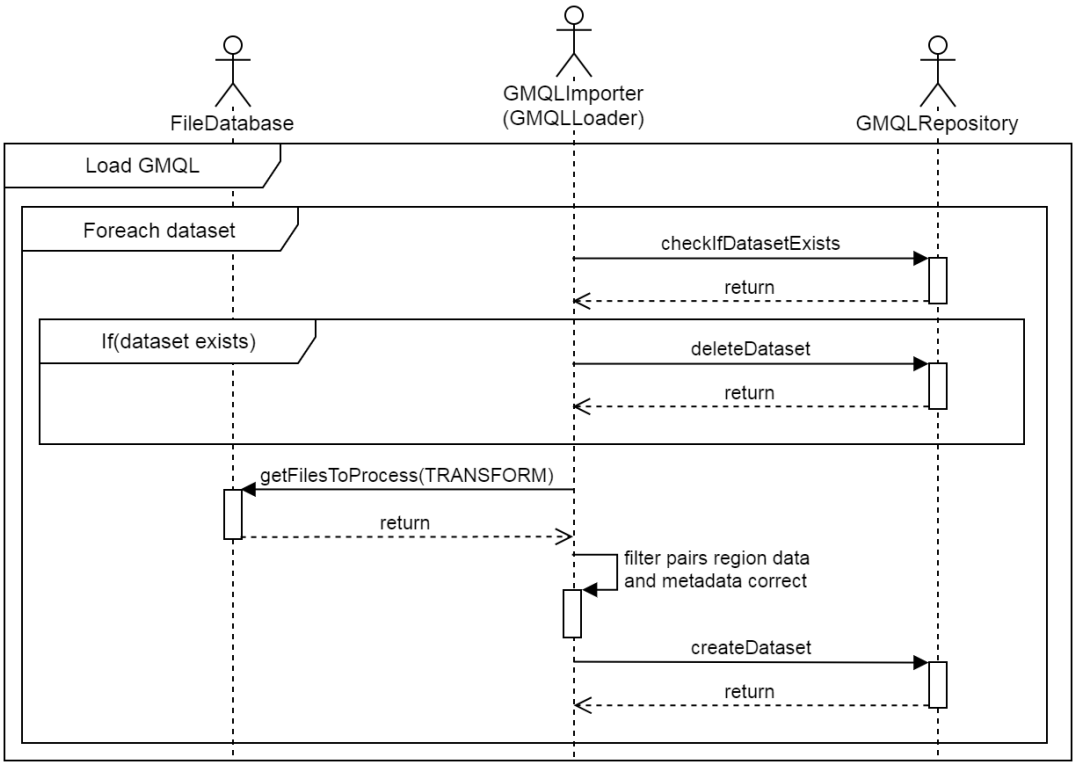
The main program loads the configuration from the XML file, with this configuration, the program recreates the sources requested and the datasets for every source. With this structure, iterates on every source for download, transform and load. For the download part, as discussed before, specific operations must be done separately for TCGA2BED and ENCODE, therefore the process of download happens completely separated between both sources. The main program by the giving configurations has to select the needed implementation for“Download process” and“Transform process”. Load process is done always by using GMQLLoader but may be extended to use another GDM repository.The transformation process has a general transformation part and a specific transformation part; in the specific one the files have to be transformed to GDM friendly format. Whilst in the general part, the region data schema and the metadata files are checked for correctness. The next steps shown are subsections of the overall execution and therefore assume the main program’s load of the XML configuration file and the existence of the FileDatabase.


FileDatabase
GMQLImporter
(FTPDownloader)
TCGA2BED
Download TCGA2BED
Foreach dataset
If(dataset does not exist)
create dataset
return
markToCompare
return
list folders
return
Foreach folder in TCGA2BED
Foreach dataset
If(current folder matches dataset folder regex
list files in the folder
return
Foreach file in the folder (includes subfolders recursively)
If(current file matches dataset file regex)
checkIfUpdate
return
If(update)
download file (retries up to 3 times)
return
If(downloaded correctly)
markAsUpdated
return
Else
markAsFailed
return
Foreach dataset
markAsOutdated
return

B.8.


FileDatabase
GMQLImporter
(ENCODEDownloader)
ENCODE
Download ENCODE
Foreach dataset
markToCompare
return
generate download
URL
download metadata.tsv
return
markAsUpdated metadata.tsv
return
Foreach line in metadata.tsv
extract file URL
checkIfUpdate
return
If(update)
download file (retries up to 3 times)
return
generate JSON URL
download JSON (retries up to 3 times)
return
If(file & JSON downloaded correctly)
markAsUpdated (both)
return
Else
markAsFailed (both)
return
markAsOutdated
return


FileDatabase
GMQLImporter
(Transformer)
GMQLTransformer
(ENCODETransformer)
(NULLTransformer)
Transform
Foreach dataset
markToCompare
return
getFilesToProcess(DOWNLOAD)
return
Foreach file
getCandidateNames
return
Foreach candidate
updateCandidateInDatabase
return
transformToGDM
return
If(candidate is metadata)
changeMetadataKeys
If(metadata file correct)
markAsUpdated
return
Else
markAsFailed
return
Else
checkRegionData
If(region data file correct)
markAsUpdated
return
Else
markAsFailed
return
markAsOutdated
return


FileDatabase
GMQLImporter
(GMQLLoader)
GMQLRepository
Load GMQL
Foreach dataset
checkIfDatasetExists
return
If(dataset exists)
deleteDataset
return
getFilesToProcess(TRANSFORM)
return
filter pairs region data
and metadata correct
createDataset
return

Appendix C. ENCODE Metadata Explanation for GDM

C.1.Metadata Organization

For the ENCODE project, the metadata for every experiment is distributed in a tree structure of entities, the root node is the experiment entity where the immediate metadata for the experiment is found, this includes metadata such as “biosample_type”, “date_released”, “replication_type” and “status”. Also in the root,non-immediate metadata is referenced, this comprehends metadata like “files”, “award” and “replicates”. The tree created is non-cyclical therefore exists a unique way of getting the information starting from the root. The root node contains 48 branches, 19 of them are leaves (immediate metadata referred as plain text) of the tree, 3 are other complex entities that represent by themselves a complete tree structure and the remaining 26 branches are arrays that can contain either leaves or other complex types.

C.2.Metadata Acquisition

Downloading metadata from ENCODE is a straightforward action that defines a metadata filter for the experiment files to download; this filter can select any metadata as criteria such as Genome Assembly, File Format, Output Type, Biosample Life Stage, among others.

ENCODE’s model has as root level the experiments, and all the experimental metadata is linked to this entity as explained before.

From the ENCODE project metadata can be downloaded in 3 different ways (where each one gives different amount and/or type of metadata).

C.3.Metadata.tsv

ENCODE defines this metadata.tsv file as a pre-defined set of metadata meant to be downloaded using their*Batch Download* method, it contains 48 fields of metadata associated to every region data file. The file defines a table where every row describes a sample file and every column gives metadata about them. The metadata contained is gathered from different nodes of the metadata tree and then the names are changed to the preferred term on ENCODE's ontology for the field name.

To download the metadata.tsv, the creation of a URL is needed. The URL creation is done in 3 blocks: prefix, filter and postfix.

The prefix is always the same:

https://www.encodeproject.org/metadata/

The filter is created by setting a conjunction of metadata filters, an example that gets the metadata for files related to experiment, where the files are bed narrowpeak and limits the files to be just the first 2 of the list:

type=Experiment&limit=2&files.file_type=bed+narrowPeak

The postfix is always the same:

/metadata.tsv

Example of download link for metadata.tsv with the filter explained before:

https://www.encodeproject.org/metadata/type=Experiment&limit=2&files.file_type=bed+narrowPeak/metadata.tsv

C.4.Tabular report

Tabular report is a customizable table, the table columns can be specified using any type of metadata related to the experiment, by acceding to ENCODE Rest API it can select any metadata to be displayed.

To download the report.tsv, the creation of a URL is needed. The creation includes 2 blocks: prefix and filter.

The prefix is constant:

https://www.encodeproject.org/report.tsv?

The filter is created as defined previously, an example that gets the metadata of files related to experiment, where files are bed narrowpeak:

type=Experiment&limit=2&files.file_type=bed+narrowPeak

Example of download link for report.tsv with the filter explained before:

https://www.encodeproject.org/report.tsv?type=Experiment&files.file_type=bed+narrowPeak

C.5.Rest API

ENCODE provides web services for getting the metadata or any information obtained in their website into JSON format, therefore the complete tree structure for each experiment can be downloaded. Full description for Rest API procedure is explained in appendix I "ENCODE metadata generation for experiment JSON".

C.6. Metadata Filtering and Renaming

As not all the metadata inside the provided JSON is related to the files, filtering is needed, an example of pruned branch is the "files" node, that contains all the files referred by the experiment but only one is needed, the one corresponding to the region data file. Another example of the filtering needed is the "replicates" array in the experiment node, it contains

all the biological replicates used in the experiment, but every file contains the references to the biological replicates used by the file, so the non-referenced have to be filtered out in order to have just metadata related to the file needed. Repeated metadata has also to be cleaned as original laboratory information is repeated in many entities while exploring the metadata tree. Finally when the metadata is clean and only the needed metadata is there, the naming of this metadata is done given the full extension of the path to follow from the root node, as an example:

Experiment__replicates__biosample__donor__organism__name = “human”

This means the name of the organism of the donor used in the biological sample of the biological replicate of the file is human. Renaming of the useful metadata has to be done to be able to keep cleaning the metadata and to give to the user a more meaningful and standardized name to potentially every metadata field.

Appendix D. XSD schema for GMQLImporter configuration file

```
<?xmlversion="1.0"?>
<xs:schemaxmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://polimi.it/GDMImporter"
xmlns="http://polimi.it/GDMImporter"
elementFormDefault="qualified">
<xs:elementname="root"type="root_type">
<xs:annotation>
<xs:documentation>Literally the root of the xml document.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:complexTypename="settings_type">
<xs:annotation>
<xs:documentation>Contains the general settings for the applica-
tion.</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:elementtype="xs:string"name="base_working_directory">
<xs:annotation>
<xs:documentation>Root working directory for the applica-
tion.</xs:documentation>
<xs:documentation>Paths inside the configuration xml are relative to this
path.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:elementtype="xs:boolean"name="download_enabled">
<xs:annotation>
<xs:documentation>Top level filter for allowing the application to down-
load.</xs:documentation>
<xs:documentation>If false, no source will be downloaded.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:elementtype="xs:boolean"name="transform_enabled">
<xs:annotation>
<xs:documentation>Top level filter for allowing the application to trans-
form.</xs:documentation>
<xs:documentation>If false, no source will be transformed.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:elementtype="xs:boolean"name="load_enabled">
<xs:annotation>
<xs:documentation>Top level filter for allowing the application to
load.</xs:documentation>
<xs:documentation>If false, no source will be loaded.</xs:documentation>
</xs:annotation>
</xs:element>
```

```
<xs:elementtype="xs:boolean"name="parallel_execution">
<xs:annotation>
<xs:documentation>decides whether the application runs with multiple threads or
not.</xs:documentation>
<xs:documentation>If false, no parallel execution is
enabled.</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexTypename="parameter_type">
<xs:annotation>
<xs:documentation>Generic parameter with description, key name and val-
ue.</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:elementtype="xs:string"name="description"minOccurs="0">
<xs:annotation>
<xs:documentation>Explanation of what means the parameter in the applica-
tion.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:elementtype="xs:string"name="type"minOccurs="0">
<xs:annotation>
<xs:documentation>type for parameter, when multiple values to use in a single
parameter.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:elementtype="xs:string"name="key">
<xs:annotation>
<xs:documentation>Name of the parameter.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:elementtype="xs:string"name="value">
<xs:annotation>
<xs:documentation>value of the parameter.</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexTypename="parameter_list_type">
<xs:annotation>
<xs:documentation>List of parameters for source/dataset.</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:elementtype="parameter_type"name="parameter"maxOccurs="unbounded"minOccurs=
"0"/>
</xs:sequence>
</xs:complexType>
<xs:simpleTypename="schema_location_type"final="restriction">
<xs:annotation>
<xs:documentation>Enumeration of possible locations for the schema
files.</xs:documentation>
```

```
</xs:annotation>
<xs:restrictionbase="xs:string">
<xs:enumerationvalue="local">
<xs:annotation>
<xs:documentation>Schema file is inside the root working directo-
ry.</xs:documentation>
</xs:annotation>
</xs:enumeration>
<xs:enumerationvalue="http">
<xs:annotation>
<xs:documentation>Access through http.</xs:documentation>
</xs:annotation>
</xs:enumeration>
</xs:restriction>
</xs:simpleType>
<xs:complexTypename="schema_type">
<xs:simpleContent>
<xs:extensionbase="xs:string">
<xs:attributetype="schema_location_type"name="location"use="required">
<xs:annotation>
<xs:documentation>URL for the schema file of the dataset.</xs:documentation>
</xs:annotation>
</xs:attribute>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
<xs:complexTypename="dataset_type">
<xs:annotation>
<xs:documentation>Represents a dataset, it belongs to a source and has internal
settings.</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:elementtype="xs:string"name="dataset_working_directory">
<xs:annotation>
<xs:documentation>
            Working directory for the dataset, subfolder of its source's work-
ing directory.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:elementtype="schema_type"name="schema_url">
<xs:annotation>
<xs:documentation>URL of the schema file.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:elementtype="xs:boolean"name="download_enabled">
<xs:annotation>
<xs:documentation>Indicates whether the dataset has to be down-
loaded.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:elementtype="xs:boolean"name="transform_enabled">
<xs:annotation>
```

```
<xs:documentation>Indicates whether the dataset has to be trans-
formed.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:elementtype="xs:boolean"name="load_enabled">
<xs:annotation>
<xs:documentation>Indicates whether the dataset has to be
loaded.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:elementtype="parameter_list_type"name="parameter_list">
<xs:annotation>
<xs:documentation>List of parameters for the dataset.</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
<xs:attributetype="xs:string"name="name"use="required">
<xs:annotation>
<xs:documentation>Name of the dataset, final name will be "source-
Name"_"datasetName".</xs:documentation>
</xs:annotation>
</xs:attribute>
</xs:complexType>
<xs:complexTypename="dataset_list_type">
<xs:annotation>
<xs:documentation>List of datasets inside a source.</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:elementtype="dataset_type"name="dataset"maxOccurs="unbounded"minOccurs="0"/
>
</xs:sequence>
</xs:complexType>
<xs:complexTypename="source_type">
<xs:annotation>
<xs:documentation>
        Represents a source, it has internal settings and contains a list of
datasets.
</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:elementtype="xs:string"name="url">
<xs:annotation>
<xs:documentation>URL address for the source.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:elementtype="xs:string"name="source_working_directory">
<xs:annotation>
<xs:documentation>
            Working directory of the source, subfolder of root working directo-
ry
</xs:documentation>
</xs:annotation>
</xs:element>
```

```
<xs:elementtype="xs:string"name="downloader">
<xs:annotation>
<xs:documentation>Indicates which downloader has to be used by the
source.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:elementtype="xs:string"name="transformer">
<xs:annotation>
<xs:documentation>Indicates which transformer has to be used by the
source.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:elementtype="xs:string"name="loader">
<xs:annotation>
<xs:documentation>Indicates which loader has to be used by the
source.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:elementtype="xs:boolean"name="download_enabled">
<xs:annotation>
<xs:documentation>
            Indicates whether the source is enabled to be downloaded, rules
over dataset and
            is ruled by root settings.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:elementtype="xs:boolean"name="transform_enabled">
<xs:annotation>
<xs:documentation>
            Indicates whether the source is enabled to be transformed, rules
over dataset and
            is ruled by root settings.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:elementtype="xs:boolean"name="load_enabled">
<xs:annotation>
<xs:documentation>
            Indicates whether the source is enabled to be loaded, rules over
dataset and is
            ruled by root settings.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:elementtype="parameter_list_type"name="parameter_list">
<xs:annotation>
<xs:documentation>List of parameters needed for the source.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:elementtype="dataset_list_type"name="dataset_list">
<xs:annotation>
<xs:documentation>List of datasets contained by the source.</xs:documentation>
```

```
</xs:annotation>
</xs:element>
</xs:sequence>
<xs:attributetype="xs:string"name="name"use="required">
<xs:annotation>
<xs:documentation>Name for which the source is going to be han-
dled.</xs:documentation>
</xs:annotation>
</xs:attribute>
</xs:complexType>
<xs:complexTypename="source_list_type">
<xs:annotation>
<xs:documentation>List of sources to be handled in the configuration
file.</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:elementtype="source_type"name="source"maxOccurs="unbounded"minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexTypename="root_type">
<xs:annotation>
<xs:documentation>Root node contains general settings and list of
sources.</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:elementtype="settings_type"name="settings"/>
<xs:elementtype="source_list_type"name="source_list"/>
</xs:sequence>
</xs:complexType>
</xs:schema>
```

Appendix E. Database design for GMQLImporter

A really important requirement for the solution is to maintain the genomic repository updated and provide historical status for every file and to know when there are new files or ones that are obsolete. A database was chosen to accomplish this task and in this appendix its model is explained. The database for GMQLImporter allows knowing the status for every file, on every dataset, on every source ever imported through GMQLImporter. Is possible to know whether a source was downloaded or transformed, also it's belonging datasets and all the files inside the dataset.

E.1.Sources, datasets and files

To know exactly the status and which files were provided by the source in a given execution, a general structure for sources, datasets and files is contained inside the GMQLImporter's database, the possibilities to know how many files are provided on the source, how many were downloaded, transformed or loaded already. The history for every file, to know when they were updated is also contained. With this implementation, is possible to know the status of the files inside the repository at any given point in time, with the correspondent details of the files at that time. GMQLImporter can give important feedback for every execution so the user can know easily the status of the repository.

E.2.Statistics

During the execution of GMQLImporter some important information is collected, such as the number of available files for download from the source, the amount of failed or replaced files, the correctly transformed ones and the also the wrong ones. In every run the total

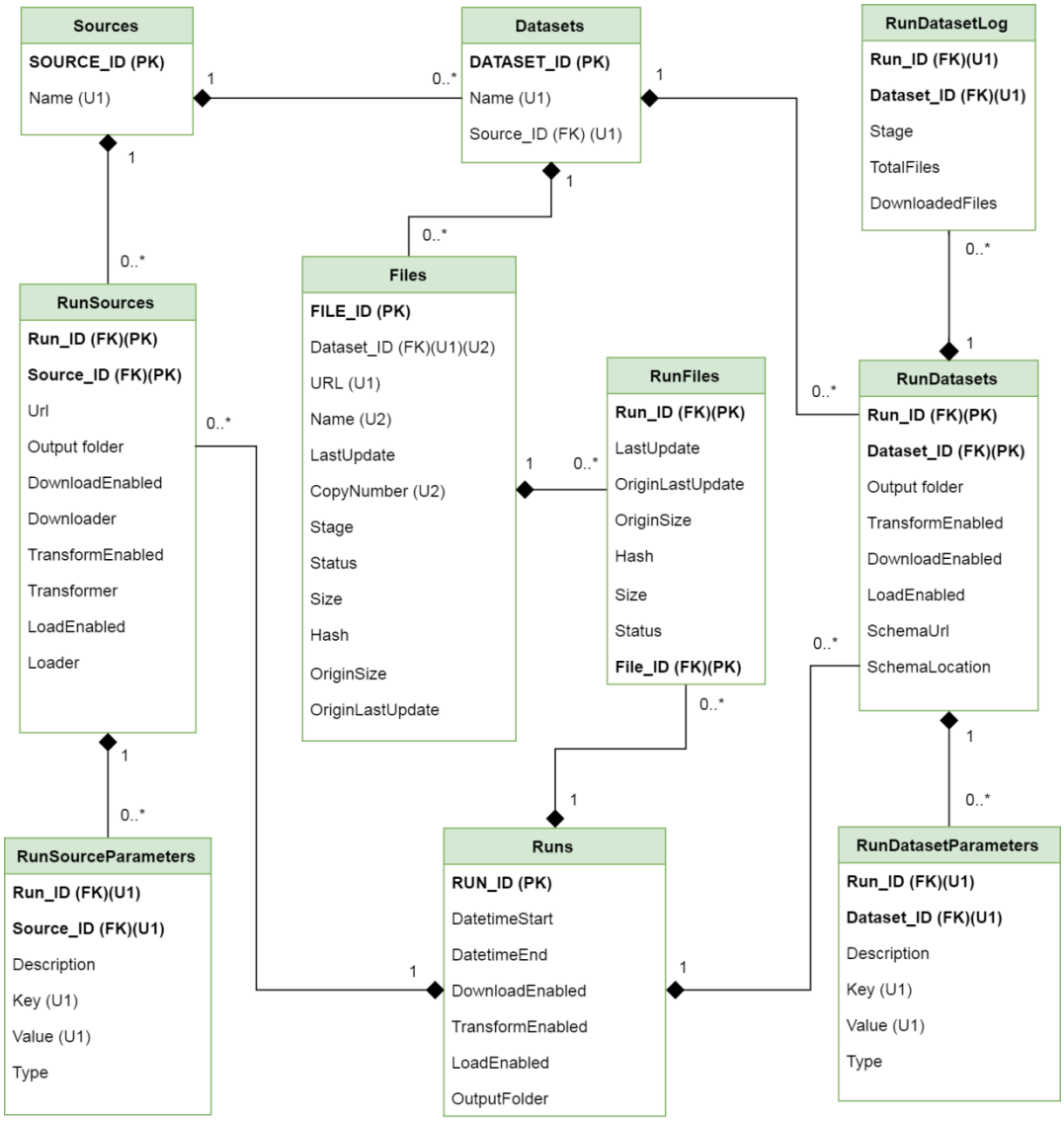
number of files available to download or transformed is stored, together with the number of correct and failed ones, the number of files that do not fit the dataset's schema and the time spent in some important tasks. GMQLImporter can provide this statistical historical data as it keeps track of them in the database.

E.3.Configuration

On every run, the configuration XML file could differ and give a different output or substantial change to the status of the repository, therefore the information provided in the configuration file is stored for every run to know what the desired actions to perform in GMQLImporter were. As the configuration XML schema, is needed to store the general settings information, the specifications for every source involved and their datasets. The XML modifications are also noticed in the sources and datasets, as their names are the only valid identifier, therefore if a name is changed, it will be considered as another source or dataset. This is used in the development of this thesis by referring to ENCODE as 2 different sources, one that provides HG19 as reference genome for the data and the other that provides GRCh38 as reference.

E.4.Database model

Here the features of the database are explained, its indices and methods. In the following diagram all the database tables are shown, the primary keys are marked with PK, foreign keys are marked with FK, and unique valid restrictions are marked with U and a number to know which attributes are linked together. Most of the IDs use surrogate keys instead of natural keys to avoid the strings comparison when handling data inside the database and natural keys are kept indexed.

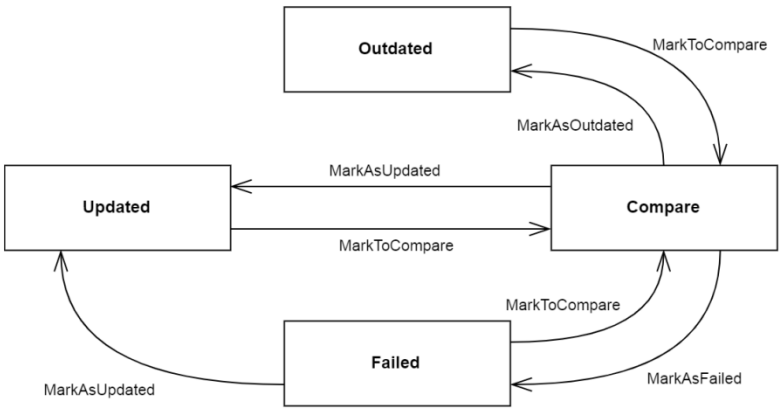


In this model, the sources with their datasets and files are represented by the tables Sources, Datasets and Files; the configurations for the execution are saved in tables Runs, RunSourceParameters and RunDatasetParameters where the configuration XML file parameters are saved distributed in the same way the XSD schema defines. Statistics are carried out using tables of RunFiles and RunDatasetLog where information is stored during runtime to let the user know the total available files to download and also failed, outdated and updated files during the execution.

To know the status of any file, 4 statuses are defined for a file:

- Updated: The file download/transformation is correctly performed in the local repository as it was in the server the last time the dataset where the file belongs was downloaded. This means the file is ready to be used in the next step of transformation or loading.
- Failed: The file download/transformation failed and the file may not be valid, in this case the file is marked as failed. This means the file is not usable for transforming or loading.
- Outdated: The file was removed from the server; this causes the file to exist locally but not remotely. This means the file is not included in the next transformation or loading step.
- Compare: Auxiliary status used to know which files do not exist anymore on the server, this status is meant to be used while the program is executing. No files are kept in this state after the program execution.

The following diagram shows the transitions of different file statuses and after every function inside is explained.



The main database methods include:

- markToCompare: by receiving a dataset, the database changes the status of every file in the dataset as to compare, this method is used to notice which files are no longer in the server side and have to be marked after as outdated in the local copy.
- markAsUpdated: indicates the file was correctly downloaded or transformed and that is ready for the next step that could be transformation or loading.
- markAsFailed: when trying to download or transform a file, if the procedure fails, the file has to be excluded from further processing and thus is marked as failed.
- markAsOutdated: once the whole dataset is downloaded or transformed this procedure allows finding the files that no longer exist in the server, and marks those files as outdated. These files are no longer used in transformation or loading procedure.
- getFileNameAndCopyNumber: as a source could have more than one file with the same name but in different folders, at the moment to put them together in the local folder this could be a problem. The database handles this when 2 or more files arrive with potentially the same name and renames them in FIFO order, adding the copy number at the end of the file name.
- checkIfUpdateFile: by receiving a file's remote data such as hash, file size or last modification date, the database checks its records to know if the file has or not to be downloaded.

The database also is configured to insert data if it does not exist or to retrieve it if it exists, as an example, when trying to create a source, it will check if the source already exists and if it exists it will access to the source already stored, on the other hand if the source does not exist, it will be created and then accessed.

Appendix F. GMQLImporter console & file logging

F.1. Introduction

GMQL-Importer provides feedback to the user both by console and file logging every time it is executed; every execution of the program is called run. This log allows the user to know the updated running status of GMQLImporter and also to search most of the operations done in the run. Also at the end of each run, it shows general statistics for the run among the different sources and datasets.

F.2. SLF4J + log4j

Simple Logging Facade for Java (SLF4J) is a Java logging API that serves as abstraction for multiple logging frameworks, such as the one used in GMQLImporter, log4j. This logger is set up at runtime and allows the communication in the console, and also straight to a file. The logging framework provides a level oriented messaging with the following levels (in decreasing priority order) ERROR, WARN, INFO, DEBUG or TRACE; ERROR is meant to communicate wrong functioning of the process, WARN is used to notice the user that something is not completely correct but the program can still work, INFO is to communicate general message feedback to the user in the application, DEBUG is used to describe the internal steps taken in the program for developers to have extra information during the program run, and TRACE is to fine grain the debugging by communicating more details to developers. The logger has the option to set up a threshold level, this allows to limit information shown to the user and to easily create multiple types of log, such as in GMQLImporter where log4j is used to show just up to INFO to the user, and DEBUG for developers.

F.3. Messages

Messages are labeled with their level (ERROR, WARN, INFO, DEBUG or TRACE), represented in a tree structure ordered by time of appearance (all possible options are shown, not all of them are mandatory) here just an explanation of what is shown in the console is given, not the exact given message in the log. The file log is set with the threshold DEBUG and the console log with INFO.

F.4.Main Program:

- [INFO] Help menu is printed if it is asked in the program arguments.
- [INFO] Path for the GMQL configuration xml file and configuration folder given in the arguments.
- [INFO] If database had to be created.
- [WARN] If no configuration folder for GMQL or no run specified when asked information for a run (see appendix J “Console manual for GMQLImporter”.
- [INFO] The configuration xml file is valid for the GMQL-Importer schema.
- [WARN] If the configuration xml file is not valid or the user cancels the run.
- [WARN] If the given metadata replacement names are repeated (see appendix K “Metadata replacement for ENCODE in GMQLImporter”)
- [INFO] Notifies when a source starts to download, transform or load.
- Download log (in section 6.3.1).
- Transform log (in section 6.3.2).
- Load log (in section 6.3.3).

- [INFO] General statistics at the end of the run, showing for each source and dataset the available files to download or transform and the correctly downloaded or transformed ones and the time taken in the processes.

F.5. Download:

F.6.FTP download

- [INFO] Connection established or retrying to connect.
- [WARN] Connection lost.
- [DEBUG] If any folder is created (usually when is the first run where no local folders exist)
- [ERROR] Could not connect to the FTP server.
- For each folder being scanned in the source:
    - [INFO] Folder's name.
    - [INFO] If the folder's name matches the regular expressions, tells is searching inside.
    - [WARN] If the folder couldn't be created.
    - [INFO] If it is trying to retry download of failed files.
    - For each file being downloaded:
        - [INFO] Current folder and file being downloaded and if the hash code matches.
        - [INFO] Current status of the file in the database, UPDATED, OUTDATED, FAILED.

        - [WARN] If the file download failed 3 times consecutively.
        - [WARN] if the internet connection was lost or couldn't access to the directory.
    - [INFO] Summary of how many total files were available from the source's folder and how many were successfully downloaded.
    - [INFO] Downloading time for the folder
- [INFO] Source download finished.
- [INFO] Downloading time for the source

F.7.ENCODE download

- [DEBUG] If any folder is created (usually when is the first run where no local folders exist)
- For each dataset to be downloaded:
    - [INFO] Name of the dataset being downloaded.
    - [INFO] Download of metadata.tsv index file.
    - [WARN] If any problem with metadata.tsv file.
    - For each file to be downloaded:
        - [INFO] Download started or finished, indicating whether it is for JSON file or region data.
        - [WARN] Connection issues or problems downloading files.
    - Dataset summary of available files to download and successfully downloaded ones.

- [INFO] Source download finished.

F.8.Transformation:

- [INFO] Starting transformation for the source.
- For each dataset in the source:
    - [INFO] Starting transformation for dataset.
    - [INFO] If the schema file was downloaded.
    - [WARN] If the schema file is not found.
    - [DEBUG] If any folder is created (usually when is the first run).
    - For each file to process:
        - [INFO] Transformation process started
        - TCGA2BED transformation (NULLTransformer):
            - [INFO] File copied, showing origin and destination paths.
            - [WARN] If the file failed to copy.
        - ENCODE transformation (ENCODETransfomer):
            - [INFO] If the file is region data and Gzipped, start unGzipping, and when finished.
            - [INFO] Start and finish times of metadata transformation.
            - [INFO] If the metadata is extracted from JSON file or metadata.tsv.
            - [WARN] If the files cannot be extracted or written.

        - [WARN] If any file does not respect the schema or could not be transformed.
        - [DEBUG] If any metadata key is modified, e.g. for normalization of separation characters, validation for java identifier.
    - Summary of modified files inside the current dataset, printing files that do not respect the schema, or if the region data or metadata were changed and transformation time for dataset.
- [INFO] Summary of modified files in the process of the source transformation, showing files that do not respect the schema, or if any region data or metadata was changed.
- [INFO] Transformation for the source finished.
- [INFO] Transformation time for the source.

F.9.Load:

- [INFO] Preparing to load the source into GMQL.
- [INFO] Each dataset to be loaded and the user to be loaded to.
- [WARN] If any region data or metadata file is missing.
- [INFO] Each dataset that finished loading the files.
- [INFO] If the dataset has no files, notify that dataset.
- [INFO] Loading time, overall and for every dataset

Appendix G. Creation of XML configuration file for GMQLImporter

This document follows the XML schema explained in chapter 5 using the XSD schema exposed in appendix D "XSD schema for GMQLImporter", complementing it with 2 sources to be imported into GMQLRepository, the first TCGA2BED (explained in appendix "TCGA2BED datasets organization") and finally ENCODE (structure defined in appendix C "ENCODE Metadata Explanation for GDM" and appendix I "ENCODE metadata generation for experiment JSON")

G.1. Root node

The root node contains the namespace for the xsd file to be used in the configuration XML, general settings for the program to run such as:

- base_working_directory: "/home/nacho/GMQL-Importer/workingDirectory" indicates the location path for the folder to be used in downloading and transforming the later specified sources.
- download_enabled: "true" sets the program to run downloads, this is a top level selection for download, if this is false: neither source nor dataset are going to be downloaded.
- transform_enabled: "true" sets the program to run transformation of downloaded datasets, this top level selection enables transformation for sources and datasets if their transform_enabled attribute is also true.
- load_enabled: "false" sets the program not to load the transformed data into GMQL repository. This top level selection blocks any source or dataset to be loaded into GMQL repository.

- parallel_execution: “true” sets the program to run in parallel when allowed inside the code, when false it will run sequentially.

G.2. Source list node

The source list contains the candidate sources to be downloaded, transformed and loaded into GMQLRepository with their basic details to get and process the datasets they provide. The sources included in the attached XML configuration file are:

G.3. HG19_TCGA

Represents the TCGA2BED source for downloading files, these files use HG19 as reference genome, its details are the following

- url: “bioinf.iasi.cnr.it” this is the base URL for connecting to the ftp server of TCGA2BED.

- source_working_directory: “HG19_TCGA” this is the local folder to be used to store all downloaded and transformed data from TCGA2BED source, this folder will be inside the base_working_directory defined in the settings in the root node.

- downloader: “it.polimi.genomics.importer.DefaultImporter.FTPDownloader” as the download for TCGA2BED is done by traversing an ftp directory recursively and matching by regular expresions, FTPDownloader is selected to perform the download process.

- transformer: “it.polimi.genomics.importer.DefaultImporter.NULLTransformer” for transformation of TCGA2BED data is needed just to copy the downloaded files, therefore NULLTransformer is needed for transformation.

- loader: “it.polimi.genomics.importer.GMQLImporter.GMQLLoader” as the datasets are meant to be loaded into GMQL repository, GMQLLoader is needed to achieve this.

- download_enabled: “true” if root download_enabled is true, this source is going to be downloaded.
- transform_enabled: “true” if root transform_enabled is true, this source is going to be transformed.
- load_enabled: “true” if root load_enabled is true, this source is going to be loaded into GMQL.
- parameter_list: here specific parameters for TCGA2BED are listed
    - metadata_replacement: “XML/metadataReplacementTcga.XML” used to change the “|” character for metadata separator for “__”.
    - gmql_user: “public” indicates which GMQL user are the files going to be added when load.
    - username: “anonymous” in the TCGA2BED FTP server the configuration is done with an anonymous user and no password.
    - password: this field is empty as no password is required.
- dataset_list for this source is explained in section 3.1.

G.4. HG19_ENCODE

Represents the ENCODE source for downloading files, in experiments using HG19 as reference genome, its details are:

- URL: “https://www.encodeproject.org/” this is the base URL for accessing ENCODE files.
- source_working_directory: “HG19_ENCODE” this is the local folder to be used to store all downloaded and transformed data from ENCODE source, this folder will be inside the base_working_directory defined in the settings in the root node.

- downloader: “it.polimi.genomics.importer.ENCODEImporter.ENCODEDownloader” as the download for ENCODE is done by specific batch download, an specific downloader is needed to get the files.
- transformer: “it.polimi.genomics.importer.ENCODEImporter.ENCODETransformer” for transformation of ENCODE a specific procedure is needed as explained in appendix C “ENCODE Metadata Explanation for GDM”.
- loader: “it.polimi.genomics.importer.GMQLImporter.GMQLLoader” as the datasets are meant to be loaded into GMQL repository, GMQLLoader is needed to achieve this.
- download_enabled: “true” if root download_enabled is true, this source is going to be downloaded.
- transform_enabled: “true” if root transform_enabled is true, this source is going to be transformed.
- load_enabled: “true” if root load_enabled is true, this source is going to be loaded into GMQL.
    - parameter_list: specific parameters for ENCODE source are the following
    - gmql_user: “public” indicates which GMQL user are the files going to be added when load.
    - metadata_prefix: “metadata/” for generation of batch download URL is needed this prefix to obtain the metadata.tsv file.
    - metadata_suffix: “/metadata.tsv” for generation of batch download URL is needed this suffix to obtain the metadata.tsv file.

- json_prefix: “experiments/” for JSON download from ENCODE is needed this prefix, as explained in “ENCODE metadata generation for experiment JSON”, the metadata root file is the experiment entity.
- json_suffix: “/?frame=embedded&format=json/” for JSON download embedded frame is used and format is JSON.
- metadata_name_separation_char: “__” specifies the double underscore as separator for metadata.
- encode_metadata_exclusion: “^contributing_files.*$” this category is excluded as it was found as not useful metadata.
- encode_metadata_exclusion: “^original_files.*$” this category is excluded as it was found as not useful metadata.
- encode_metadata_exclusion: “^.*analysis_step_version.*$” this category is excluded as it was found as not useful metadata.
- encode_metadata_exclusion: “^.*derived_from.*$” this category is excluded as it was found as not useful metadata.
- encode_metadata_exclusion: “^.*revoked_files.*$” this category is excluded as it was found as not useful metadata.
- encode_metadata_exclusion: “^assembly$” this category is excluded as it was found as not useful metadata.
- metadata_replacement: “XML/metadataReplacement.XML” reference to an XML file containing the replacements for metadata names.
- metadata_extraction: “tsv” for Encode, json or tsv extraction method for metadata can be specified.

  - assembly_exclude: “GRCh38” as some files with GRCh38 reference genome are passed by Encode, this parameter is used to filter them out.

- dataset_list for this source is explained in section 3.2.

G.5. GRCh38_ENCODE

Represents the ENCODE source for downloading files, in experiments using GRCh38 as reference genome, its details are:

- url: “https://www.encodeproject.org/” this is the base URL for accessing ENCODE files.

- source_working_directory: “GRCh38_ENCODE” this is the local folder to be used to store all downloaded and transformed data from ENCODE source, this folder will be inside the base_working_directory defined in the settings in the root node.

- downloader: “it.polimi.genomics.importer.ENCODEImporter.ENCODEDownloader” as the download for ENCODE is done by specific batch download, a specific downloader is needed to get the files.

- transformer: “it.polimi.genomics.importer.ENCODEImporter.ENCODETransformer” for transformation of ENCODE an specific procedure is needed as explained in “ENCODE Metadata Explanation for GDM”.

- loader: “it.polimi.genomics.importer.GMQLImporter.GMQLLoader” as the datasets are meant to be loaded into GMQL repository, GMQLLoader is needed to achieve this.

- download_enabled: “true” if root download_enabled is true, this source is going to be downloaded.

- transform_enabled: “true” if root transform_enabled is true, this source is going to be transformed.

- load_enabled: “true” if root load_enabled is true, this source is going to be loaded into GMQL.
- parameter_list: specific parameters for ENCODE source are the following
    - gmql_user: “public” indicates which GMQL user are the files going to be added when load.
    - metadata_prefix: “metadata/” for generation of batch download URL is needed this prefix to obtain the metadata.tsv file.
    - metadata_suffix: “/metadata.tsv” for generation of batch download URL is needed this suffix to obtain the metadata.tsv file.
    - json_prefix: “experiments/” for JSON download from ENCODE is needed this prefix, as explained in appendix I“ENCODE metadata generation for experiment JSON”, the metadata root file is the experiment entity.
    - json_suffix: “/?frame=embedded&format=json/” for JSON download embedded frame is used and format is JSON.
    - metadata_name_separation_char: “__” specifies the double underscore as separator for metadata.
    - encode_metadata_exclusion: “^contributing_files.*$” this category is excluded as it was found as not useful metadata.
    - encode_metadata_exclusion: “^original_files.*$” this category is excluded as it was found as not useful metadata.
    - encode_metadata_exclusion: “^.*analysis_step_version.*$” this category is excluded as it was found as not useful metadata.
    - encode_metadata_exclusion: “^.*derived_from.*$” this category is excluded as it was found as not useful metadata.

- encode_metadata_exclusion: “^.*revoked_files.*$” this category is excluded as it was found as not useful metadata.
- encode_metadata_exclusion: “^assembly$” this category is excluded as it was found as not useful metadata.
- metadata_replacement: “XML/metadataReplacement.XML” references to an XML file containing the replacements for metadata names.
- metadata_extraction: “tsv” for ENCODE, json or tsv extraction method for metadata can be specified.
- assembly_exclude: “hg19” as some files with hg19 reference genome are passed by ENCODE, this parameter is used to filter them out.

- dataset_list for this source is explained in section 3.3.

G.6. Datasets by source

G.7. HG19_TCGA datasets

TCGA2BED dataset generation is explained in “Preparation of configuration.XML file for downloading TCGA2BED datasets” its implementation is the following:

G.8. cnv

Contains cnv experiments in GDM format for region data and metadata.

- dataset_working_directory: “cnv” this is the local folder to store downloaded and transformed files for the dataset, this folder will be under its source folder.
- schema_url: “location=http” means the schema file has to be downloaded with http method, “ftp://bioinf.iasi.cnr.it/bed/acc/cnv/header.schema” indicates the full URL to download the schema file.

- download_enabled: “true” lowest level activator for download, it will only be downloaded if its source and root nodes are active.
- transform_enabled: “true” lowest level activator for transformation, it will only be transformed if its source and root nodes have transform_enabled set to true.
- load_enabled: “true” lowest level activator for load, it will only be loaded if root and source nodes are active also.
- parameter_list: specific parameters for cnv dataset
    - folder_regex: “^/bed/.*/cnv” is the regular expression to access all cnv folders in TCGA2BED FTP server.
    - files_regex: “.*\.bed(\.meta)?$” regular expression to download .bed and .bed.meta from every cnv folder.
    - md5_checksum_tcga2bed: “md5table.txt” this file on each cnv folder has details for md5 hash of all files in the folder.
    - exp_info_tcga2bed: “exp_info.tsv” this file contains the number of files inside cnv folder to download.

G.9. dnamethylation

Contains dnamethylation (27 and 450) experiments in GDM format for region data and metadata.

- dataset_working_directory: “dnamethylation” this is the local folder to store downloaded and transformed files for the dataset, this folder will be under its source folder.

- schema_url: “location=http” means the schema file has to be downloaded with http method, “ftp://bioinf.iasi.cnr.it/bed/lusc/dnamethylation27/header.schema” indicates the full URL to download the schema file.
- download_enabled: “true” lowest level activator for download, it will only be downloaded if its source and root nodes are active.
- transform_enabled: “true” lowest level activator for transformation, it will only be transformed if its source and root nodes have transform_enabled set to true.
- load_enabled: “true” lowest level activator for load, it will only be loaded if root and source nodes are active also.
- parameter_list: specific parameters for dnamethylation dataset
    - folder_regex: “^/bed/.*/dnamethylation.*” is the regular expression to access all dnamethylation folders in TCGA2BED FTP server.
    - files_regex: “.*\.bed(\.meta)?$” regular expression to download .bed and .bed.meta from every dnamethylation folder.
    - md5_checksum_tcga2bed: “md5table.txt” this file on each dnamethylation folder has details for md5 hash of all files in the folder.
    - exp_info_tcga2bed: “exp_info.tsv” this file contains the number of files inside dnamethylation folder to download.

G.10. dnaseq

Contains dnaseq experiments in GDM format for region data and metadata.

- dataset_working_directory: “dnaseq” this is the local folder to store downloaded and transformed files for the dataset, this folder will be under its source folder.

- schema_url: “location=http” means the schema file has to be downloaded with http method, “ftp://bioinf.iasi.cnr.it/bed/lusc/dnaseq/header.schema” indicates the full URL to download the schema file.
- download_enabled: “true” lowest level activator for download, it will only be downloaded if its source and root nodes are active.
- transform_enabled: “true” lowest level activator for transformation, it will only be transformed if its source and root nodes have transform_enabled set to true.
- load_enabled: “true” lowest level activator for load, it will only be loaded if root and source nodes are active also.
- parameter_list: specific parameters for dnaseq dataset
    - folder_regex: “^/bed/.*/dnaseq” is the regular expression to access all dnaseq folders in TCGA2BED FTP server.
    - files_regex: “.*\.bed(\.meta)?$” regular expression to download .bed and .bed.meta from every dnaseq folder.
    - md5_checksum_tcga2bed: “md5table.txt” this file on each dnaseq folder has details for md5 hash of all files in the folder.
    - exp_info_tcga2bed: “exp_info.tsv” this file contains the number of files inside dnaseq folder to download.

G.11. mirnaseq_isoform

Contains mirnaseq_isoform experiments in GDM format for region data and metadata.

- dataset_working_directory: “mirnaseq_isoform” this is the local folder to store downloaded and transformed files for the dataset, this folder will be under its source folder.

- schema_url: “location=http” means the schema file has to be downloaded with http method, “ftp://bioinf.iasi.cnr.it/bed/lusc/mirnaseq/isoform.quantification/header.schema” indicates the full URL to download the schema file.
- download_enabled: “true” lowest level activator for download, it will only be downloaded if its source and root nodes are active.
- transform_enabled: “true” lowest level activator for transformation, it will only be transformed if its source and root nodes have transform_enabled set to true.
- load_enabled: “true” lowest level activator for load, it will only be loaded if root and source nodes are active also.
- parameter_list: specific parameters for mirnaseq_isoform dataset
    - folder_regex: “^/bed/.*/mirnaseq/isoform.quantification” is the regular expression to access all mirnaseq_isoform folders in TCGA2BED FTP server.
    - files_regex: “.*\.bed(\.meta)?$” regular expression to download .bed and .bed.meta from every mirnaseq_isoform folder.
    - md5_checksum_tcga2bed: “md5table.txt” this file on each mirnaseq_isoform folder has details for md5 hash of all files in the folder.
    - exp_info_tcga2bed: “exp_info.tsv” this file contains the number of files inside mirnaseq_isoform folder to download.

G.12. mirnaseq_mirna

Contains mirnaseq_mirna experiments in GDM format for region data and metadata.

- dataset_working_directory: “mirnaseq_mirna” this is the local folder to store downloaded and transformed files for the dataset, this folder will be under its source folder.
- schema_url: “location=http” means the schema file has to be downloaded with http method, “ftp://bioinf.iasi.cnr.it/bed/lusc/mirnaseq/mirna.quantification/header.schema” indicates the full URL to download the schema file.
- download_enabled: “true” lowest level activator for download, it will only be downloaded if its source and root nodes are active.
- transform_enabled: “true” lowest level activator for transformation, it will only be transformed if its source and root nodes have transform_enabled set to true.
- load_enabled: “true” lowest level activator for load, it will only be loaded if root and source nodes are active also.
- parameter_list: specific parameters for mirnaseq_mirna dataset
    - folder_regex: “^/bed/.*/mirnaseq/mirna.quantification” is the regular expression to access all mirnaseq_mirna folders in TCGA2BED FTP server.
    - files_regex: “.*\.bed(\.meta)?$” regular expression to download .bed and .bed.meta from every mirnaseq_mirna folder.
    - md5_checksum_tcga2bed: “md5table.txt” this file on each mirnaseq_mirna folder has details for md5 hash of all files in the folder.
    - exp_info_tcga2bed: “exp_info.tsv” this file contains the number of files inside mirnaseq_mirna folder to download.

G.13. rnaseq_exon

Contains rnaseq_exon experiments in GDM format for region data and metadata.

- dataset_working_directory: “rnaseq_exon” this is the local folder to store downloaded and transformed files for the dataset, this folder will be under its source folder.
- schema_url: “location=http” means the schema file has to be downloaded with http method, “ftp://bioinf.iasi.cnr.it/bed/lusc/rnaseq/exon.quantification/header.schema” indicates the full URL to download the schema file.
- download_enabled: “true” lowest level activator for download, it will only be downloaded if its source and root nodes are active.
- transform_enabled: “true” lowest level activator for transformation, it will only be transformed if its source and root nodes have transform_enabled set to true.
- load_enabled: “true” lowest level activator for load, it will only be loaded if root and source nodes are active also.
- parameter_list: specific parameters for rnaseq_exon dataset
    - folder_regex: “^/bed/.*/rnaseq/exon.quantification” is the regular expression to access all rnaseq_exon folders in TCGA2BED FTP server.
    - files_regex: “.*\.bed(\.meta)?$” regular expression to download .bed and .bed.meta from every rnaseq_exon folder.
    - md5_checksum_tcga2bed: “md5table.txt” this file on each rnaseq_exon folder has details for md5 hash of all files in the folder.
    - exp_info_tcga2bed: “exp_info.tsv” this file contains the number of files inside rnaseq_exon folder to download.

G.14. rnaseq_gene

Contains rnaseq_gene experiments in GDM format for region data and metadata.

- dataset_working_directory: “rnaseq_gene” this is the local folder to store downloaded and transformed files for the dataset, this folder will be under its source folder.
- schema_url: “location=http” means the schema file has to be downloaded with http method, “ftp://bioinf.iasi.cnr.it/bed/lusc/rnaseq/gene.quantification/header.schema” indicates the full URL to download the schema file.
- download_enabled: “true” lowest level activator for download, it will only be downloaded if its source and root nodes are active.
- transform_enabled: “true” lowest level activator for transformation, it will only be transformed if its source and root nodes have transform_enabled set to true.
- load_enabled: “true” lowest level activator for load, it will only be loaded if root and source nodes are active also.
- parameter_list: specific parameters for rnaseq_gene dataset
    - folder_regex: “^/bed/.*/rnaseq/gene.quantification” is the regular expression to access all rnaseq_gene folders in TCGA2BED FTP server.
    - files_regex: “.*\.bed(\.meta)?$” regular expression to download .bed and .bed.meta from every rnaseq_gene folder.
    - md5_checksum_tcga2bed: “md5table.txt” this file on each rnaseq_gene folder has details for md5 hash of all files in the folder.

- exp_info_tcga2bed: “exp_info.tsv” this file contains the number of files inside rnaseq_gene folder to download.

G.15. rnaseq_spljxn

Contains rnaseq_spljxn experiments in GDM format for region data and metadata.

- dataset_working_directory: “rnaseq_spljxn” this is the local folder to store downloaded and transformed files for the dataset, this folder will be under its source folder.
- schema_url: “location=http” means the schema file has to be downloaded with http method, “ftp://bioinf.iasi.cnr.it/bed/lusc/rnaseq/spljxn.quantification/header.schema” indicates the full URL to download the schema file.
- download_enabled: “true” lowest level activator for download, it will only be downloaded if its source and root nodes are active.
- transform_enabled: “true” lowest level activator for transformation, it will only be transformed if its source and root nodes have transform_enabled set to true.
- load_enabled: “true” lowest level activator for load, it will only be loaded if root and source nodes are active also.
- parameter_list: specific parameters for rnaseq_spljxn dataset
    - folder_regex: “^/bed/.*/rnaseq/spljxn.quantification” is the regular expression to access all rnaseq_spljxn folders in TCGA2BED FTP server.
    - files_regex: “.*\.bed(\.meta)?$” regular expression to download .bed and .bed.meta from every rnaseq_spljxn folder.

- md5_checksum_tcga2bed: “md5table.txt” this file on each rnaseq_spljxn folder has details for md5 hash of all files in the folder.
- exp_info_tcga2bed: “exp_info.tsv” this file contains the number of files inside rnaseq_spljxn folder to download.

G.16. rnaseqv2_exon

Contains rnaseqv2_exon experiments in GDM format for region data and metadata.

- dataset_working_directory: “rnaseqv2_exon” this is the local folder to store downloaded and transformed files for the dataset, this folder will be under its source folder.
- schema_url: “location=http” means the schema file has to be downloaded with http method, “ftp://bioinf.iasi.cnr.it/bed/lusc/rnaseqv2/exon.quantification/header.schema” indicates the full URL to download the schema file.
- download_enabled: “true” lowest level activator for download, it will only be downloaded if its source and root nodes are active.
- transform_enabled: “true” lowest level activator for transformation, it will only be transformed if its source and root nodes have transform_enabled set to true.
- load_enabled: “true” lowest level activator for load, it will only be loaded if root and source nodes are active also.
- parameter_list: specific parameters for rnaseqv2_exon dataset
    - folder_regex: “^/bed/.*/rnaseqv2/exon.quantification” is the regular expression to access all rnaseqv2_exon folders in TCGA2BED FTP server.

- files_regex: ".*\.bed(\.meta)?$" regular expression to download .bed and .bed.meta from every rnaseqv2_exon folder.
- md5_checksum_tcga2bed: "md5table.txt" this file on each rnaseqv2_exon folder has details for md5 hash of all files in the folder.
- exp_info_tcga2bed: "exp_info.tsv" this file contains the number of files inside rnaseqv2_exon folder to download.

G.17. rnaseqv2_gene

Contains rnaseqv2_gene experiments in GDM format for region data and metadata.

- dataset_working_directory: "rnaseqv2_gene" this is the local folder to store downloaded and transformed files for the dataset, this folder will be under its source folder.
- schema_url: "location=http" means the schema file has to be downloaded with http method, "ftp://bioinf.iasi.cnr.it/bed/lusc/rnaseqv2/gene.quantification/header.schema" indicates the full URL to download the schema file.
- download_enabled: "true" lowest level activator for download, it will only be downloaded if its source and root nodes are active.
- transform_enabled: "true" lowest level activator for transformation, it will only be transformed if its source and root nodes have transform_enabled set to true.
- load_enabled: "true" lowest level activator for load, it will only be loaded if root and source nodes are active also.
- parameter_list: specific parameters for rnaseqv2_gene dataset

- folder_regex: “^/bed/.*/rnaseqv2/gene.quantification” is the regular expression to access all rnaseqv2_gene folders in TCGA2BED FTP server.
- files_regex: “.*\.bed(\.meta)?$” regular expression to download .bed and .bed.meta from every rnaseqv2_gene folder.
- md5_checksum_tcga2bed: “md5table.txt” this file on each rnaseqv2_gene folder has details for md5 hash of all files in the folder.
- exp_info_tcga2bed: “exp_info.tsv” this file contains the number of files inside rnaseqv2_gene folder to download.

G.18. rnaseqv2_isoform

Contains rnaseqv2_isoform experiments in GDM format for region data and metadata.

- dataset_working_directory: “rnaseqv2_isoform” this is the local folder to store downloaded and transformed files for the dataset, this folder will be under its source folder.
- schema_url: “location=http” means the schema file has to be downloaded with http method, “ftp://bioinf.iasi.cnr.it/bed/lusc/rnaseqv2/isoform.quantification/header.schema” indicates the full URL to download the schema file.
- download_enabled: “true” lowest level activator for download; it will only be downloaded if its source and root nodes are active.
- transform_enabled: “true” lowest level activator for transformation, it will only be transformed if its source and root nodes have transform_enabled set to true.
- load_enabled: “true” lowest level activator for load, it will only be loaded if root and source nodes are active also.

- parameter_list: specific parameters for rnaseqv2_isoform dataset
    - folder_regex: “^/bed/.*/rnaseqv2/isoform.quantification” is the regular expression to access all rnaseqv2_isoform folders in TCGA2BED FTP server.
    - files_regex: “.*\.bed(\.meta)?$” regular expression to download .bed and .bed.meta from every rnaseqv2_isoform folder.
    - md5_checksum_tcga2bed: “md5table.txt” this file on each rnaseqv2_isoform folder has details for md5 hash of all files in the folder.
    - exp_info_tcga2bed: “exp_info.tsv” this file contains the number of files inside rnaseqv2_isoform folder to download.

G.19. rnaseqv2_spljxn

Contains rnaseqv2_spljxn experiments in GDM format for region data and metadata.

- dataset_working_directory: “rnaseqv2_spljxn” this is the local folder to store downloaded and transformed files for the dataset, this folder will be under its source folder.
- schema_url: “location=http” means the schema file has to be downloaded with http method, “ftp://bioinf.iasi.cnr.it/bed/lusc/rnaseqv2/spljxn.quantification/header.schema” indicates the full URL to download the schema file.
- download_enabled: “true” lowest level activator for download; it will only be downloaded if its source and root nodes are active.
- transform_enabled: “true” lowest level activator for transformation, it will only be transformed if its source and root nodes have transform_enabled set to true.

- load_enabled: “true” lowest level activator for load, it will only be loaded if root and source nodes are active also.
- parameter_list: specific parameters for rnaseqv2_spljxn dataset
    - folder_regex: “^/bed/.*/rnaseqv2/spljxn.quantification” is the regular expression to access all rnaseqv2_spljxn folders in TCGA2BED FTP server.
    - files_regex: “.*\.bed(\.meta)?$” regular expression to download .bed and .bed.meta from every rnaseqv2_spljxn folder.
    - md5_checksum_tcga2bed: “md5table.txt” this file on each rnaseqv2_spljxn folder has details for md5 hash of all files in the folder.
    - exp_info_tcga2bed: “exp_info.tsv” this file contains the number of files inside rnaseqv2_spljxn folder to download.

G.20. HG19_ENCODE datasets

Datasets for HG19_ENCODE include broadPeak and narrowPeak.

G.21. broadPeak

Contains broadPeak data for HG19 reference genome stored in the ENCODE project

- dataset_working_directory: “broadPeak” this is the local folder to store downloaded and transformed files for the dataset, this folder will be under its source folder.
- schema_url: “location=local” means the schema file is stored in local media relative to root node base_working_directory; “schemas/broadPeak.schema” indicates the relative path for the broadPeak schema.
- download_enabled: “true” lowest level activator for download, it will only be downloaded if its source and root nodes are active.

- transform_enabled: “true” lowest level activator for transformation, it will only be transformed if its source and root nodes have transform_enabled set to true.
- load_enabled: “true” lowest level activator for load, it will only be loaded if root and source nodes are active also.
- parameter_list: specific parameters for HG19_ENCODE broadPeak dataset
    - loading_name: “HG19_ENCODE_BROAD_MAY_2017” indicates the name that will be given to the dataset in the loading process. By default it joins source name and dataset name with an underscore between them.
    - type: “Experiment” used in the creation of the batch download URL, defines the experiment entity as the base to search for.
    - files.file_type: “bed+broadPeak” used in the creation of the batch download URL, filters experiments with bed broadPeak files.
    - award.project: “ENCODE” used in the creation of the batch download URL, defines only the files from project ENCODE to be downloaded.
    - files.assembly: “hg19” used in the creation of the batch download URL, defines only hg19 assembly files to be downloaded.

G.22. narrowPeak

Contains narrowPeak data for HG19 reference genome stored in the ENCODE project

- dataset_working_directory: “narrowPeak” this is the local folder to store downloaded and transformed files for the dataset, this folder will be under its source folder.

- schema_url: “location=local” means the schema file is stored in local media relative to root node base_working_directory; “schemas/narrowPeak.schema” indicates the relative path for the broadPeak schema.
- download_enabled: “true” lowest level activator for download; it will only be downloaded if its source and root nodes are active.
- transform_enabled: “true” lowest level activator for transformation, it will only be transformed if its source and root nodes have transform_enabled set to true.
- load_enabled: “true” lowest level activator for load, it will only be loaded if root and source nodes are active also.
- parameter_list: specific parameters for HG19_ENCODE narrowPeak dataset
    - loading_name: “HG19_ENCODE_NARROW_MAY_2017” indicates the name that will be given to the dataset in the loading process. By default it joins source name and dataset name with an underscore between them.
    - type: “Experiment” used in the creation of the batch download URL, defines the experiment entity as the base to search for.
    - files.file_type: “bed+narrowPeak” used in the creation of the batch download URL, filters experiments with bed narrowPeak files.
    - award.project: “ENCODE” used in the creation of the batch download URL, defines only the files from project ENCODE to be downloaded.
    - files.assembly: “hg19” used in the creation of the batch download URL, defines only hg19 assembly files to be downloaded.

G.23. GRCh38_ENCODE datasets

Datasets for GRCh38_ENCODE include broadPeak and narrowPeak.

G.24. broadPeak

Contains broadPeak data for GRCh38 reference genome stored in the ENCODE project

- dataset_working_directory: “broadPeak” this is the local folder to store downloaded and transformed files for the dataset, this folder will be under its source folder.
- schema_url: “location=local” means the schema file is stored in local media relative to root node base_working_directory; “schemas/broadPeak.schema” indicates the relative path for the broadPeak schema.
- download_enabled: “true” lowest level activator for download; it will only be downloaded if its source and root nodes are active.
- transform_enabled: “true” lowest level activator for transformation, it will only be transformed if its source and root nodes have transform_enabled set to true.
- load_enabled: “true” lowest level activator for load, it will only be loaded if root and source nodes are active also.
- parameter_list: specific parameters for GRCh38_ENCODE broadPeak dataset
    - loading_name: “GRCh38_ENCODE_BROAD_MAY_2017” indicates the name that will be given to the dataset in the loading process. By default it joins source name and dataset name with an underscore between them.
    - type: “Experiment” used in the creation of the batch download URL, defines the experiment entity as the base to search for.
    - files.file_type: “bed+broadPeak” used in the creation of the batch download URL, filters experiments with bed broadPeak files.
    - award.project: “ENCODE” used in the creation of the batch download URL, defines only the files from project ENCODE to be downloaded.

- files.assembly: “GRCh38” used in the creation of the batch download URL, defines only GRCh38 assembly files to be downloaded.

G.25. narrowPeak

Contains narrowPeak data for GRCh38 reference genome stored in the ENCODE project

- dataset_working_directory: “narrowPeak” this is the local folder to store downloaded and transformed files for the dataset, this folder will be under its source folder.
- schema_url: “location=local” means the schema file is stored in local media relative to root node base_working_directory; “schemas/narrowPeak.schema” indicates the relative path for the broadPeak schema.
- download_enabled: “true” lowest level activator for download; it will only be downloaded if its source and root nodes are active.
- transform_enabled: “true” lowest level activator for transformation, it will only be transformed if its source and root nodes have transform_enabled set to true.
- load_enabled: “true” lowest level activator for load, it will only be loaded if root and source nodes are active also.
- parameter_list: specific parameters for GRCh38_ENCODE narrowPeak dataset
    - loading_name: “GRCh38_ENCODE_NARROW_MAY_2017” indicates the name that will be given to the dataset in the loading process. By default it joins source name and dataset name with an underscore between them.
    - type: “Experiment” used in the creation of the batch download URL, defines the experiment entity as the base to search for.

- files.file_type: “bed+narrowPeak” used in the creation of the batch download URL, filters experiments with bed narrowPeak files.
- award.project: “ENCODE” used in the creation of the batch download URL, defines only the files from project ENCODE to be downloaded.
- files.assembly: “GRCh38” used in the creation of the batch download URL, defines only GRCh38 assembly files to be downloaded.

Appendix H. TCGA2BED datasets organization

Folders available inside TCGA2BED ftp (ftp://bioinf.iasi.cnr.it/) in bed format are:

Where inside any one of these folders we find the following structure:

Also inside the mirnaseq folder the inner structure is:

Indice di /bed/*/mirnaseq/

Nome
[directory principale]
isoform.quantification/
mirna.quantification/

Order for the rnaseq folder:

Indice di /bed/*/rnaseq/

Nome
[directory principale]
exon.quantification/
gene.quantification/
spljxn.quantification/

And for the rnaseqv2 folder follows this order:

Indice di /bed/*/rnaseqv2/

Nome
[directory principale]
exon.quantification/
gene.quantification/
isoform.quantification/
spljxn.quantification/

Inside any folder in the following list (* meaning any subfolder):

- /bed/*/cnv/
- /bed/*/dnamethylation27/
- /bed/*/dnamethylation450/
- /bed/*/dnaseq
- /bed/*/mirnaseq/isoform.quantification/
- /bed/*/mirnaseq/mirna.quantification/
- /bed/*/rnaseqv2/exon.quantification/
- /bed/*/rnaseqv2/gene.quantification/
- /bed/*/rnaseqv2/isoform.quantification/
- /bed/*/rnaseqv2/spljxn.quantification/

Every folder inside contains:

- exp_info.tsv: contains nº of experiment files in the folder.
- header.schema: region data file schema.
- md5table.txt: contains md5 checksum for every file inside the folder.
- File1.bed (data file)
- File1.bed.meta (metadata file)
- (...more files .bed and .bed.meta)

Appendix I. ENCODE metadata generation for experiment JSON

For the ENCODE project, the metadata of every experiment is distributed in a tree structure of items (item is the base object for every entity in ENCODE, so any object inherits from it), the root node item is the experiment. Experiment inherits from dataset that is another item in the ENCODE project and also implements other properties. Metadata generation is done in 3 sections, first section is ENCODE's"Shared calculated properties[1] and mixin properties[2]", this section explains which general attributes (mixin and shared calculated properties are used among many items in ENCODE) are contained inside every experiment. Second section is dataset specific properties that contain the experiment. Third section comprehends the experiment specific information. The 4th section in this appendix summarizes the usage of ENCODE JSON metadata and its possibilities. The final section explains the substructures used in the first 3 sections described in this appendix.

I.1. Shared Calculated and Mixin Properties

These properties are common ENCODE utilities used by many items, 2 files are used to explain this in the ENCODE documentation: shared_calculated_properties.py and mixins.json. And the information that could be added by these utilities is the following:

- submitted_by: id of the user who created the item, is auto assigned by the server and it is also a link to the same user.
- date_created: date and time for the creation of the item, is auto assigned by the server.

[1] https://github.com/ENCODE-DCC/encoded/blob/master/src/encoded/types/shared_calculated_properties.py
[2] https://github.com/ENCODE-DCC/encoded/blob/master/src/encoded/schemas/mixins.json

- submitter_comment: additional information specified by the submitter to be displayed as a comment on the portal.
- biosample_term_id: ontology id describing the biosample.
- biosample_term_name: ontology term describing biosample.
- biosample_synonyms: set of other terms in the ontology that describe the biosample.
- biosample_type: categorization of the biosample, is a value from the following enumeration ("primary cell", "immortalized cell line", "tissue", "in vitro differentiated cells", "induced pluripotent stem cell line", "whole organisms", "stem cell").
- developmental_slims: set of ontology terms referring to the category “developmental” associated to the biosample_term_id.
- organ_slims: set of terms in the ontology that refer to the organ associated with the biosample_term_id.
- accession: unique identifier to be used to reference the object.
- alternate_accessions: set of accessions previously assigned to objects that have been merged with this object.
- accessioned_status: status of the accession, value in the enum “in progress”, “deleted”, “replaced”, “released”, “revoked”.
- notes: DCC (Data Coordinating Center) internal notes.
- lab: laboratory associated with the submission.
- award: grant associated with the submission.

- uuid: universally unique identifier (UUID) is a 128-bit number used to identify information in computer systems.
- references: publications that provide more information about the object. It is an set of links to the references.
- system_slims: preferred name on the ontology for the system which the biosample_term_id belongs.
- schema_version: the version of the JSON schema that the server uses to validate the object.
- aliases: set of lab specific identifiers to reference an object. Current convention is colon separated lab name and lab identifier.
- assay_term_name: OBI (Ontology for Biomedical Investigations) ontology term for the assay. It's possible values are in the following enumeration: "ChIP-seq", RNA-seq", DNase-seq", "eCLIP", "shRNA knockdown followed by RNA-seq", "RNA Bind-n-Seq", "transcription profiling by array assay", "DNA methylation profiling by array assay", "whole-genome shotgun bisulfite sequencing", "RRBS", "siRNA knockdown followed by RNA-seq", "RAMPAGE", "comparative genomic hybridization by array", "CAGE", "single cell isolation followed by RNA-seq", "Repli-seq", "microRNA-seq", "microRNA counts", "MRE-seq", "RIP-seq", "Repli-chip", "MeDIP-seq", "ChIA-PET", "FAIRE-seq", "ATAC-seq", "PAS-seq", "RIP-chip", "RNA-PET", "genotyping by high throughput sequencing assay", "CRISPR genome editing followed by RNA-seq", "protein sequencing by tandem mass spectrometry assay", "5C", "HiC", "TAB-seq", "iCLIP", "DNA-PET", "Switchgear", "5' RLM RACE", "MNase-seq", "5' RACE", "3' RACE", "small RNA-seq", "Bru-seq", "BruChase-seq", "genetic modification followed by DNase-seq", "CRISPRi followed by RNA-seq"

I.2. Dataset Properties

Properties of dataset item in ENCODE. The metadata generation is done mainly with 2 files: the schema dataset.json[3] and the python script dataset.py[4]. The information that could be added by this section are (shared calculated and mixin properties are not shown in the list):

- description: plain text description of the dataset.
- dbxrefs: set of unique identifiers from external resources.
- internal_tags: some datasets are part of particular data collections. Those collections are the following tags: “DREAM”, “ENCORE”, “ENTEx”, “SESCC”, “dbGaP”, “ENCYCLOPEDIAv3”, “ENCYCLOPEDIAv4”, “cre_inputv10”, “cre_inputv11”. This property is a set.
- status: State of the accession, value in the enum “proposed”, “started”, “submitted”, “ready for review”, “deleted”, “released”, “revoked”, “archived”, “replaced”.
- date_released: autogenerated date and time assigned when the object is released.
- contributing_files: set of files used to create the dataset. Includes “released” and excludes “revoked”, “deleted” and “replaced” files.
- original_files: set of files used to create the dataset. Includes every file status.
- hub: section of the lab where the dataset was created.
- annotations: set of annotation files produced by ENCODE.
- superseded_by: set of datasets that replace this dataset.
- assembly: GRC genome assembly to which the target coordinates relate. E.g. GRCh38. It is an set created from all the assemblies referred in every file in the dataset’s files.

[3] https://github.com/ENCODE-DCC/encoded/blob/master/src/encoded/schemas/dataset.json
[4] https://github.com/ENCODE-DCC/encoded/blob/master/src/encoded/types/dataset.py

- month_released: calculated month extracted from date_released.

When embedded JSON is requested extra metadata is added, this dataset properties include:

(Dots indicate the right term is contained inside the left one and plurals denote set of items)

- files: set of files used to create the dataset. Includes “released” and “archived” files and excludes “revoked”, “deleted” and “replaced”. For every file, also contains the following properties:
    - replicate
    - replicate.experiment
    - replicate.experiment.lab
    - replicate.experiment.target
    - submitted_by
    - lab
- revoked_files: set of files used in the creation of the dataset but have been replaced. Includes “revoked” only files. For every file also contains the following properties:
    - replicate
    - replicate.experiment
    - replicate.experiment.lab
    - replicate.experiment.target
    - submitted_by

  - lab

- award.pi.lab: lab associated with the principle Investigator (pi) of the grant (award).
- lab: lab where the experiment comes from.
- documents.lab: lab where each document used in the experiment come from.
- documents.award: grant associated to each document used in the dataset.
- documents.submitted_by: user who submitted each document.
- references: publications that provide more information about the dataset.

I.3. Experiment Properties

Properties of experiment item in ENCODE. The metadata generation is done mainly by 2 files: the schema experiment.json[5] and the python script experiment.py[6]. An experiment is a special case of dataset. It includes assay metadata, replicate information and data files. It could add the following properties (shared calculated, mixin and dataset properties are not shown):

- target: for assays, such as ChIP-seq or RIP-seq, the name of the gene whose expression or product is under investigation for the experiment.
- biosample_type: categorization of the biosample.
- documents: protocols or other documents that describe the assay or the results.
- supersedes: experiment(s) that this experiment supersedes by virtue of being newer, better etc. than the one(s) it supersedes.

[5] https://github.com/ENCODE-DCC/encoded/blob/master/src/encoded/schemas/experiment.json
[6] https://github.com/ENCODE-DCC/encoded/blob/master/src/encoded/types/experiment.py

- internal_status: status of an experiment in the DCC process.
- pipeline_error_detail: explanation of why the experiment failed pipeline analysis.
- possible_controls: set of experiments that contain files that can serve as scientific controls for this experiment.
- assay_title: gets the preferred term in the ontology for the assay title based in assay_term_id.
- replication_type: indicates the replication type used for the assay.
- assay_slims: preferred term for the assay in the ontology.
- category_slims: preferred term for the category in the ontology.
- objective_slims: preferred name for the objective in the ontology.
- assay_term_id: ENCODE id referring to assay_term_name.
- type_slims: preferred term for the type in the ontology.
- related_files: to be removed in a future release after data cleanup.

When embedded JSON is requested extra metadata is added, experiment properties include all the properties of the dataset plus:

(Dots indicate the right term is contained inside the left one and plurals denote set of items)

- files (appends to the files subcategory in dataset):
    - platform: measurement device used to collect data
    - analysis_step_version.analysis_step

    - analysis_step_version.analysis_step.documents
    - analysis_step_version.analysis_step.documents.award
    - analysis_step_version.analysis_step.documents.lab
    - Analysis_step_version.analysis_step.documents.submitted_by
- replicates (appends to the replicates subcategory in dataset):
    - library
    - library.documents.lab
    - library.documents.submitted_by
    - library.documents.award
    - library.biosample.submitted_by
    - library.biosample.source
    - library.biosample.documents
    - library.biosample.organism
    - library.biosample.donor
    - library.biosample.donor.organism
    - library.biosample.genetic_modifications
    - library.biosample.genetic_modifications.target
    - library.biosample.genetic_modifications.modification_techniques
    - library.biosample.genetic_modifications.treatments
    - library.treatments

- And many more attributes of lab, award and submitted by for the library.biosample

I.4. Extracting the metadata

The acquisition of the metadata is done by downloading the JSON of an experiment provided by the ENCODE API and is requested with the frame "embedded" which is explained before. For the useful extraction of the metadata the experiment JSON has to be filtered as it contains much more information that is not related to the file being examined. Therefore sections that summarize data from multiple files together have to be consistent with the file being downloaded by filtering out the information related to other files. Also information as the legal documents for the laboratories or the different grants that were won for the development of the project are meaningless when referring to a file's metadata.

With the complete structure of the organization for the metadata is clearer which properties belongs to experiment, dataset or shared calculated and mixin properties.

The complete structure of the JSON attribute is known by now; but the usability of the metadata to be extracted depends on the final purpose of the processing of those metadata. More complex analysis of this source's information should be performed to allow a better integration of the ENCODE metadata with other sources' metadata.

I.5. More child nodes in ENCODE metadata

The following is a list of encode items mentioned before with their nested properties explained, obtained from the encode GitHub[7] directories schemas[8] and types[9]. These items are part of the ENCODE JSON metadata structure and these items are nested inside the structures explained before in this appendix:

---

[7] https://github.com/ENCODE-DCC/encoded
[8] https://github.com/ENCODE-DCC/encoded/tree/master/src/encoded/schemas
[9] https://github.com/ENCODE-DCC/encoded/tree/master/src/encoded/types

(Plurals denote set of items, unexplained items are assumed to be auto explanatory)

- user
    - email.
    - first_name.
    - last_name
    - lab: lab user is primarily associated with.
    - submits_for: labs user is authorized to submit data for.
    - groups: additional access control groups.
    - viewing_groups: the group that determines which set of data the user has permission to view.
    - other user info such as skype, telephone, etc.
- lab
    - name: short unique name for the lab, current convention is lower cased and .hyphen delimited of PI's first and last name (e.g. john-doe).
    - title: unique name for affiliation identification, current convention is comma separated PI's first & last name and institute label. (e.g. John Doe, UNI).
    - pi: principle investigator (user) of the lab.
    - awards: grants associated with the lab.
    - institute_label
    - other laboratory information such as address, institute_name, city, etc.
- award

  - title: grant name from the NIH database, if applicable.
  - name: official grant number from the NIH database, if applicable.
  - description.
  - start_date.
  - end_date.
  - url.
  - other grant information such as viewing_group, project, url.
- reference
  - reference_type: category that best describes the reference set.
  - organism.
  - software_used: list of software used to derive the reference file set.
- annotation
  - annotation_type: category that best describes the annotation set.
  - encyclopedia_version: version of the ENCODE encyclopedia to which this annotation belongs.
  - organism.
  - relevant_timepoint: time point for which the annotation is relevant.
  - relevant_timepoint_units.
  - relevant_life_stage: life_stage for which the annotation is relevant.

  - targets: for predictions of particular features (e.g. distribution of a histone mark), specify the predicted feature(s).
  - software_used: list of software used to derive the annotation calls.
  - supersedes: annotation set(s) that this annotation set supersedes by virtue of being newer, better etc. than the one(s) it supersedes.
- file
  - accession.
  - external_accession.
  - read_count: number of reads in fastq file.
  - fastq_signature: fastq file flowcell based unique signature to reference a file.
  - file_format.
  - file_format_type: files of type bed and gff require further specification.
  - file_format_specifications: text or .as files (document) the further explain the file format.
  - restricted: flag to indicate whether this file is subject to restricted access.
  - md5sum: md5sum of the file being transferred.
  - content_md5sum: MD5sum of the uncompressed file.
  - file_size.
  - platform: measurement device used to collect data.

- read_length: for high-throughput sequencing, the number of contiguous nucleotides determined by sequencing.
- mapped_read_length: the length of the reads actually mapped, if the original read length was clipped.
- mapped_run_type: mapped run type of the alignment file which may differ from the fastqs it is derived from.
- flowcell_details: for high-throughput sequencing, the flowcells used for the sequencing of the replicate.
- output_type: description of the file's purpose or contents.
- run_type: indicates if file is part of a single or paired run.
- paired_end: which pair the file belongs to (if paired end library).
- derived_from: files participating as inputs into software to produce this output file.
- controlled_by: files that control this file.
- supersedes: files that this file replaces.
- paired_with: file that corresponds with this file.
- dataset: experiment or dataset the file belongs to.
- replicate: experimental replicate designation for the file.
- assembly: genome assembly that files were mapped to.
- genome_annotation: genome annotation that file was generated with.
- submitted_file_name: local file name used at time of submission.

  - status: one among “uploading”, ”uploaded”, ”upload failed”, "format check failed", ”in progress", ”deleted", ”replaced", ”revoked", ”archived", ”released" and ”content error".
  - dbxrefs: unique identifiers from external resources.
  - step_run: run instance of the step the file was generated from.
  - content_error_detail: explanation of why the file failed the automated content checks.
- replicate
  - antibody: for Immunoprecipitation assays, the antibody used.
  - biological_replicate_number: data collection under the same methods using a different biological source, measuring the variability in the biological source.
  - technical_replicate_number: data collection under the same methods using the same biological source, measuring the variability in the method.
  - experiment: experiment the replicate belongs to.
  - library: nucleic acid library used in this replicate.
  - rbns_protein_concentration: for use only with RNA Bind-n-Seq replicates to indicate the protein concentration.
  - rbns_protein_concentration_units: unit for the dependant rbns_protein_concentration.
  - status: one among "in progress", "deleted", "released", "preliminary", "proposed", "archived", "revoked".
  - target.

  - dbxref: unique identifier from external resource.
  - organism: organism bearing the target.
  - gene_name: HGNC or MGI identifier for the target.
  - label: common name for the target including post-translational modifications, if any.
  - investigated_as: context(s) the target was investigated in.
  - status: one among “proposed”, “current”, “deleted”, “replaced”.
- document
  - document_type: category that best describes the document.
  - description: plain text description of the document.
  - urls: external resources with additional information to the document.
- platform
  - term_id: OBI (Ontology for Biomedical Investigations) ontology identifier for the measurement device.
  - term_name: OBI (Ontology for Biomedical Investigations) ontology term for the measurement device.
  - dbxrefs: unique identifiers from external resources.
  - url: external resource with additional information about the measurement device.
- analysis_step
  - name: unique name of the analysis step.

  - title: preferred viewable name of the analysis step, likely the same as the name.
  - analysis_step_types: classification of the software.
  - input_file_types: file types used as input for the analysis.
  - output_file_types: file types generated as output for the analysis.
  - qa_stats_generated: the QA statistics generated by the analysis.
  - parents: the precursor steps.
- analysis_step_run
  - analysis_step_version: reference to template step in pipeline
  - dx_applet_details: metadata capture from DNA Nexus applets.
  - status: one among “waiting”, “running”, “finished”, “error”, “virtual”.
- analysis_step_version
  - version: version of the analysis step.
  - analysis_step: reference to template step in pipeline.
  - software_versions: software version used in the analysis.
- source
  - description: plain text description of the source.
  - title: complete name of the originating lab or vendor.
  - name: auto generated from the title as lower cased and hyphen delimited.
  - url: external resource with additional information about the source.

- library
    - accession.
    - spikeins_used: datasets containing the fasta and the concentrations of the library spike-ins.
    - biosample: the biosample that nucleic acid was isolated from to generate the library.
    - product_id: the product identifier provided by the vendor, for nucleic acids or proteins purchased directly from a vendor (e.g. total RNA).
    - lot_id: the lot identifier provided by the vendor, for nucleic acids or proteins purchased directly from a vendor (e.g. total RNA).
    - source: the vendor, for nucleic acids or proteins purchased directly from a vendor (e.g. total RNA).
    - nucleic_acid_term_name: SO (Sequence Ontology) term best matching the molecule isolated to generate the library (e.g. 'RNA' for a total RNA library, even if that library is subsequently reverse transcribed for DNA sequencing).
    - dbxrefs: unique identifiers from external resources.
    - nucleic_acid_starting_quantity: starting amount of nucleic acid before selection and purification.
    - nucleic_acid_starting_quantity_units: units used for starting amount of nucleic acid.
    - extraction_method: short description or reference of the nucleic acid extraction protocol used in library preparation, if applicable.

    - fragmentation_method: short description or reference of the nucleic acid fragmentation protocol used in library preparation, if applicable.
    - fragmentation_date: The date that the nucleic acid was fragmented.
    - library_size_selection_method: short description or reference of the size selection protocol used in library preparation, if applicable.
    - lysis_method: short description or reference of the cell lysis protocol used in library preparation, if applicable.
    - crosslinking_method: short description or reference of the crosslinking protocol used in library preparation, if applicable.
    - size_range: measured size range of the purified nucleic acid, in bp.
    - strand_specificity: preparation of the library using a strand-specific protocol.
    - treatments.
    - depleted_in_term_name: SO (Sequence Ontology) term best matching the nucleic acid that was diminished from the library.
- organism
    - name: short unique name for the organism (e.g. 'mouse' or 'human').
    - scientific_name: genus species for the organism (e.g. 'Mus musculus').
    - taxon_id: NCBI taxon ID for the organism (e.g. 10090).
- biosample
    - accession.

- description: plain text description of the biosample. Do not include experiment details, constructs or treatments.
- constructs: expression or targeting vectors stably or transiently transfected (not RNAi or TALEN).
- rnais: RNAi vectors stably or transiently transfected.
- talens: TALEN constructs used to modify the biosample
- treatments.
- dbxrefs: unique identifiers from external resources.
- documents: documents that describe the biosample preparation.
- donor.
- organism.
- passage_number: calculating passage number, include passages from the source.
- depleted_in_term_name: UBERON (Uber Anatomy Ontology) term best matching the tissue(s)/body part(s) that were removed from the biosample.
- model_organism_mating_status: mating status of the animal.
- internal_tags: some biosamples are part of particular data collections.
- derived_from: biosample that the sample derives from via a construct or treatment.
- pooled_from: biosamples from which aliquots were pooled to form the biosample.

- part_of: biosample from which a discrete component was taken. That component is this biosample.
- host: biosample that was hosting this biosample.
- subcellular_fraction_term_name: GO (Gene Ontology) term name for cellular component that constitutes the biosample.
- phase: cell-cycle phase.
- transfection_type: persistence of the transfection construct.
- transfection_method: how the transfection was performed on the biosample to introduce the contruct or RNAi.
- culture_harvest_date: for cultured samples, the date the biosample was taken.
- culture_start_date: for cultured samples, the date the culture was started. For cell lines, the date this particular growth was started, not the date the line was established.
- date_obtained: for tissue samples, the date the biosample was taken.
- starting_amount: initial quantity of cells or tissue obtained.
- starting_amount_units.
- url: external resource with additional information about the biosample.
- model_organism_sex: model_organism_sex is not valid for a human biosample.
- mouse_life_stage: mouse_life_stage is not valid for a human biosample.
- fly_life_stage: ly_life_stage is not valid for a human biosample.

  - fly_synchronization_stage: stage at which flies were synchronized.
  - post_synchronization_time: use in conjunction with fly_sychronization_stage or worm_synchronization_stage to specify time elapsed post-synchronization.
  - post_synchronization_time_units: use in conjunction with post_synchronization_time to specify time elapsed post-synchronization.
  - post_treatment_time: use in conjunction with treatment to specify time elapsed post-treatment.
  - post_treatment_time_units: use in conjunction with treatment to specify time elapsed post-treatment.
  - worm_life_stage: worm_life_stage is not valid for a human biosample.
  - worm_synchronization_stage: stage at which worms were synchronized.
  - model_organism_age: age or age range of the model donor organism when biological material was sampled.
  - model_organism_age_units: model_organism_age_units are not valid for a human biosample.
  - model_organism_health_status: model_organism_health_status is not valid for a human biosample.
  - status: one among "in progress", "deleted", "replaced", "released", "re-voked", "preliminary", "proposed"
- donor
  - accession.
  - organism: organism of the donor.

    - url: an external resource with additional information about the donor.
    - internal_tags: some donors are part of particular data collections.
- genetic_modification
    - url: external resource with additional information about the modification.
    - target: name of the gene whose expression or product is modified by the genetic modification.
    - modified_site: genomic coordinates of the modification (assembly, chromosome, start, end).
    - description: plain text description of the genetic modification.
    - modification_type: type of the genetic modification.
    - purpose: purpose of the genetic modification.
    - zygosity: zygosity of the genetic modification.
    - modification_techniques: genetic modification technique(s)/tool(s) used to perform the modification.
    - treatments: treatment(s) used to perform the genetic modification.
- modification_techniques
    - dbxrefs: unique identifiers from external resources.
- antibody_lot
    - accession.
    - lot_id: lot identifier provided by the originating lab or vendor.

  - lot_id_alias: lot identifiers for this lot deemed to be exactly the same by the vendor.
  - antigen_description: plain text description of the antigen used in raising the antibody (e.g. amino acid residue locations of the antigen).
  - antigen_sequence: amino acid sequence of the antigen.
  - clonality: diversification of the immune cell lineage to make the antibody.
  - dbxrefs: unique identifiers from external resources.
  - host_organism: organism the antibody was grown in.
  - isotype: class of antibody (e.g. IgA, IgD, IgE, IgG or IgM).
  - purifications: purification protocols used to isolate the antibody.
  - targets: name of the gene whose expression or product is the intended goal of the antibody.
  - url: external resource with additional information about the antibody.
- treatment
  - lab: lab associated with the submission.
  - documents: documents that describe the treatment protocol.
  - dbxrefs: unique identifiers from external resources.
  - amount: amount of treatment applied.
  - amount_units.
  - duration.
  - duration_units.

  - temperature.
  - temperature_units.
  - biosamples_used: biosamples used in this treatment.
  - antibodies_used: antibodies used in this treatment.
- talen
  - name: name of the TALEN construct.
  - description: plain text description of the TALEN construct.
  - url: external resource with additional information about the construct.
  - vector_backbone_name: cloning vector used to make the construct. E.g. PEGFP (delGFP-TAL2-truncNLS)
  - RVD_sequence: The repeat variable diresidue sequence. E.g. NI,NG,NI,HD,NG,NN,NG,NG,NN,HD,NI,NI,NI,NI,NM,HD,HD,NG.
  - target_sequence: target genome sequence recognized by the TALE domain.
  - talen_platform: TALEN platform used to make the construct. E.g. Golden Gate.
  - target_genomic_coordinates: genomic coordinates where the TALEN cuts.
  - pairings: list of possible pairings with other TALENs.
- rnai
  - rnai_type: classification of the interfering RNA (e.g. shRNA or siRNA).
  - url: external resource with additional information about the RNAi construct.

  - target: name of the gene whose expression or product is modified by the RNAi construct.
  - vector_backbone_name: name of the vector backbone used for the construct.
  - rnai_sequence: sequence of the inhibitory RNA.
  - rnai_target_sequence: genomic sequence targeted by the RNA.
- construct
  - construct_type: type of sequence expressed from the construct.
  - description: plain text description of the construct. May include backbone name, description of the insert or purpose of the construct.
  - url: external resource with additional information about the construct.
  - target: name of the gene whose expression or product is modified by the construct.
  - tags: recombinant tags in the construct.
  - vector_backbone_name: name of the vector backbone used for the construct.
  - genomic_integration_site: genomic coordinates where construct is integrated, if known.
  - insert_sequence: DNA sequence inserted into the vector backbone.
  - insert_genome_coordinates: genomic coordinates of the insert. e.g. GRCh38 or dm6.
  - promoter_used: name of the gene that the promoter regulates natively.

- promoter_genome_coordinates: genomic coordinates of the promoter. Use NCBI assembly version:chromosome number:nucleotide range (e.g. WBcel235:III:1433720-1434340).
- promoter_position_relative_to_target: relative distance of promoter sequence in the construct upstream of the target gene TSS.

Appendix J. Console manual for GMQLImporter

To run GMQLImporter is done by command line by invoking the following line:

(Assuming the compiled Scala code is in GMQLImporter.jar)

java –jar GMQLImporter.jar *configuration_xml_path gmql_conf_folder*

where *configuration_xml_path* is the location for the configuration file and *gmql_conf_folder* contains the path to the folder with corresponding variables to start GMQLRepository service.

If *log* added at the end, it will show the number of executions already performed and stored in the database.

If added *log –n* where n is the n-th past execution, the statistical summary for that run is shown. Also multiple runs can be requested at the same time by separating them with comma. As example log -1, 2, 3 will show (if they exist) the last execution's statistics, plus the statistics for the 2 executions done before, it notifies also if a number is not valid and will not show that specific information if so (as example: if 5 executions are already run, using log -6 will not show statistics but will tell the user that run does not exist).

If added *–retry* GMQLImporter tries to download the failed files on the local datasets.

Appendix K. Metadata replacement for ENCODE in GMQLImporter

This is an example of metadata replacement, extract from the one used for ENCODE. This file is referred from the main configuration XML to replace by using regular expressions metadata field names in the transformation process:

```
<metadata_replace_list>
<metadata_replace>
<regex>^file\|accession$</regex>
<replace>File accession</replace>
</metadata_replace>
<metadata_replace>
<regex>^accession$</regex>
<replace>Experiment accession</replace>
</metadata_replace>
<metadata_replace>
<regex>^file\|file_type$</regex>
<replace>File format</replace>
</metadata_replace>
<metadata_replace>
<regex>^file\|output_type$</regex>
<replace>Output type</replace>
</metadata_replace>
<metadata_replace>
<regex>^assay_title$</regex>
<replace>Assay</replace>
</metadata_replace>
<metadata_replace>
<regex>^biosample_term_id$</regex>
<replace>Biosample term id</replace>
</metadata_replace>
<metadata_replace>
<regex>^biosample_term_name$</regex>
<replace>Biosample term name</replace>
</metadata_replace>
</metadata_replace_list>
```